PLANNING PRACTICAL PATHS IN HIGH DIMENSIONAL SPACE

JING YANG

A DISSERTATION SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

GRADUATE PROGRAM IN COMPUTER SCIENCE
YORK UNIVERSITY
TORONTO, ONTARIO
MARCH 2014

© Jing Yang 2014

Abstract

Unlike traditional manipulator robots which tend to have small numbers of degree of freedom (DOF), tentacle robots utilize redundant DOFs in order to enhance their ability to deal with complex environments and tasks. However, it also makes the planning and control of such devices extremely difficult. One of the fundamental tasks robots have to perform is planning their motions while avoiding collisions with obstacles in the environment, which is known to be PSPACE-complete in the robot's DOF. As a consequence heuristic sampling-based approaches have been developed to solve high-dimensional real-world path planning problems. A shortcoming of the current sampling-based algorithms is that they can obtain highly non-optimal solutions since they rely upon randomization to explore the search space. Although these planners may find a valid solution, the solutions found are often not practical in that they do not take into account soft application-specific constraints.

This thesis integrates soft constraints in addition to the basic geometric or hard constraints in the general path planning process for high DOF robots. The prac-

ticality of paths are formulated based on the notion of soft constraints found in the Planning Domain Definition Language 3 (PDDL3). A range of optimization strategies are developed targeted towards user-preferred qualities by integrating soft constraints in the pre-processing (i.e. sampling), planning and post-processing phases of the sampling-based path planners. An auction-based resource allocation approach coordinates competing optimization strategies. This approach uses an adaptive bidding strategy for each optimizer and in each round the optimizer with the best predicted performance is selected. This general coordination system allows for flexibility in both the number and types of the optimizers to be used. Experimental validation with real and simulated tentacle robots demonstrate the effectiveness of the approach.

Acknowledgements

Among people that have contributed to the completion of my doctoral research and dissertation, I'd first like to thank my supervisor, Professor Michael Jenkin, who has been a wonderful mentor throughout my Ph.D research. His broad knowledge in Robotics inspired me many times when I was stuck. I couldn't have done it without his help. He is also a fun Professor with various research interests, which greatly opened my eyes. What I learned from him will benefit me for my whole life.

I'd also like to thank my co-supervisor, Professor Patrick Dymond, who led me into the palace of scientific research. He has been very supportive in various ways and I appreciate the flexibility he provided, which made my research career a pleasant journey instead of a tedious work.

At last, I'd like to thank my family. My newborn baby girl has been my main motivation to finish my thesis in time as I am dying to dress her as a little Doctor on my convocation. Bo, my love of life, thank you for being the bad guy to push me so hard. I know you meant well.

Table of Contents

1 Introduction

1.1 Background

The development of algorithms that get a robot from 'here' to 'there' in the presence of obstacles in the environment is perhaps **the** problem that defines mobile robotics [69]. This basic path planning problem is also called the generalized movers' problem [100, 20] and is closely related to a range of search-like problems. In the basic version of the problem it is assumed that the robot is a simple rigid object and that the environment consists of static rigid obstacles and the goal is to find a continuous motion of the robot from the start to the goal while remaining in the environment's free space (Figure 1.1 illustrates two instances of the basic path planning problem). Figure 1.1(a) shows the path found to move a simple rectangle among polygonal obstacles in a two-dimensional workspace. This type of problem is easily addressed using traditional techniques. The path planning problem becomes somewhat more difficult as the dimensionality of the problem and complexity of the environment increases. This is illustrated in Figure 1.1(b) which shows the 3D

motion of a complex object in a complex environment with narrow passages. It has been proven that the basic path planning problem is PSPACE-complete in the dimensionality of the problem [100, 20]. That is, it is not believed that there exists an algorithm that solves the path planning problem in time that is polynomial in the dimensionality of the problem. The best known complete algorithm, Canny's roadmap algorithm [20], takes time exponential in the number of the degrees of freedom (DOFs) of the robot.

There is an extremely large literature on robot path planning and indeed for a range of different versions of the task the problem can in many ways be considered solved [69]. For small dimensional problems, with reasonably small state spaces, sophisticated algorithms exist that will find paths from the start to the goal state in a reasonable amount of time. As the size or dimensionality of the problem becomes larger, however, complete algorithms are no longer possible and probabilistic solutions become appropriate. A large number of probabilistic solutions now exist for many of these problems as well, including sample-based algorithms such as the Probabilistic Roadmap Method (PRM) [60] and the Rapidly exploring Random Tree (RRT) [6] algorithm. (Chapter 2 provides a survey of such approaches and their variants.) Since it can be hard to plan a path for robots with many DOFs, most methods for high DOF tasks aim at finding any solution within a reasonable time, rather than searching for an optimal solution or even one that minimizes (or

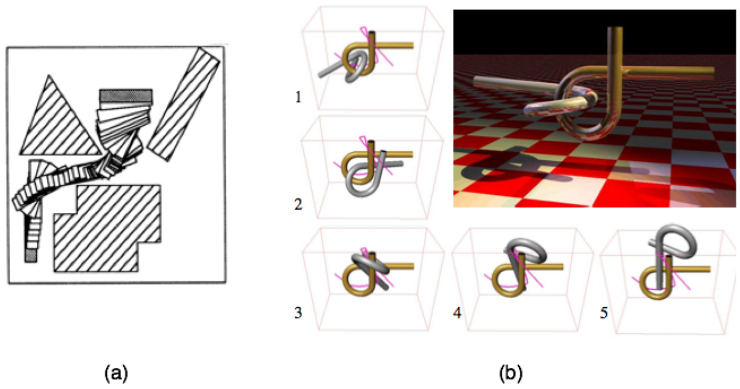


Figure 1.1: A simple example and a difficult one of the basic path planning. (a) The path of a rectangle among polygonal obstacles in a two-dimensional workspace is displayed. Figure reprinted from [69]. (b) The Alpha 1.0 Puzzle which is a well-known difficult path planning problem due to the narrow passage difficulty. The puzzle consists of two tubes, each twisted into an alpha shape; one tube is the obstacle and the other the moving object (robot). The objective is to separate the intertwined tubes. Figure reprinted from [71].

reduces) one or more soft constraints. Neither standard PRM nor RRT are asymptotically optimal, i.e. the cost of the solution returned by the algorithm is not guaranteed to converge to the optimal cost as the number of samples increases [58]. As a consequence optimality cannot be generated by simply sampling more densely, nor can the solution be improved by simply dedicating more resource to the problem. Sampling-based planners typically seek to find **any** path from the start to the goal and ignore issues related to optimality and the like. This can result in highly 'non-optimal' solutions to the path planning problem.

1.2 Motivation

Although robot path planning problems such as those considered in Figure 1.1 have a certain theoretical interest, the path planning problem arises in realistic robot applications as well. To take a timely example, consider the growing requirement to ascertain the state of nuclear systems [2, 4, 17]. There are many challenges associated with nuclear maintenance. First, the site is extremely dangerous, so human access can be impossible or at least very limited. Second, the inspection or repair may be required to be done in an extremely confined space, so that it can be difficult to deliver the tool or the sensor to the work site. Given these constraints access to these confined spaces requires a device with a high number of DOFs (joints) and it can be difficult to control a device that has the necessary flexibility.

Figure 1.2(c-d) shows a hyper-redundant robotic arm (also known as a tentacle or snake robot) developed by OC Robotics designed to address the problem of robotic repair of devices such as nuclear reactors. The robot is designed to be very flexible. By adjusting the bends (joints) in the robot the device can be directed to different task spots within the environment. High flexibility requires a large number of DOFs.

There are a number of different properties any robot should exhibit, but safety is perhaps the most important issue for operations in nuclear sites. The operators must ensure that the robot follows a collision free path once it is deployed in the site. As a tentacle robot has many DOFs there are many possible ways for the robot to reach a given goal. Given the complexity of the path planning task for tentacle robots, typically these devices are driven manually using a nose-following strategy. The operator drives the robot by controlling its end-effector (its nose) and the computer drives the rest of the robot by following it. The path must be correct (the robot must remain in the free workspace), but it must also be practical in that it should work to reduce domain-specific constraints such as remaining away from objects, reaching desired positions precisely or using a small amount of energy. These later constraints can be thought of as *soft* in that the robot should work to reduce the cost associated with these factors, but that solutions with a high soft cost are still valid, just not as preferred as those with a low soft cost. How should

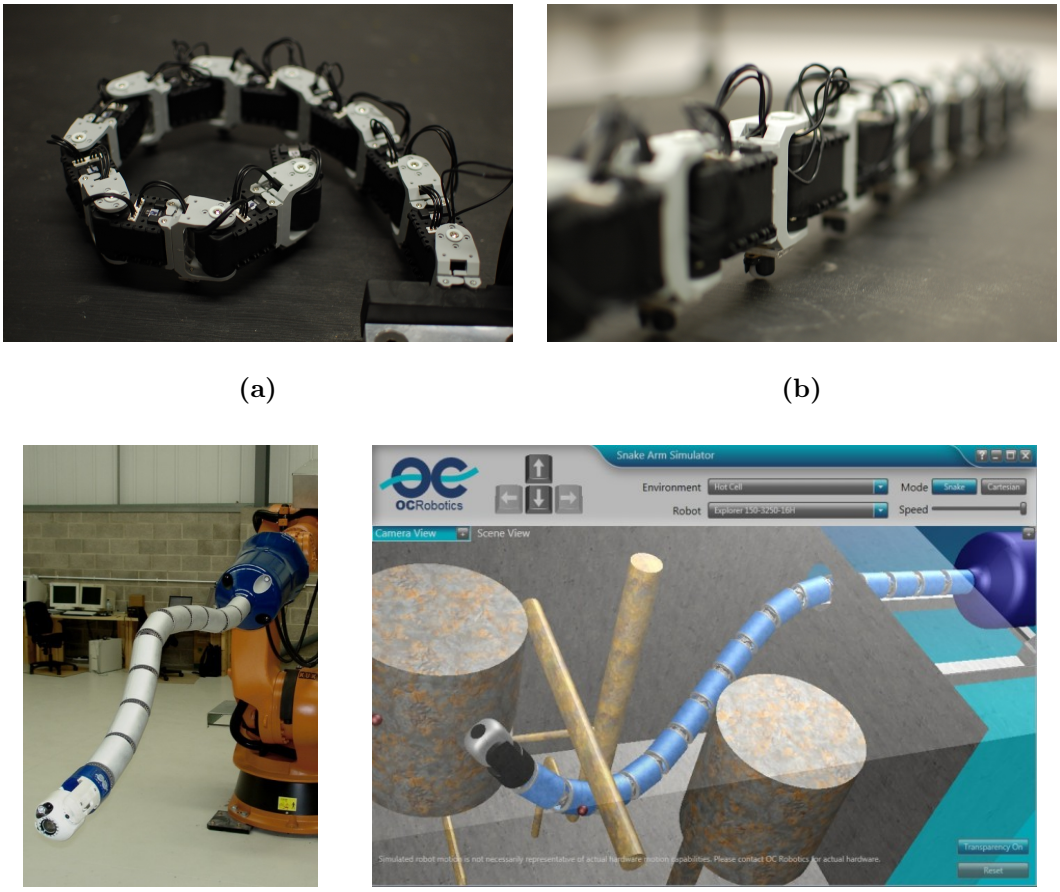
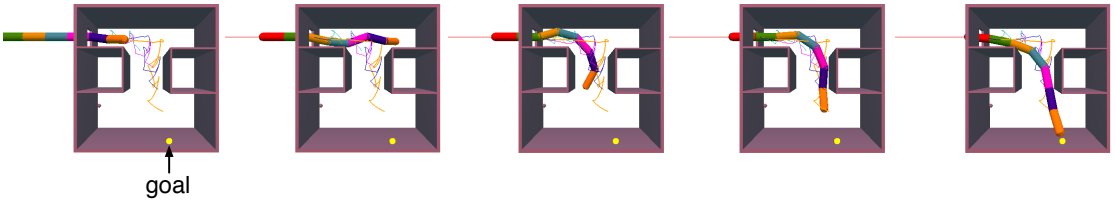
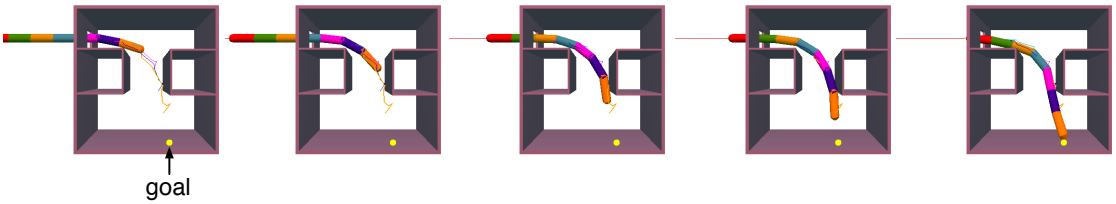


Figure 1.2: Tentacle robots. (a) and (b) are images of a planar tentacle robot created from ten Dynamixel AX-12 servos. (c) and (d) are real and simulated images of OC Robotics 3D tentacle robot. The pictures (c) and (d) are ©OC Robotics and are used with permission.

soft constraints be integrated into the planning problems? Furthermore, how can multiple constraints be considered in the path planning? These are the questions



(a)



(b)

Figure 1.3: Comparison of paths of the snake robot computed by (a) the PRM algorithm and (b) the soft constraint-based PRM developed in this work. The robot must be inserted into the workcell so that its end-effector reaches the yellow dot at the bottom.

addressed in this work. This thesis presents a framework within which plans can be developed for high DOF robotic systems that meet the hard constraints of path feasibility while at the same time seek to reduce the cost of a collection of domain-specific soft constraints. This framework is intended to be robot independent, but the approach is grounded in the capabilities and tasks associated with high DOF tentacle robots such as the ones shown in Figure 1.2.

In order to motivate this problem further, it is illustrative to consider some of the issues encountered in performing path planning for high DOF robots. Consider path planning for the 7-link snake robot with a mobile base shown in Figure 1.3. Here the robot's base translates in one dimension and each joint consists of two DOFs – pitch and yaw. The robot has 15 DOFs in total, i.e. the path planning for the robot can be viewed as a search problem in a 15-dimensional abstract space. Figure 1.3(a) shows a sample path for this robot computed by a PRM. Although the planner is able to find a path in a timely manner and the path found is executable (the robot gets to the goal without hitting any obstacles), the planned path contains many extra motions (zig-zags and unnecessary rotations of the robot) that result from the stochastic nature of the planner. In addition, the planned path of the robot causes the robot to approach very close to obstacles in the environment while it navigates the space, which is a safety issue (this is especially important for nuclear robots). Developing algorithms that find both correct and safe paths for this type of robot is the goal of this work. Figure 1.3(b) illustrates the kind of path that is desired. Here again the path is correct, but now in addition to meeting the hard constraint of correctness, the path is both smooth and more centered in the free space of the environment.

1.3 Objectives of the dissertation

With the development of sampling-based algorithms and their application in practice, the focus has shifted to considering the quality of the path obtained [98, 38, 63, 36, 119, 7, 107, 58]. Much work has been done to augment sampling-based algorithms to find short paths or paths with high clearance, but to the author's knowledge there does not exist a general solution for any type of practical requirements. In this work, we introduce the concept of the generalized path practicality which can be expressed by a soft constraint cost function and explore how to integrate this concept in the planning process of the sampling-based motion planners.

This thesis considers real soft constraints in addition to the basic geometric or hard constraints in the general path planning process for high DOF robots so as to find paths that meet the requirements of high-level user preferences. There are a number of different formalisms for describing hard and soft constraints within a planning framework [40]. This work uses the notation of soft constraints found in the Planning Domain Definition Language 3 (PDDL3) [40] to formally describe such practical issues. The first contribution of this thesis is to formulate the practicality of paths in terms of soft constraints as defined in PDDD3.

One common way of taking these constraints into account is to use an appropriate controller that takes the path identified by the path planner as input and

then integrates the soft constraints while following the path [16, 65]. There are many issues with this approach. Perhaps most critically the paths produced may be infeasible for a real robot. For example, following the path produced may require that the robot move extremely slowly in order to minimize the influence of dynamics and other physical constraints. These controllers are also system specific, and it can be very hard to develop a good 'general' controllers or to know which controller to use for which task.

A more general approach is to augment sampling-based path planning with mechanisms to provide paths that are both correct but that also optimize soft constraints. Such augmentation could take place at different points in the path planning algorithm. There are several existing methods, such as shortcutting method for short path length, retraction method for high clearance, etc. Each of these techniques is designed to optimize a specific soft constraint cost, but not a general one or a combination of several soft constraints. To overcome this limitation, this thesis categorizes soft constraints into node-level, edge-level and global-path-level and develops a range of optimization strategies targeted user-preferred qualities by integrating soft constraints in the pre-processing (i.e. sampling), planning, and post-processing phases of the sampling-based path planners.

Several optimization strategies are developed, which have unique strengths in addressing different types of soft constraints. However, it is difficult to determine

a priori which optimizer is most suitable at a certain time for a certain problem. This work develops a novel auction-based approach that allows multiple optimizers to compete in a 'market' for computation resources. This approach is online and adaptive and the optimizer with the best predicted performance is dynamically selected as the process goes on.

In order to validate our algorithm, we have applied it to several benchmark puzzle problems. The need to properly represent and use soft constraints is particularly important for redundant DOF robots such as tentacle devices. Further evaluations are provided using real and simulated tentacle robots. Our experiments show that this algorithm outperforms alternatives and improves performance on real world robotic systems.

1.4 Structure of the thesis

This thesis is structured as follows: Chapter 2 reviews the current literature in sampling-based path planning with particular emphasis on algorithms and approaches that not only seek paths within the environment but which also permit other soft constraints to be considered within the planning process. Chapter 3 formulates the practicality of paths in terms of soft constraints, and path planning strategies are developed to find paths of user-preferred qualities based on this formalism. In Chapter 4, an auction-based approach is developed to combine and

coordinate multiple optimizations so that their strengths are preserved and computational resource is allocated dynamically among them. Chapter 5 compares paths obtained with a practicality-aware planning approach and existing path planners to different test environments using both real and simulated tentacle robots. Finally Chapter 6 summarizes the work and provides possible directions for future research.

1.5 Previously published material

Much of the work described in this dissertation has been previously published in refereed conferences and journals: [127], [125], or is currently under review for publication [128]. The basic concept of separating soft from hard constraints described in chapter 3 of this thesis is introduced in [127] and [125]. The auction-based resource allocation mechanism described in chapter 4 is the subject of [128].

2 Related work

Planning practical paths for complex (high DOF) robots is a difficult process. In this work planning practical paths is defined as the task of developing paths that meet the hard constraints of planning coupled with the desirability of minimizing soft constraints associated with path practicality. This chapter reviews salient work in these two areas. Optimization of soft constraints requires the expenditure of optimization resources among competing optimization strategies, and the basics of optimization allocation through an auction process are reviewed at the end of the chapter.

2.1 The robot path planning problem

In its simplest form, path planning is a purely geometric problem: given a description of the geometry of an object and a static environment, a path planner is an algorithm that finds a sequence of motions that enables the object to move from an initial configuration to a specified goal configuration, avoiding obstacles, while

satisfying any constraints specified on its motions [69].

The path planning literature is unified around the concept of a configuration space as described by Latombe [69]. A configuration space is defined by the set of possible transformations that can be applied to the robot. With this abstraction the problem of finding a path for a high dimensional robot in its workspace is reduced to finding a path for a point robot in a higher-dimensional configuration space. Once the configuration space is clearly understood, many path planning problems that appear different in terms of geometry and kinematics can be addressed by the same planning algorithms.

The path planning problem can be approached using many types of methods. Traditional combinatorial path planning approaches are perhaps most the studied branch of planning. These methods for few-DOF robots can be very efficient, even in large and complex environments encountered in practical problems. For many-DOF problems and more complex problems, traditional planning is inappropriate and randomized or probabilistic algorithms are needed. Of particular interest here is the development of sampling-based planners, which have shown power in solving complex path planning problems for serial manipulators and in other situations where higher-dimensional planning is required.

2.1.1 Extensions of the basic path planning problem

The basic path planning problem is a purely geometric problem where all information about the robot and its workspace are known. It assumes that the robot has perfect knowledge about its own geometry and about the workspace's geometry. However, no errors or uncertainty about the robot's location or movement are assumed. These assumptions can limit the practicality of solutions found. In other words, it may be quite difficult to reduce an actual robotic problem to the basic path planning problem [69]. Here we discuss several important extensions to the basic problem.

Dynamic constraints One crucial extension towards more physical realism is to take into account dynamic constraints. A real robot is not a "free-flying" object. It has motor limitations that impose bounds on its maximum velocity and acceleration [117, 110, 107, 75]. Car-like robots, for example, can move forwards (and/or backwards) but not sideways. This is a classic example of a non-holonomic constraint. Such dynamic constraints can significantly increase the complexity of path planning.

Changing environment A series of extensions in the literature remove the assumptions of a static environment. The workspace may include moving obstacles.

For example, a known environment might contain objects with known trajectories. In the best case, the obstacles execute known motions and information about the robot's allowable velocity and/or acceleration is also available [117]. If the movements of obstacles are not known beforehand and/or dynamic constraints apply, the problem becomes significantly harder [69, 117, 77].

Multiple robots Another extension involves planning with multiple robots, where all robots move simultaneously, while mutual collisions and collisions with obstacles should be avoided. This problem is handled by centralized or decoupled methods [69]. Centralized methods compute the paths for all robots simultaneously in a joint $\mathcal{C}$-space [103, 101, 16]. These methods can be complete but generally are computationally expensive. Decoupled methods compute a path for each robot independently and then coordinate the resulting motions [101, 93]. They are often much more efficient than centralized methods but the resulting paths can be far from optimal.

Uncertainty Dealing with uncertainty is important for path planning in real-world applications. In many cases, the state of the world is derived from sensors that are prone to noise and error. In addition, the underlying control algorithms that are responsible for the physical motion of a robot also introduce error in the execution of motion plans once they have been computed [18, 69, 92]. We often do

not worry about these imperfections because they are small relative to the tolerance of the task being performed, but this is not always the case. The more incomplete the prior knowledge, the less important the role of planning. To solve the motion planning problem with bounded uncertainty, the robot may interweave planning and execution monitoring activities such that the uncertainty can be reduced and the robot moves toward the goal [69].

2.1.2 Complexity

Theoretical results show that a complete planner may require time exponential in the number of DOFs of the robot. In 1979, Reif gave the first lower bound for path planning [100]:

Planning a free path for a robot made of an arbitrary number of polyhedral bodies connected together at some joint vertices, among a finite set of polyhedral obstacles, between any two given configurations, is a PSPACE-hard problem.

Later, Schwartz and Sharir [103] proposed a complete general-purpose path planning algorithm, which provides an upper bound on path planning complexity: *A free path in a configuration space of any fixed dimension d, when the free space is a set defined by n polynomial constraints of maximal degree m, can be computed by an algorithm whose time complexity is exponential in d and polynomial in both n ("geometrical complexity") and m ("algebraic complexity").*

This algorithm partitions the d-dimensional configuration space into simple pieces or cells, whose connectivity can easily be determined. Unfortunately the tools they used, namely cell decompositions and Sylvester resultants, have poor complexity bounds for problems in high dimensions. As the dimension d increases, both the number of cells, and the degree of the algebraic surfaces enclosing them, grow doubly exponentially.

Dropping from double to single exponential growth, the roadmap algorithm described in J. Canny's Ph.D. thesis [20] solves the problem in exponential time in d. A major problem with this algorithm is that even after knowing the connectivity of the roadmap, it is a considerable challenge to obtain a parameterization of each curve on the roadmap. For this and many other technical reasons, no general implementation of Canny's algorithm appears to exist [69, 71].

Canny's algorithm established an exponential algorithm for path planning for arbitrary dimensions. In realistic cases where the problem is more complex it is not known if the problem is even decidable except for specific cases [24]. Some examples with proven lower bound include the following:

- *Shortest path problem.* The shortest path problem is defined in two or three dimensions as the problem of finding the shortest obstacle-avoiding path between two given points under an L^p metric, with polygonal or polyhedral obstacles respectively. The L^p norm of a vector $v = (x, y, z)$ is defined by

$\|v\|_p = \sqrt[p]{|x|^p + |y|^p + |z|^p}$ and the L^p distance between vectors u and v is $\|u - v\|_p$. While the Euclidean shortest path has efficient solutions in two dimensions, the three-dimensional problem is much more difficult. In [20], finding a shortest path under any L^p metric in a three-dimensional polyhedral environment is proven to be NP-hard.

- *Multiple rectangles.* Suppose the environment consists of arbitrarily many rectangles in an empty rectangular workspace. Each rectangle can only translate with its sides parallel to the sides of the workspace boundary. The problem is to plan the coordinated motion of the rectangles between two given configurations, so that they do not intersect. This problem, which is equivalent to planning a path in the configuration space of the multi-bodied robot consisting of all the rectangles, is PSPACE-complete [46, 47].

- *Dynamic motion planning.* Planning the motion of a rigid object translating without rotation in three dimensions among arbitrarily many moving obstacles that may both translate and rotate is PSPACE-hard if the velocity modulus of the object is bounded, and an NP-hard problem otherwise [99]. A simpler problem – planning the motion of a point in the plane, with bounded velocity, among arbitrarily many convex polygonal obstacles moving at constant linear velocity without rotation – has also been shown to

be NP-hard [20].

These theoretical versions of the path planning problem have helped in calibrating the complexity of path planning and understanding its combinatorial nature [69]. With the development and deployment of robots in various fields, researchers have been required to develop practical path planners targeted towards realistic applications. [70] summarizes some of the important achievements in the development of path planning techniques and discusses the problems regarding computational issues.

2.2 Path planning methods

The path planning problem can be approached in many different ways. Traditional combinatorial path planning approaches are perhaps the most studied branch of planning. These methods can be very efficient for few-DOF robots, even in large and complex environments encountered in practical problems. For many-DOF problems and more complex problems, traditional planning is inappropriate and randomized or probabilistic algorithms are needed. Of particular interest here is the development of sampling-based planners, which have proven efficient in solving complex path planning problems for serial manipulators and in other situations where higher-dimensional planning is required.

2.2.1 Traditional combinatorial path planning algorithms

Considering the abilities of the robot and the structure of the environment, traditional combinatorial path planning methods deterministically pre-compute a sparse graph representation of the environment and then search for a solution in the graph (a summary of these approaches can be seen in [69]). The graph must be an exact representation of the configuration space, i.e. these planners return a path whenever one exists in the graph and indicate none exists otherwise. As the size of the problem grows such approaches become infeasible – it becomes impractical to actually decompose the space into a graph and the size of the resulting graph becomes so large as to prohibit effective searching on it. Two key combinatorial planners are the roadmap and cell decomposition approaches.

2.2.1.1 Roadmap

The roadmap approach is based on the formal concept of configuration space. The roadmap $\mathcal{R} = (N, E)$ is a graph, i.e. a network of one-dimensional curves, capturing the connectivity of $\mathcal{C}_{free}$, where N is a set of configurations of $\mathcal{A}$ appropriately chosen over $\mathcal{C}_{free}$, E is a set of (simple) paths; an edge (a, b) corresponds to a feasible path connecting the configuration a and b. Once $\mathcal{R}$ is constructed, each motion-planning query is processed by first connecting c_{init} and c_{goal} to two nodes

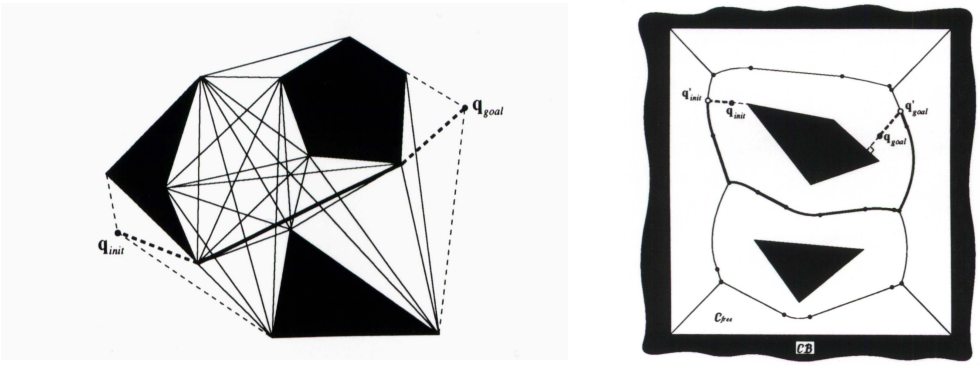


Figure 2.1: Examples of roadmap approaches applied on two-dimensional configuration spaces with polygonal *C-obstacles*. The shaded areas represent obstacles. The solid lines are the edges of $\mathcal{R}$. The dotted lines connect c_{init} and c_{goal} to $\mathcal{R}$. (a) In the Visibility Graph (VG), the vertices of $\mathcal{C}_{obst}$ are connected by a link if the straight line segment joining them does not intersect the interior of $\mathcal{C}_{obst}$. (b) The Generalized Voronoi Diagram (GVD) is a network of line segments and parabolic curves, which are the set of points equidistant from at least two *C-obstacles*. The path generated by this method keeps the robot as far away from the obstacles as possible, unlike the VG. Figure reprinted from [69].

in $\mathcal{R}$ and then searching the roadmap for a path connecting these two nodes (e.g. using a graph search algorithm such as Breadth-First search, Dijkstra's shortest path algorithm or A* search).

A well constructed roadmap $\mathcal{R}$ should have these two characteristics [102]: (1) Any configuration in $\mathcal{C}$ can be easily connected to $\mathcal{R}$; and (2) Each connected component in $\mathcal{R}$ corresponds to one and only one connected component in $\mathcal{C}$. Classical examples of roadmaps include Visibility Graphs [78] (Figure 2.1(a)), Generalized Voronoi Diagrams [91] (Figure 2.1(b)).

Visibility Graph (VG) The definition of the VG is that its nodes share an edge if they are visible to each other, and that all points in the $\mathcal{C}_{free}$ are visible to at least one node on the VG (see Figure 2.1(a)). There has been considerable study of VGs and algorithms to compute them. For planar environments the algorithm described in [73] computes the VG in $O(n^2 \log n)$ time complexity, where n is the number of nodes in the graph. Running time was improved to $O(n^2)$ in [120]. In [41], an output-sensitive algorithm which runs in $O(n \log n + k)$ time, where k is the number of arcs in the VG is presented. In the worst case, k is of n^2. Such optimal algorithms compute the VG of *all* vertices of the obstacles.

Generalized Voronoi Diagram (GVD) The GVD (see Figure 2.1(b)) is the set of points where the distance to the two closest obstacles is the same, which can

be used to extract high-clearance paths [25]. The GVD for a robot with n DOFs is a collection of k-dimensional ($0 \leq k < n$) geometric features (e.g., surfaces, curves and points). These features are connected if the free space in which the robot operates is connected [26]. Hence, the GVD is a complete representation for path planning purposes.

Vleugels and Overmars [118] approximate the GVD by applying spatial subdivision and isosurface extraction techniques. Although the calculations are easy and robust, and can be generalized to higher dimensions, the technique only works for disjoint convex sets and consumes an exponential amount of memory. Another approach, proposed in [82] constructs the GVD incrementally by finding the maximal inscribed disks in a two dimensional discretized workspace. Although this algorithm is also extensible to higher-dimensional problems, it suffers from the same drawbacks as the preceding algorithm.

2.2.1.2 Cell decomposition

The idea behind cell decomposition methods is to decompose $\mathcal{C}_{free}$ into simple regions, called cells, such that a path between any two configurations in a cell can be easily generated. Paths are then constructed between adjacent empty cells. A cell is marked as 1 (dark) if it is occupied by an *C-obstacle*, or else marked as 0 (white) if it is free. An undirected connectivity graph is constructed to represent

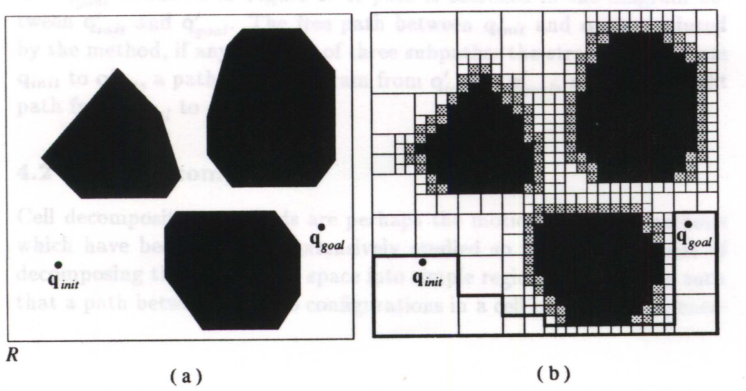


Figure 2.2: An illustration of the quadtree decomposition method in a two-dimensional configuration space. (a) The configuration space is bounded by a rectangle and contains three polygonal *C-obstacles*. (b) The rectangle is divided into four identical rectangles. If the interior of a rectangle lies completely in $\mathcal{C}_{free}$ or in $\mathcal{C}_{obst}$, then it is not decomposed further. Otherwise, it is recursively decomposed into four rectangles until some predefined resolution is attained. A channel extracted from this decomposition is shown in bold contour. Figure reprinted from [69].

the adjacency relation between the cells. This graph can then be searched to find a sequence of cells connecting the two cells that contain c_{init} and c_{goal}, from which the final free path is extracted.

The cell decomposition can be exact or approximate. Exact cell decomposition methods decompose $\mathcal{C}_{free}$ into cells whose union is exactly $\mathcal{C}_{free}$, but approximate

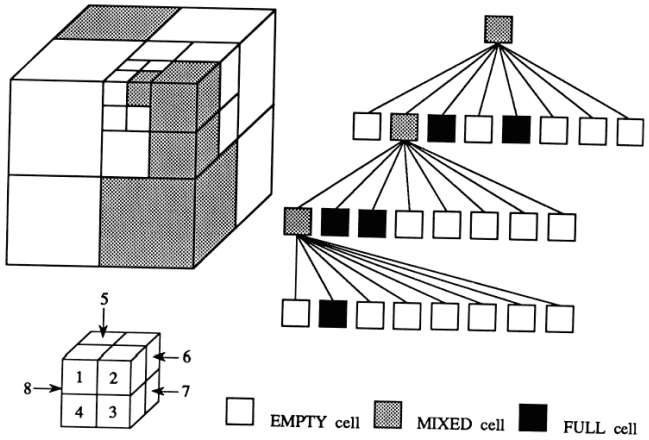


Figure 2.3: Octree decomposition is very similar to the quadtree decomposition. Instead of rectangles, each node in octree is a cube. Each cube can be recursively decomposed into eight cubes. Figure reprinted from [69].

ones do not. Examples of exact methods include trapezoidal maps and triangulation maps (tetrahedralization maps in 3D) [69]. Many approximate cell decomposition methods have a recursive nature. A coarse approximation becomes finer at each level by subdividing cells that partically overlap both free and forbidden space. Examples of such recursive representations include the quadtree shown in Figure 2.2 and the octree shown in Figure 2.3.

2.2.2 Sampling-based path planning algorithms

Instead of computing an exact representation of the planning space as is done in combinatorial planners, sampling-based planners approximate the planning space

using samples and test motions between these configurations. Such planners usually represent motions as a graph as in the Probabilistic Roadmap Method (PRM) [59, 60], or as a tree as in the Rapidly-exploring Random Tree (RRT) [72]. These methods are probabilistically complete and it is not guaranteed that these planners will find a path even though one exists, but if they do find a path it will take the device from the initial configuration to the goal. We can classify sampling-based planners based on their strategy for exploring the configuration space. Roadmap-based planners construct a global exploration of the space. Tree-based planners start from one or two configurations and explore the space incrementally from these initial configurations.

Roadmap-based planners The basic roadmap-based planner PRM [60, 59] is one of the most successful methods for solving complex path planning problems. The PRM proceeds in two main phases: the preprocessing phase and the query phase. In the preprocessing phase (also referred as learning phase or roadmap construction phase in the literature) the roadmap $\mathcal{R}$ is constructed by connecting randomly sampled collision-free configurations into the roadmap using a simple local planner. This phase is outlined below:

1. *Generation of random nodes* (Figure 2.4(b)-(c)). Configurations are sampled by picking random configurations of $\mathcal{A}$. The basic PRM uses a uniform sam-

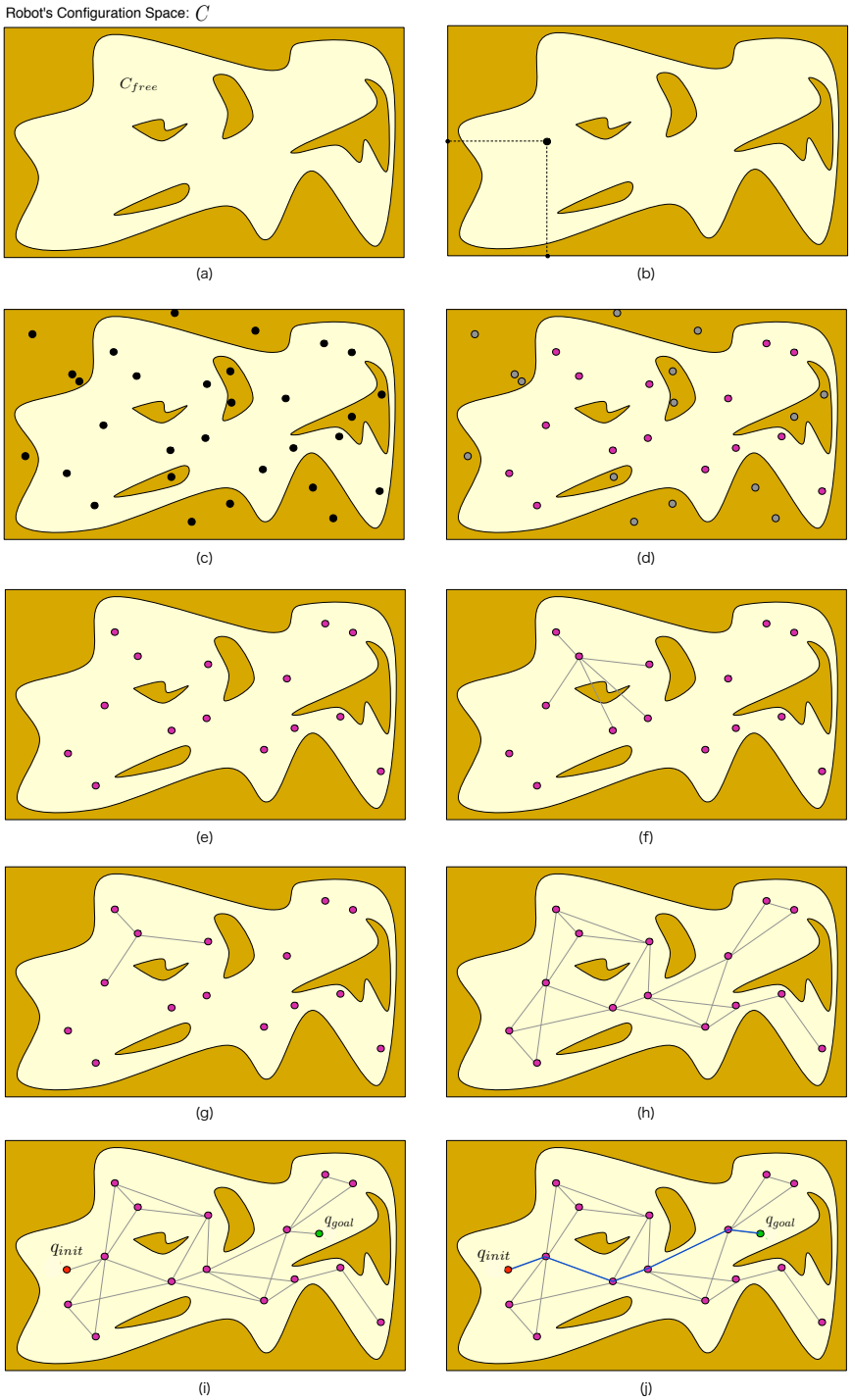


Figure 2.4: Basic PRM steps. See text for details. Figure derived from [25].

pling scheme which has difficulty in finding paths through narrow passages in the scene, because it places many samples in "open" regions but not enough samples in "tight" regions. A thorough analysis on this issue is given in [49]. The sampling methodology is crucial to the algorithm's performance, and many different sampling strategies have been proposed (e.g. [121, 49, 67]).

2. *Collision detection* (Figure 2.4(d)-(e)). Sampled configurations from the above step are tested for collision with obstacles and self-collision in workspace. Collision-free configurations (configurations in $\mathcal{C}_{free}$) are retained in $\mathcal{R}$. There exist a variety of techniques for efficient collision detection such as the grid method and the Bounding Volume Hierarchy (BVH) method [114, 64]. Usually the first two steps are interleaved until a pre-specified number N of nodes has been computed.

3. *Interconnection of the nodes with a local planner* (Figure 2.4(f)-(h)). Given some metric defined on $\mathcal{C}$, for each node x, all other nodes are ordered according to increasing distance from x and the local planner tries to connect x to each of the K (K is a predefined parameter) closest nodes. Choosing the proper distance function, local planner, and K is important to the performance of the PRM planner. There are tradeoffs in the choice of the local planner. Powerful local planners often succeed in finding a local path when

one exists, but they take more time and require more space to store the local paths. Simple deterministic local planners are less successful in connecting two nodes and thus require more nodes to be generated in the roadmaps, but the local paths computed do not need to be recorded since they can easily be recomputed in the query phase. Good experimental results have been obtained using simple deterministic planners [110, 59]. As an example, a typical fast local planner that tries to connect two configurations with a straight line in $\mathcal{W}$ has shown success in solving planning problems for very high-dimensional holonomic robots [60, 59].

In the query phase, a query (c_{init}, c_{goal}) is processed by first connecting c_{init} and c_{goal} to $\mathcal{R}$ (Figure 2.4(i)). Assume that $\mathcal{R}$ is a single-component graph. If the attempt to connect c_{init} and c_{goal} to $\mathcal{R}$ fails, then report failure. Otherwise, perform a graph search on $\mathcal{R}$ for a global path that starts at c_{init} followed by a concatenation of local paths and ends at c_{goal} (Figure 2.4(j)). The local paths here are recomputed without collision checking and should be the same as the ones computed when the roadmap was constructed.

Much work has been done to improve the basic PRM algorithm. Perhaps the most researched aspect of roadmap-based planners is the sampling strategy used. The main focus here is to improve the chance of producing samples inside narrow passages in the environment where the local planner is likely to perform poorly.

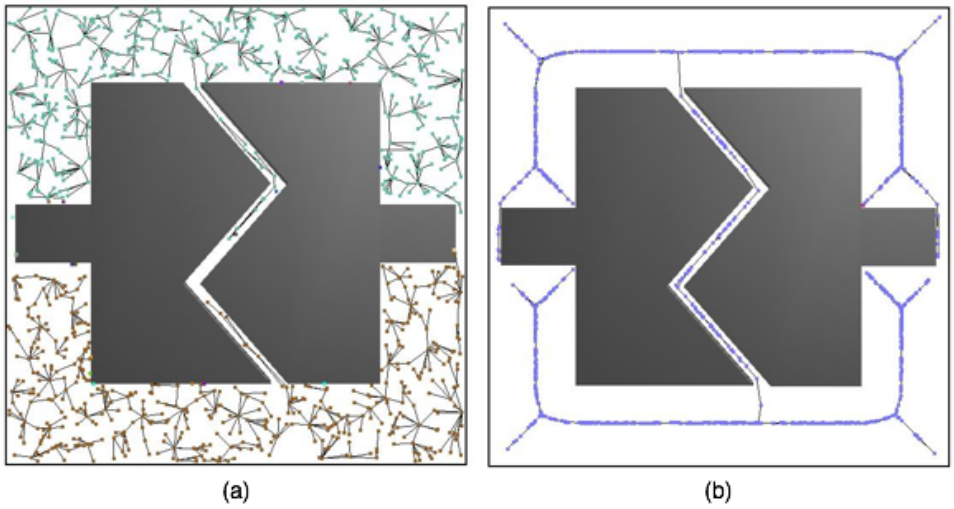


Figure 2.5: (a) Basic PRM with uniform sampling. (b) MAPRM retracts nodes towards the medial axis. Figure derived from [121].

For example, OBPRM [1], the obstacle-based PRM takes generated invalid configurations and pushes them in random directions to generate valid configurations around the boundaries of *C-obstacles*. MAPRM [121] performs uniform sampling and retracts every valid and invalid configuration towards the medial axis of the $\mathcal{C}_{free}$. Figure 2.5 shows an example of MAPRM. Although MAPRM can be implemented practically only for rigid bodies in three-dimensinoal space, an approximate version has been shown to perform well for high-DOF problems [76].

PRMs may pre-compute a roadmap and use it to process many queries. This requires the constructed roadmap to "cover" the entire $\mathcal{C}_{free}$ well. In contrast, single-query planners build a new roadmap for each new query and do not have to

achieve good coverage of $\mathcal{C}_{free}$. This is discussed in the next section.

Tree-based planners Tree-based planners root a tree at each of some set of valid configurations (typically the c_{init} and c_{goal}) and then they expand the trees in increments [6, 9, 48, 72]. The incremental exploration of the space makes these planners particularly well suited for single-query problems and problems with non-holonomic constraints (mobile robots with car-like motion constraints).

The Randomized Path Planner (RPP) [6] is perhaps the earliest sampling-based planner. Starting from c_{init}, RPP selects the new sample by alternating "down motions" to track the negated gradient of a potential field and "random motions" to escape local minima. The potential function is based on an estimation of the distance to c_{goal}. In this way, RPP incrementally builds a tree towards c_{goal}.

The Ariadne's Clew algorithm [9] builds a tree from c_{init} (shown in Figure 2.6). The algorithm operates by interleaving the exploration of $\mathcal{C}$ with searches for paths to c_{goal}. During exploration, new configurations are placed in $\mathcal{C}_{free}$ as far as possible from one another. The selection of good configurations can be difficult and is done through genetic optimization. For each new configuration, a local search is performed to determine if c_{goal} is reachable from it.

The Expansive-Space Tree (EST) [48] and Rapidly-exploring Random Tree (RRT) [72] algorithms are two widely used tree-based planners. EST and RRT

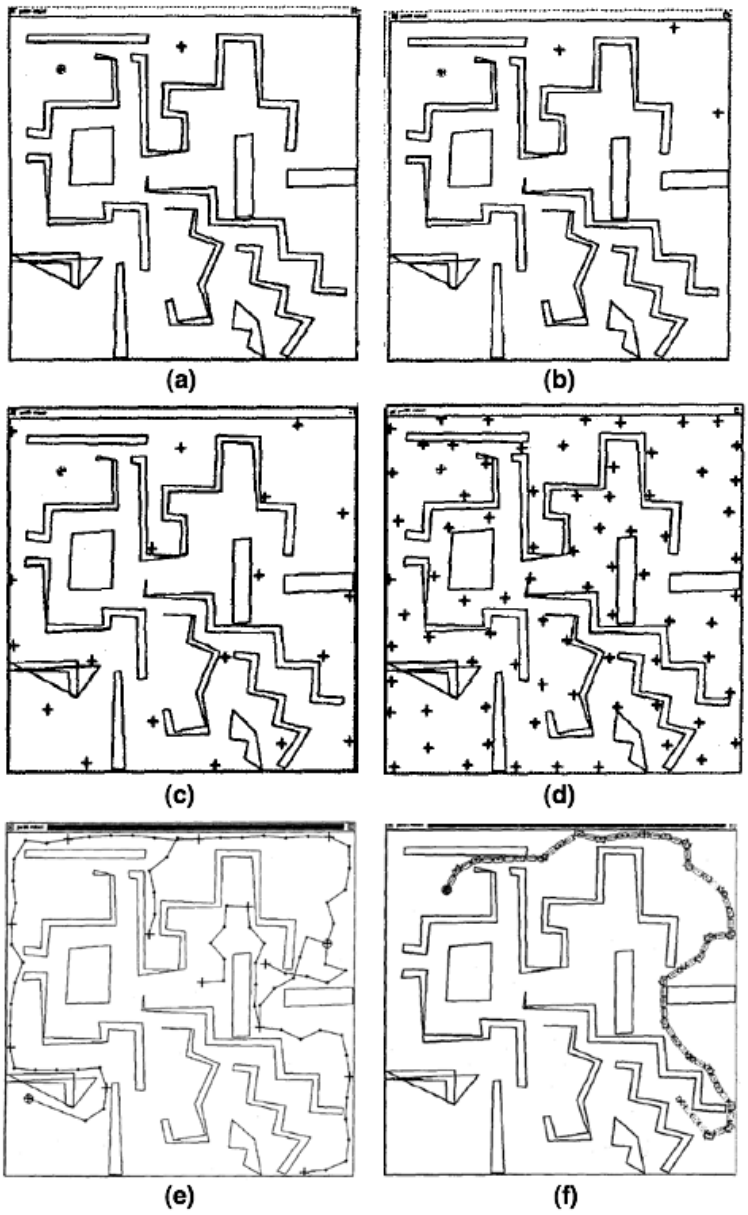


Figure 2.6: The Ariadne's Clew algorithm. (a) The initial configuration and the first generated configuration. (b-d) New configurations are generated to spread over the search space. (e) A tree of configurations allow to go about the free space. (f) A path found by the algorithm. Figure derived from [9].

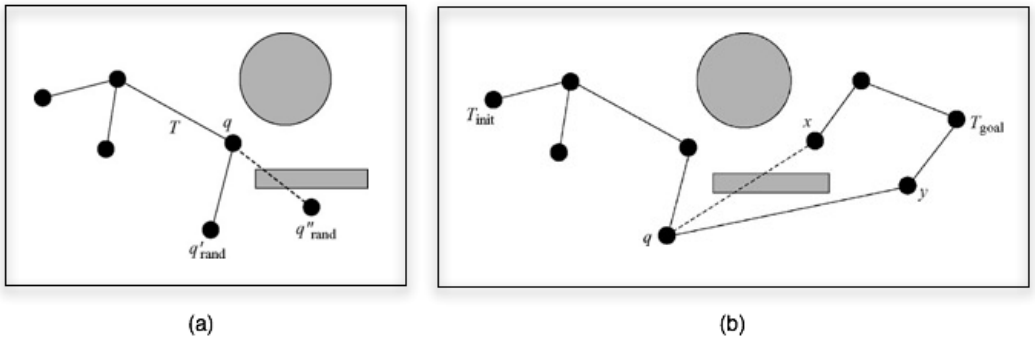


Figure 2.7: The EST algorithm. (a) Expansion. A configuration q is randomly selected from T upon the density of configurations in the configuration space. Then q'_{rand} and q''_{rand} are created in q's neighborhood. The local planner succeeds in connecting q to q'_{rand}, so q'_{rand} and the edge (q, q'_{rand}) are added to T. (b) Merging two EST trees. Suppose q is just added to the T_{init}. The local planner attempts to connect q to its closest configurations x and y in T_{goal}. The local planner fails to connect q to x but succeeds in the case of y. Therefore the two trees are merged. Figure derived from [25].

are initialized with trees rooted at c_{init} and c_{goal}. These algorithms alternate between two basic operations: expansion and merging. In the expansion step, new nodes are randomly sampled from $\mathcal{C}_{free}$ near the boundaries of the two trees. A node is added to a tree only if it can be connected by the local planner to some existing node. In the merging step, the local planner attempts to find a linking sequence between the two trees. These two operations are repeated until a path is found between c_{init} and c_{goal} or a maximum number of expansion and connec-

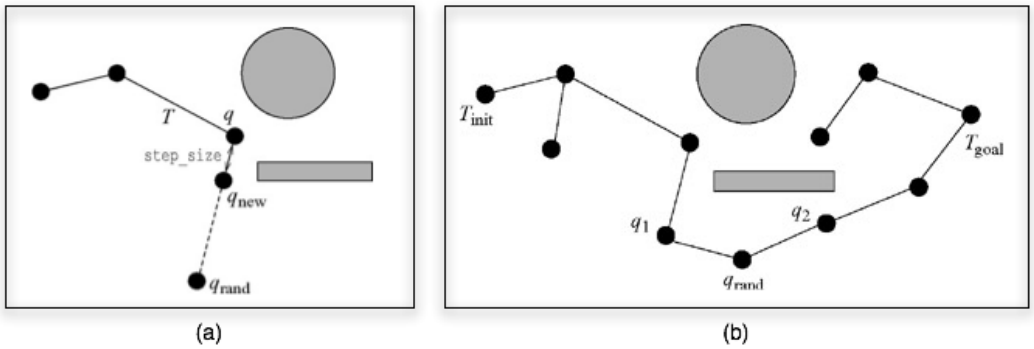


Figure 2.8: The RRT algorithm. (a) Expansion. It first selects a random configuration q_{rand} from a uniform distribution in $\mathcal{C}_{free}$. Configuration q is the closest configuration in T to q_{rand}. The new configuration q_{new} is obtained by walking q by *step_size* toward q_{rand}. Only q_{new} and the edge (q, q_{new}) are added to the RRT. (b) Merging two RRT trees. Configuration q_{rand} is generated randomly from a uniform distribution in $\mathcal{C}_{free}$. Configuration q_1 was extended to q_{rand}. q_2 is the closest configuration to q_{rand} in T_{goal}. It was possible to extend q_2 to q_{rand}. Therefore, T_{init} and T_{goal} are merged. Figure derived from [25].

tion steps is reached. The differences between the two algorithms are illustrated in Figures 2.7 and 2.8.

One of the bottlenecks of RRT is that in some environments most of the randomly selected samples will cause the expansion from the closest node in the RRT tree to fail. This produces a significant increase in the runtime of the algorithm. One of the newer RRT-like algorithms is based on utility trees [18]. The main improvement for this type of tree is that more aspects of the tree growth are eval-

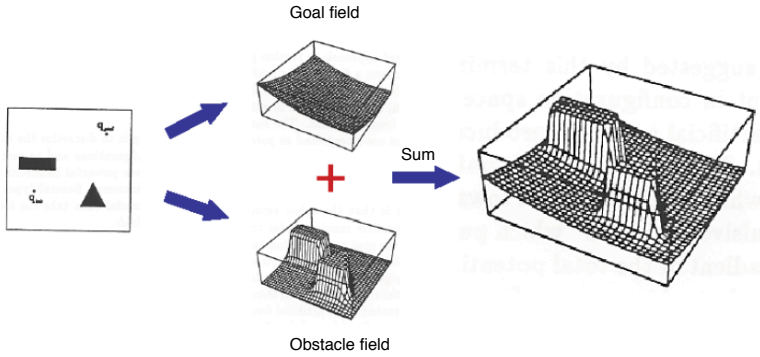


Figure 2.9: Potential field function can be the sum of the goal field function and the obstacle field function. Figure reprinted from [69].

uated when expanding a node, including: the utility of the node to be expanded, the expansion direction, the expansion distance and connection attempts.

2.2.3 Other planning methods

Besides traditional combinatorial planners and sampling-based planners, there exist a large number of other classes of planning algorithms and it is impossible to review all of them here. (A thorough description of many of these alternatives can be found in [69] and [25].) In this section, we review a few of the more important planning methods that have found application in real world tasks.

2.2.3.1 Search-based path planning algorithms

Search-based algorithms currently dominate path planning in field robotics because of the relative ease of their implementation [123]. More importantly, field robots usually require the path quality to maintain close to optimal and search-based algorithms are able to find solutions for low-DOF systems with known bounds on sub-optimality. The basic idea behind search-based planning is this: a grid of regularly sized cells is used to represent the $\mathcal{C}$. The start and goal configurations of the robot are known within the grid and a search is run on the grid to solve the path planning problem. Simple (but expensive) algorithms such as Dijkstra's or A^* [43] are used to find paths. As further information is encountered the path must be replanned. Algorithms such as D^* [108] can be used to reduce the effort involved in replanning.

There exist a number of popular algorithms for robot navigation that abandon the idea of planning altogether. These algorithms try to move the robot towards the goal greedily and apply different methods to deal with obstacles encountered on the way. The Bug algorithm [79] and its variants, for example, will make the robot follow the boundary of an obstacle when encountered. Another approach, the potential field method, was initially proposed for online collision avoidance, but can also be used to solve general planning problems [70]. It uses an artificial potential

function as a heuristic to guide the search for a path. The function is produced by c_{goal} as an "attractive potential" which pulls $\mathcal{A}$ toward c_{goal} and the *C-obstacles* as a "repulsive potential" which pushes $\mathcal{A}$ away from them (see Figure 2.9). The negated gradient at a given configuration c suggests the most promising direction of motion at c. These greedy algorithms are simple to implement and work well in environments with sparse obstacles. Perhaps even more importantly they are well-suited for very large environments since the size of an environment does not have an effect on their complexity. However, these approaches have a hard time dealing with local minima created by obstacles in the environment [70, 102].

2.2.3.2 Planning with uncertainty

Path planning in realistic scenarios may have to consider dynamically changing environments, partially known environments and inaccuracies in sensing and the robot's movement. This problem is known as planning with uncertainty, which many researchers regard as one of the most interesting problems in path planning [70]. Nearly all the path planners discussed above make the assumption of no uncertainty, but some have been adapted to deal with uncertainty. For example, based on A^*, the dynamic search-based algorithm, D^* [108], was introduced in mid-1990s. It has since become the basis of planning for the vast majority of field robots. Focused D^* [109] and quad-tree D^* [124] have been developed to address

the time-complexity and space-complexity limitations of the D^*, respectively.

Tree-based planners can provide an effective framework to deal with uncertainty when planning. [34] presents an RRT planner that is the probabilistic analogue to the family of D^* algorithms [108]. Specifically, an RRT tree is grown to cover the space until an obstacle is sensed in the way. The problem of PRM planning in dynamic environments was first considered by Leven and Hutchison [74] who propose a roadmap representation for the dynamic environment that can be efficiently modified as obstacles move. Another approach [55] assumes a finite set of dynamic obstacles, such as doors, whose motion is known. In this work the roadmap is augmented to label edges as possibly obstructed by a dynamic obstacle. When a path is computed in the roadmap, these possibly obstructed edges are rechecked to ensure they are free. If a path cannot be found, a local planner is used to reconnect and replace the obstructed edge in the roadmap.

Machine learning techniques have also be employed in the context of robot path planning to deal with the complexity and the uncertainty of the problem. Through the use of learning, the robot can improve its performance and adapt to changes in the environment. For example, machine learning techniques have been used to select which path planner is applicable to a particular region of the space [86] and to adapt the mixture of several sampling strategies based on the past success of each strategy [50, 67]. Burns et al. [19] propose a model-based path planning method,

which views the task of building an approximate model of the configuration space as a classification task. Let the configuration space $\mathcal{C}$ be represented by a binary classification function $V(c)$ which returns 1 if the configuration $c \in \mathcal{C}$ is free and 0 otherwise. Given a collection of sampled configurations that have been labeled with their state, machine learning techniques can be used to construct an approximation of the function V. The machine learning literature provides numerous algorithms for classification (e.g. [85]). In Burns' work, they use locally weighted regression [28] to incrementally construct and refine an approximate statistical model of the configuration space. The model indicates the areas which are simple and the areas which are complex and also makes predictions about unexplored parts of the configuration space. Cassandra et al. [22] model the world as a partially observable Markov decision process (POMDP). The state space for the POMDP is a set of abstract observations of the environment (door, hallway, etc.) and abstract actions (move-forward, turn-left, etc). This abstract state space is constructed on top of a lower level system which is error prone. The POMDP formulation explicitly represents and reasons about the possibility of sensor error. Several heuristic strategies for approximating the computationally intractable optimal policy are suggested. These approaches have been successfully used for mobile robot navigation in office environments. The POMDP formulation requires that the state space of the robot be discrete or that it be well represented by a discrete approximation.

2.3 Planning practical paths

With the development of sampling-based algorithms and their application in practice, the focus in the research literature shifted to considering the quality or practicality of the computed path. Intuitively the quality of the computed path of sampling-based planners relies on the sampling scheme, the number of generated samples, properties of the local path planner, and the global search method. Existing randomized path planning algorithms addressing the path quality problem can be divided into three broad categories based on where practical issues are integrated within the algorithm: pre-processing, post-processing, and customized learning. The first category performs operations before the query phase. The second category optimizes the computed path after the query phase. The third category performs enhancement or customization during the query phase, and these strategies are usually applied to navigation planning when the path is planned online. Each of these approaches are considered in detail below. Note that in this section we focus on static environments, i.e. there are no moving obstacles in the environment. We also assume that the robot has prior knowledge of the environment. These are often realistic constraints to the problem, especially for environments like the nuclear inspection/repair tasks where time can be taken to build sophisticated computer models and access to the environment is restricted. Figures 1.2(c-d)

and 1.3 provide examples of such environments.

2.3.1 Pre-processing

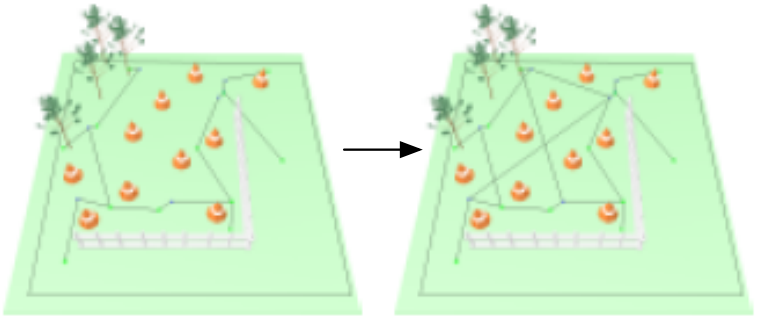
Pre-processing approaches consider the specific preferences (soft constraints) of desired paths in the pre-processing phase, i.e. during the roadmap construction before a query is made. When constructing the roadmap, we may want to base it on some structured graph representation, such as the Visibility Graph (VG), the Generalized Voronoi Diagram (GVD) presented in Section 2.2.1, or some similar structure that is informed by the environment. Such geometric models have special properties and are often used to represent a complete environment. Although the exact computation of these graphs is not practical for high-DOF problems, approximation of these representations can provide insights into the planning problem and help in finding "better" paths.

The visibility-based PRM (VIS-PRM) [90] is a variant of the PRM that takes advantage of the visibility notion. While normally each collision free configuration generated by the basic PRM is added to the roadmap, VIS-PRM only keeps configurations which either connect two connected components of the roadmap (called connectors), or are not visible by so-called "guard configurations". Otherwise, the generated configuration is rejected. This method tends to create smaller number of configurations when compared to the basic roadmap approach.

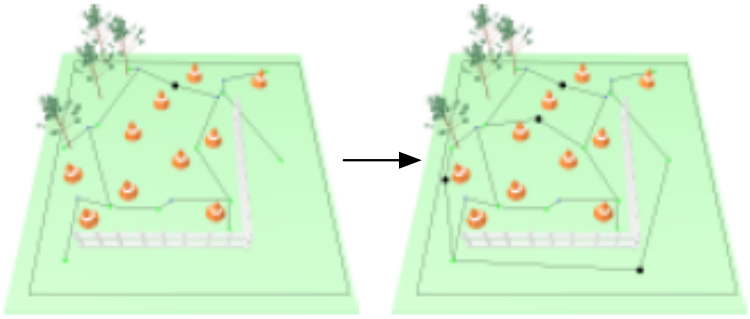
Based on the close relationship between the workspace and the robot's configuration space, an approximate GVD of the workspace can provide useful information for probabilistic path planning. In [36], the constraint-based motion planner first uses a method proposed in [45] to quickly compute a discretized, error-bounded approximation of the GVD of the workspace. It then uses constraint forces to move the robot along an estimated path computed using the approximate GVD through the environment, while avoiding collisions with obstacles and enforcing joint and positional constraints. This enables the moving robot to satisfy all constraints while remaining "maximally" clear of nearby obstacles.

Since the VG tends to yield paths with semi-free configurations (i.e. zero clearance) and the GVD tends to yield paths that may be much longer than the shortest one, Wein et al. [119] introduced the $VV^{(c)}$ diagram, a hybrid between the VG and the GVD in the plane. It evolves from the VG to the GVD as the clearance grows from 0 to ∞. The processing time of constructing the $VV^{(c)}$ diagram is $O(n^2 \log n)$ and this is improved to $O(n \log n)$ where n is a multiple of the number of obstacle vertices in [11].

After the roadmap construction phase most PRMs utilize an enhancement phase during which additional nodes and edges are added to the roadmap in order to discover and deal with difficult regions (known as narrow passages in the literature). Similar strategies can be applied to improve the ability of the roadmap to extract



(a)



(b)

Figure 2.10: (a) "Useful" edges are added to the roadmap for extracting short paths and providing alternative routes. (b) "Useful" nodes that lie on the medial axis are added to the roadmap for extracting high clearance paths and providing alternative routes. Figure reprinted from [38].

practical paths at query time.

Aiming at finding shorter paths with little variation between the executions, Nieuwenhuisen and Overmars [89] proposed a technique that adds "useful" cycles to

the roadmap. The technique is based on adding edges that have a high probability of introducing paths that are in different homotopy classes than existing paths in the roadmap. Based on this work, in [38], both nodes and edges are added to the roadmap to create "useful" cycles, which provide short paths and alternative routes which allow for variation in the routes the robots can take. On one hand the algorithm attempts to add edges connecting existing nodes that have a high probability of introducing a new homotopy class to the existing homotopy classes of paths (see Figure 2.10(a)). On the other hand the algorithm attempts to add new nodes that lie on the medial axis which can also introduce new homotopy classes of paths (see Figure 2.10(b)). Finally assuming that each node of the roadmap lies on the medial axis, the edges are retracted to the medial axis for high clearance paths. The retraction technique can also be used in the post-processing phase, see below.

2.3.2 Post-processing

Given a path found by the sampling-based path planner, post-processing approaches modify the path in accordance with the required practicality preference by adding new nodes, smoothing the path, eliminating unnecessary loops or detours, etc. Post-processing approaches are used widely to address the path quality problems in practice. Two retraction algorithms are presented in [37] to add clearance to a given path. The first one is called *W-RETRACTION* and is performed in the workspace

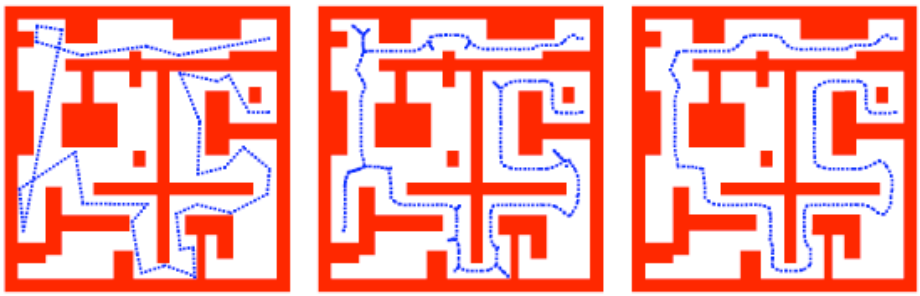


Figure 2.11: Retraction of a path traversed by a square robot in a 2D workspace. (a) The given path given by the sampling-based planner. (b) The path has been retracted to the medial axis of the workspace. (c) Branches of the path have been moved. Figure reprinted from [38].

(see Figure 2.11). This approach retracts each configuration along the path to the medial axis. Such a retracted configuration will have (at least) two-equidistant nearest points to the obstacles in the workspace, resulting in a large clearance. This approach can be accurate and fast but is only suitable for translating rigid robots with two to three DOFs. For articulated robots or free-flying robots, [37] introduces the *C-RETRACTION* algorithm which increases the clearance of the configurations on the path by moving them in a random direction *dir* in configuration space iteratively. The random direction *dir* incorporates all of the degrees of freedom of the robot. The process is terminated when the average clearance of the paths does not continue to improve. This method can handle a larger range of robots which

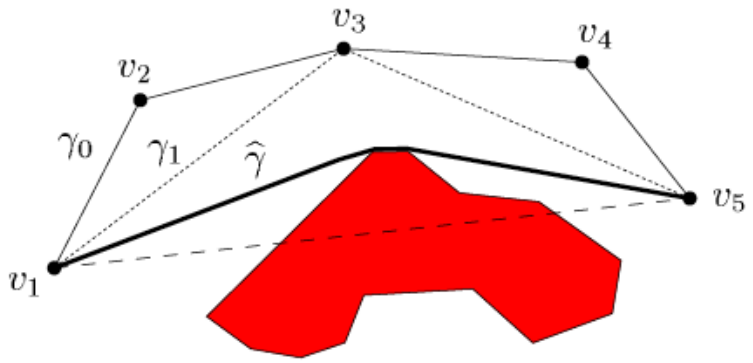


Figure 2.12: Shortcut method. The path γ_0 becomes γ_1 after one iteration of the shortcut. However, γ_1 is far from the minimum $\hat{\gamma}$. Figure reprinted from [48].

may reside in arbitrary high-dimensional configuration spaces, but is slower than *W-RETRACTION*. Experiments indicate that this path enhancement step may be too slow to be applied online.

Due to their probabilistic nature, sampling-based planners create paths that may contain unnecessary and jerky motions. Applications that require high-quality paths can employ a post-processing step to enhance the quality of the path by smoothing it, eliminating unnecessary loops or detours, etc. For a discrete path $\mathcal{P}$, path pruning is a simple eliminating technique that removes a node c_{i+1} from the path if the local path between node c_i and c_{i+2} is collision-free. This technique is simple, efficient and deterministic.

The shortcut method is expected to create shorter paths than the pruning

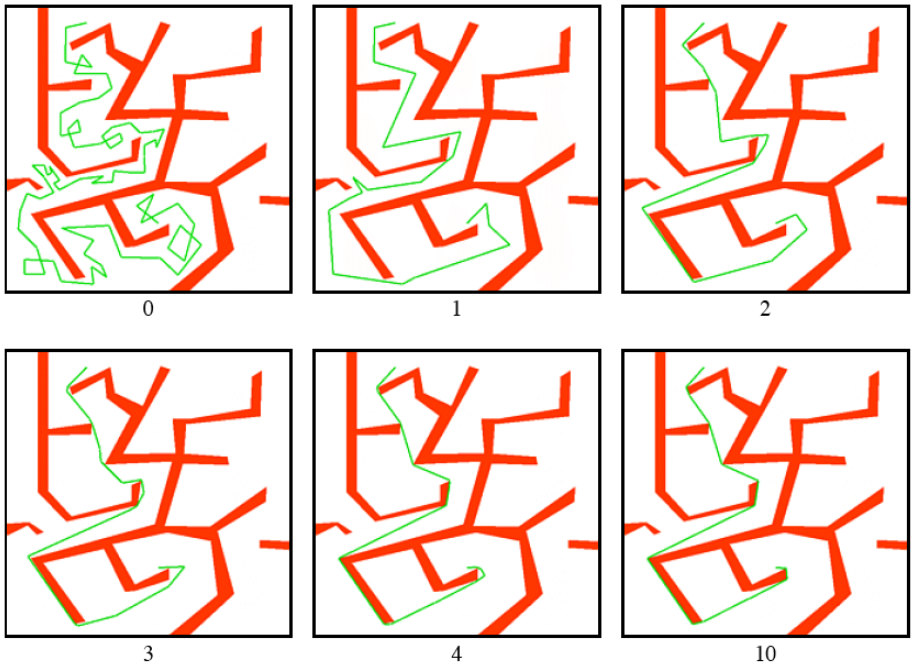


Figure 2.13: Shortest paths computed by the Adaptive-Shortcut after 0, 1, 2, 3, 4, and 10 iterations. Figure reprinted from [48].

method at the cost of increased computation time. As shown in Figure 2.12, the shortcut heuristic discussed in [48] recursively breaks a path $\mathcal{P}$ into two sub-paths and then checks whether they can be replaced by a new local path. It takes linear time to execute but may stop far short of reaching the minimum-length path. In order to address this problem, the Adaptive-Shortcut algorithm can be used. This algorithm scans through each configuration in the path and adds additional configurations as necessary. Then the shortcut algorithm is invoked again. The process terminates when no further improvement is possible. A computed example

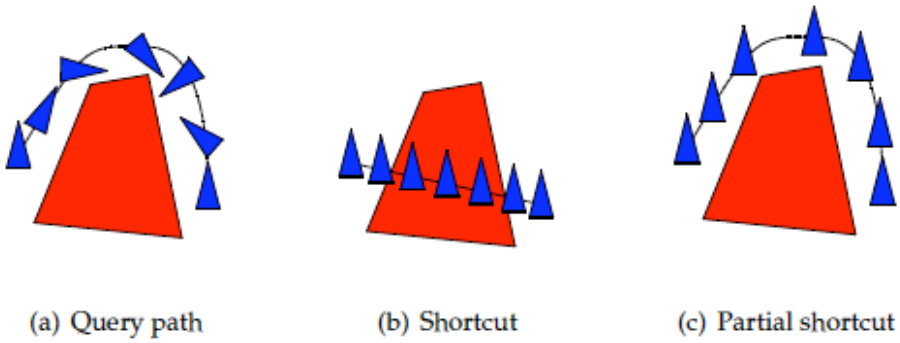


Figure 2.14: Shortcut and partial shortcut of the query path, i.e. the part between two configurations (chosen deterministically or randomly) on the given $\mathcal{P}$. (a) The query path traversed by a mobile robot in a 2D workspace. (b) The shortcut of the query path. (c) The partial shortcut of the query path for which only the shortcut of the rotation is considered. Figure reprinted from [38].

is shown in Figure 2.13.

Another version of the shortcut algorithm [38] takes two random configurations on the path. If the path between these two configurations can be replaced by a better path produced by a local planner, then the original part is replaced by the new path. The configurations can be chosen randomly [59] or deterministically [53]. The partial shortcut [39], a variant of the shortcut method, takes only one DOF into account in each optimization step (see Figure 2.14). A particular DOF f is chosen depending on its predefined weight. For the query path between two randomly

chosen configurations on $\mathcal{P}$, the partial shortcut is generated by replacing in each configuration the value of a DOF by the new value.

Post-processing algorithms may take multiple paths as inputs rather than just a single one. For example, the path merging algorithm described in [98] computes a path with improved quality by hybridizing high-quality sub-paths from two or more initial input paths. This method creates a hybridization graph of the input paths, where the vertices are the states of the input paths, and edges indicate valid paths between the states. Once a set of valid bridges between the input paths are discovered and inserted as additional edges into the graph, a classical graph search can be used to find a shorter, hybrid path. The algorithm considers the generalized formulation of path quality measures rather than specific requirements.

2.3.3 Customized learning

Although post-processing algorithms have shown some success in improving the path quality and can generally be integrated into any path planner, the final path found depends on the path found originally, i.e. they cannot find alternative routes that deviate considerably from the original one. In order to address this problem customized learning algorithms integrate the requirement for path quality in the learning phase, i.e. after a query is made but before the path is found. Iterative approaches are relatively straightforward and can be applied to many problems.

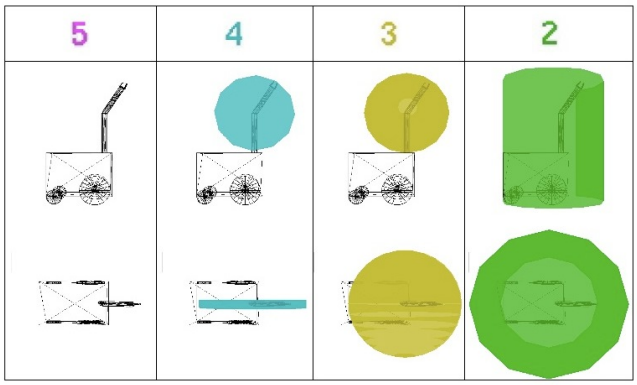
During the learning phase they enhance the path until it meets the specified requirements using an iterative process.

The common philosophy of initially finding an approximate solution is utilized by the Fuzzy PRM [88], Lazy PRM [14], IRC (Iterative Relaxation of Constraints) [7] and Customizable PRM (C-PRM) [107] algorithms where the roadmap nodes and edges are not validated or are only partially validated, during roadmap construction. During the learning phase, the path is refined iteratively by strengthening the constraints. These methods are designed to decrease the roadmap construction costs, while only increasing the query costs slightly. To take one example, the C-PRM is specifically designed for answering variable and adaptive query requirements [107]. C-PRM follows the traditional PRM paradigm. The main difference is that the C-PRM builds a coarse roadmap and postpones the detailed validation of the path including customization for any particular query preferences to the query phase. In the query phase, the shortest path (or the path with some other specified preference) between the start configuration and goal configuration is first searched as in traditional PRM. Then collision checking and any desired query specifications are enforced on the path. If any node or edge on the path fails to meet these requirements it is removed from the roadmap. A new shortest path is then searched for in the refined roadmap, and the process iterates until a desired path is found or failure is reported. This approach has been shown to be able to deal with

variable query preferences such as large clearance, small rotation, smoothness, few sharp corners, avoiding singularities for manipulators, and low potential energies for ligand binding and protein folding [107].

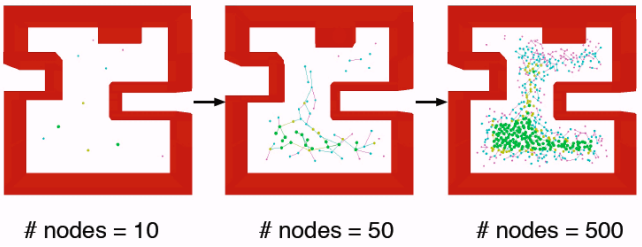
Aiming at establishing the reachable workspace of the kinematic structure moving in a cluttered environment, the Hierarchical PRM (HPRM) [126] also uses an iterative approach in the planning process. As shown in Figure 2.15 the algorithm builds a hierarchy of the DOFs in terms of their predefined "importance." In [126] the importance of the DOF is determined by its effect on the volume of the kinematic structure's occupancy in the workspace. Validation of configurations begins by doing fast tests on simple occupational representations of the kinematic structure and only progresses to more accurate (and more expensive) evaluations if necessary. Therefore, when the kinematic structure executes a path from the start to the goal, it only moves its 'less important' DOF (in this case the arm is less important than the mobile base) when necessary. This reduces unnecessary flailing of less important DOFs (here the arm) while the mobile base is moving.

Kim et al. [63] use an augmented version of Dijkstra's algorithm to extract a path from a roadmap on criteria other than length. The system enables any combination of optimization criteria, such as collision detection, minimum clearance and kinematic/dynamic constraints, to be used when selecting paths from roadmaps. For example, the minimum clearance along the path is optimized by incorporating



(a)

(b)



(c)

Figure 2.15: Hierarchical PRM on a mobile robot with a two-link manipulator. (a) Representations of the hierarchical occupancy of the robot are shown in different colors. (b) The generation and classification of one node. (c) Hierarchical PRM in action. The coverage and connectivity of the roadmap increases as more nodes are added. Figure reprinted from [126].

a higher cost for edges that represent a small amount of clearance. The algorithm depends on a well connected roadmap, which can be expensive to generate because of the large number of collision checks necessity. In addition, the resultant path rarely provides an optimal solution because it is restricted to the randomly generated nodes and edges in the roadmap.

Greedy and incremental approaches are generally applicable to robot navigation (navigation basically means the problem of determining the elementary motion that the robot should perform during the next time-step). In [8], a greedy exploration strategy is used to bias the search towards a better path when the planner incrementally updates its tree data structure. This approach is designed for robot exploration when the robot only has partial knowledge about the environment. As its configuration space is evolving in the process, re-planning is required.

Classic grid-based methods such as A* and D* can be used to compute resolution-optimal paths over a costmap, but these methods are limited to low-dimensional spaces. Two important RRT variants that address this issue are the heuristically guided RRT (hRRT) [112] and Transition-based RRT (T-RRT) [54]. The hRRT biases the search by using a quality measure based on the integral of the cost along the path from the root node and an estimation of the optimal cost to the goal. However, the estimated cost to goal in hRRT is heuristic, and tends to bias the search straight toward the goal at the expense of lower quality solution paths. Moreover,

it has only been demonstrated on simple low-dimensional examples with discrete cost states (invalid, low cost, and high cost, respectively). Its scalability and performance for problems which involve complex cost spaces in higher dimensions have yet to be established [112]. Aiming at addressing cost functions in high-dimensional configuration space, the T-RRT combines the exploratory strength of RRTs with stochastic-optimization methods that use transition tests to accept or reject new potential states. The filtering of the transition test relies on the gradient of cost function along the local motion to connect a given state to the RRT tree that results in an expansion biased to follow the valleys and the saddle points of the configuration-space costmap.

Karaman and Frazzoli [58] proposed the sampling-based algorithms PRM*, RRG and RRT*, which always converge to an optimal solution that minimizes the length of the path, or the time required to execute it. PRM* is a variant of PRM with a variable connection radius that scales in the number of samples, while standard PRM uses a constant radius value to select neighbors to connect. RRG and RRT* are variants of RRT, which incrementally build a connected roadmap, augmenting the RRT algorithm with connections within a sphere scaling the radius with the number of samples. RRT* has been shown to return significantly shorter paths than RRT given a specified number of samples when planning.

The EST approach has also been extended [94] to search for a low cost and

relatively straight paths for systems with constraints on controls. It incorporates a heuristic cost function and the degree of the node in addition to tree density to bias node selection, such that the search is guided towards the low cost region. Once a path is found, a path gradient descent approach refines the existing path according to a cost function by following the gradient of the path. The cost function is defined as the sum of a term to penalize being near obstacles and a term to penalize for high control cost.

2.4 Soft constraints

Many of the problems with planning practical paths are related to having to deal with both hard and soft constraints within a common representation or framework. The hard constraint is the need to obtain a collision free path. The soft constraints are the desire to obtain paths with good clearance, short length and the like. Hard constraints are constraints that must be satisfied by a solution to the problem, however, soft constraints may be violated and serve as guides to *encourage* or *influence* the planner to find the best solutions.

Soft constraints have been studied more generally outside of the problem of robot path planning and can be found in the Constraint Satisfaction Optimization Problem (CSOP) (e.g., [12, 30, 35]), control theory (e.g., [61, 66, 130]), and Artificial Intelligence (AI) planning (e.g., [40, 115, 5]). Here we review soft constraints

in robot planning and control and AI planning as they are closely related to this work.

2.4.1 Soft constraints in robot planning and control

The integration of soft constraints in robot path planning is still limited. One common way of taking soft constraints into account is to use an appropriate controller that takes the path identified by the path planner as input and while implementing the desired path optimizes these other soft constraints [16, 65]. Optimal control deals with the problem of producing the appropriate input forces or torques for the robot to follow the desired motion while achieving a certain optimality criterion. A typical objective is to minimize the duration of execution of the trajectory. Meanwhile, user-specified soft constraints can be added to the objective cost function in the optimization. One approach to computing time-optimal trajectories is to approximate the trajectories with B-splines [104]. Alternatively, it is possible to discretize the trajectory and use finite-difference methods to solve the optimization problem [129].

Control methods are developed to simultaneously optimize multiple performance criteria for a redundant manipulator [27, 32]. In [32] cost functions used by the controller are formulated by summing together the weighted soft constraints, which address problems associated with singularities, joint velocity demands, joint limits

and collision with workspace obstacles. To optimize the given soft constraints, the Gradient Projection Method (GPM) modifies the manipulator configuration by controlling the amount of self-motion added to the joint velocity vector.

Nearly all of the path planners discussed above make assumptions about the uncertainty that may arise from dynamic environments, partially known environments, inaccuracy in sensing and the robot's movement etc, but leave these issues to the control phase when the path is executed. Work is being done to bring uncertainty issues into the earlier path planning phase [8, 51, 113]. For example, the LQG-MP introduced in [113] calculates the *a priori* probability distributions of the state and controls along a given path before it is executed, based on a linear-quadratic controller with Gaussian models of uncertainty. These distributions are then used to assess the quality of the path, for example, the probability of collisions and the likelihood that the robot will arrive at the goal. Then the best path can be chosen from the large set of paths generated by any path planner.

There are many issues involved in separating path planners and controllers. Typically, controllers alone cannot avoid obstacles in the environment, and that is why an obstacle free path must be found first. The paths produced may be infeasible for a real robot. Path planners typically create paths without considering device dynamics. Even when the controller manages to follow a desired path this may require that the robot moves extremely slowly to minimize the influence of

dynamic and physical constraints. Finally, these controllers are system specific, and as today's robots become increasingly complex it becomes very hard to develop good general controllers or to know which controller to use for which task. It is desirable to develop a path planning framework to take such soft constraints into account to reduce the burden of controllers.

Several approaches blend planning and control by defining a global control policy over the entire environment. A partially observable Markov decision process (POMDP), for instance, can be used with motion uncertainty and sensing uncertainty to optimize the probability of success [22, 68, 96]. However, the POMDP formulation requires discretization of the state and control input spaces, so it suffers from issues of scalability. Several heuristic strategies for approximating the computationally intractable optimal policy have been suggested. These approaches have been used successfully for mobile robot navigation in office environments.

Designed for dynamic environments with moving obstacles, [36] proposed the constraint-based path planning algorithm which reformulates the problem as simulating a constrained dynamic system and guides this system using generalized Voronoi Diagrams. This approach uses penalty forces to represent soft constraints including surface repulsion, goal attraction and high-level path following. By combining all the virtual forces the constrained dynamic system can guide the robot to follow these constraints, leading the path toward the goal.

Path planning with constraints is common in the field of computer graphics, where the goal is to create an animation involving one ore more simulated bodies, that will not only result in a correct motion, but also achieve a particular desired visual behavior during its execution of the trajectory [87, 131]. For example, in [87] sampling-based path planning is integrated with constraint-based dynamics simulation to compute collision-free and realistic motion for deformable models. First a sampling-based path planner generates a desired path for the deformable model. The solution is then refined using the constraint-based dynamics simulation to generate physically plausible motion. The method allows user-provided soft constraints including smoothness of the path (no two segments of the path form an angle larger than a specified bound) and goal attraction, which are converted into forces imposed on the dynamical system of the deformable robot model.

In [97], a covariant gradient decent technique was used for motion planning for a 7-DOF manipulator. This algorithm called Covariant Hamiltonian Optimization and Motion Planning (CHOMP) directly encodes the collision-free constraints using a global potential field and continuously refine a potentially invalid input trajectory. It also represents various soft constraints (smoothness, torque, etc.) in terms of additional penalty terms to the objective function. This technique often converges to a valid trajectory in a local minimum of the optimization space. A similar approach STOMP [56] was later developed to handle general cost functions for which

gradients are not available. It also develops a stochastic trajectory optimization to overcome local minima that CHOMP can get stuck in.

2.4.2 Soft constraints in AI planning

Representing and reasoning with soft constraints has been extensively studied in AI planning. In the AI context planning typically takes on a discrete flavor. The task might be to solve a puzzle, such as the Rubik's cube or a sliding-tile puzzle, or to achieve a task that is modeled discretely, such as building a stack of blocks. At each time step only a finite set of actions can be applied to a discrete set of states and a solution is constructed by giving the appropriate sequence of actions [71]. Many real-life optimization problems have both hard and soft constraints. For example, in a product configuration problem, the producer may pose some hard constraints on the component compatibility and soft constraints on supply time, while the user may provide the system with her subjective preferences over alternative products. The notion of plan quality is of great practical importance because the problems with a large set of solutions or with a set of goals that cannot all be achieved have to be addressed [40, 30].

Many formalisms for describing soft constraints have been proposed in the literature and there is a significant amount of work on planning algorithms with soft constraints that is related, in varying degrees (e.g., [13, 61, 36, 40]). It is impossi-

ble to review them all especially the related planning algorithms. Here we review a few important formalisms for describing soft constraints in the literature, which we need to formulate the problem of planning practical paths.

There has been work that casts the preference-based planning problem as an answer set programming problem (ASP) [106], as a constraint satisfaction problem (CSP) [13, 15], and as a satisfiability (SAT) instance [42]. In the CSP domain, each assignment to the variables of a constraint is annotated with the level of its desirability, and the desirability of a complete assignment is computed by a combination operator applied to the "local" preference values [13]. Also related is the work on partial satisfaction planning problems (PSPs) [116, 105]. PSPs can be understood as a planning problem with no hard goals but rather a collection of soft goals each with an associated utility; actions also have costs associated with them.

The Planning Domain Definition Language (PDDL) is the standard planning language specifically designed to provide a uniform syntax for describing planning problems in the context of the International Planning Competition (IPC) since 1998 [84]. It has gone through a number of revisions and is currently a *de facto* standard for describing planning problems [5]. Planning with soft constraints was a theme of the 5th IPC, and PDDL3 [40] was specifically designed for it. It extends PDDL2.2 [33] to include, among other things, facilities for expressing soft constraints and supports quantifying the value of achieving different preferences

through the specification of a metric function.

PDDL3 [40] introduced hard constraints in the form of modal-logic expressions, which should be true for the trajectory produced by the plan and soft constraints in the form of logical expressions, similar to hard constraints, but their satisfaction is not necessary. Traditionally plan quality is expressed using metrics such as time, energy used, number of plan steps, etc. PDDL3 incorporates soft constraints into the plan metric to minimize their violations or to just measure the quality of a plan. Different priorities of soft constraints are also taken into account in the plan quality evaluation. PDDL3 has chosen a simple quantitative approach: each soft constraint is associated with a numerical weight representing the cost of its violation in a plan and hence also its relative importance with respect to the other specified soft constraints. In the robot path planning literature, path quality is typically measured by *ad hoc* cost functions like path length, time for the robot to execute the path, etc. These ad hoc criteria make it difficult to express a wide range of applications and it is possible to leverage results from the AI planning community to define soft constraints within this or other standard formalisms.

2.5 Resource allocation

In sampling-based path planning, the problem of optimizing soft constraints involves optimizing over a collection of possible constraints at various stages in the

computation. Given a set of optimization methods associated with different optimization costs, it is necessary to determine how to distribute a constrained optimization budget among them. We first review existing path planning related strategies that address the issue of resource allocation among multiple algorithms.

2.5.1 Hybrid path planning methods

Different sampling strategies have different strengths. Building on this observation one fruitful idea is to combine the usually complimentary strengths of different sampling strategies. In [50] an adaptive strategy for selecting the most cost effective sampler out of a set of already existing ones using a simple reinforcement learning method is presented. It assigns a weight to each strategy and dynamically updates the weights according to the performance of each strategy. A reward is given to a component sampler whenever it samples a node that improves the coverage and/or connectivity of the current roadmap, while the cost is the time used for sampling the node. Clearly a component sampler with larger total reward per unit cost is considered as a better component sampler and should be selected.

Another idea is applying existing samplers in a chain-like fashion [111]. The starting sampler is a uniform one. The following samplers take a sample as input and produce another one as output. A chain is formed in such manner. This algorithm combines the advantages of multiple samplers into one. The main disad-

vantage of this approach is the increased overhead for generating samples. Another approach along the same lines is presented in [67]. This algorithm uses different samplers for different components of the robot, where different components here refers to specific features of the robot geometry. The intuition behind this is that a solution – a path in the configuration space – corresponds to a path for every point on the robot in the workspace. It can be easier to reason about the workspace since a complete representation of it is available. Sampling according to certain features of the robot in the workspace produces different samplers. Information from these samplers is then used to guide the sampling process in the configuration space. The importance given to each of the feature samplers is updated dynamically using machine learning techniques.

The hybrid algorithms discussed above only involve combining multiple algorithms during the sampling phase, and work on combining algorithms across different phases of the path planning is still limited. A recent approach in this line is a meta-algorithm that combines the online planning method and the offline shortcut and hybridization methods to find the shortest path [80]. The algorithm operates over a fixed time budget, repeatedly computing solutions using any path planner. After obtaining a new solution path, the algorithm improves the shortest known solution by alternating phases of shortcutting and hybridization. Once the time budget is exhausted the shortest solution found so far is returned. The success

of this method arises mainly because of how the path is shortened. Shortcutting performs micro-level optimizations by removing extra motions from the path, and hybridization allows for macro-level improvements to a solution path by composing a new path from large portions of the existing paths. Clearly such an alternating strategy would not be effective if one optimization was significantly better than the other.

2.5.2 Auction-based resource allocation

Automated resource allocation is a key problem in the area of multiagent systems, in which participating agents interact in both the resources each agent's activities required and the results from these activities. Because each agent has limited competence and awareness of the decisions produced by others, some sort of coordination is required to maximize the performance of the overall system [52]. One common solution to resource allocation among multiple agents is to apply well known results and insights from auction theory (e.g., [21, 83]) to represent the task and its solution. In an auction, a set of items is offered by an auctioneer in an announcement phase, and the participants can make an offer for these items by submitting bids to the auctioneer. Once all bids are received or a prespecified deadline has passed, the auction is then cleared in the winner determination phase by the auctioneer who decides which items to award and to whom. In practical ap-

plications, the items for sale are typically tasks, roles, or resources. The bid prices reflect each agent's costs or utilities associated with completing a task, satisfying a role, or utilizing a resource [29].

In the simplest kind of auction in which only one item is offered, each participant submits a bid, and the auctioneer awards the item to the highest bidder (if there is more than one winning bid, the winner is picked randomly). Alternatively, the auctioneer retains the item if no bid beats the auctioneer's price (called a reserve price). There are two common approaches to determining the sale price of the auctioned item. In a first-price auction, the sale price is the same as the winning bid; in a Vickrey auction, the sale price is the value of the second-highest bid and is intended to motivate truthful bids from the participants. The two approaches are equivalent to first-price auctions if the bidders are designed to behave truthfully. A survey on single-item auction theory can be found in [122].

In some systems bids are compared based on utilities, in which case the highest bids win auctions and the system attempts to maximize the global utility function. Utilities often encapsulate multiple factors, some representing the benefit or expected quality of task execution and others representing cost estimates. Cost estimates can also include diverse factors such as the time taken to compute solutions and the loss of efficiency caused by transitioning between tasks. Thus, utility and cost functions that combine multiple factors often require finding a reasonable set

of weights between the different components considered [29].

The process of estimating costs for bid valuation can also be difficult. Though participants in the market may have well-defined cost or utility functions, these functions still rely on having accurate models of the world state and may require computationally expensive operations. Thus, heuristics and approximation algorithms are commonly used, implying that bid prices may not always be entirely accurate. Inaccurate bids can result in tasks not being awarded to the robots best able to complete them. In this case reauctioning tasks can often improve solution quality [29].

2.6 Summary

Path planning is a key problem in Robotics. Basic path planning is a geometric problem which the ability for a robot to move from start to goal without colliding any obstacles. With the abstraction of configuration space of the robot, the problem of finding a path for a potentially high dimensional robot in Euclidean space is transformed into the problem of finding a path for a point in the high dimensional configuration space.

High dimensional problems are not tractable in general and heuristic algorithms have been developed to handle the curse of dimensionality. With probabilistic completeness, these algorithms have shown success in finding a "correct" solution

for high dimensional problems within reasonable time. Due to their probabilistic nature the sampling-based algorithms can result in highly 'non-optimal' solutions. Such issues may not be ignored in real world problems where the practicality of the paths should be taken into consideration. Therefore, the focus shifts from finding any path to finding practical or effective paths.

Many probabilistic path planning algorithms split the problem into one of identifying nodes, corresponding to robot states, along with their connectivity, and then searching the resulting graph. Often higher-level constraints are ignored in the process of path planning. Real robots are subject to requirements of constraints such as clearance, energy, smoothness and friction that sometimes need to be taken into account. These constraints are beyond the basic path planning but are important issues for real world problems. Path planning algorithms that are designed to handle these constraints are augmented to find practical paths for complex systems. There are generally three main phases in the PRM planning process that can be used to select more practical paths: the pre-processing phase (before a query is made), the post-processing phase (after a path is found), and the learning phase (after a query is made and before a path is found).

Most of these algorithms are designed to find paths with specific preferences, typically in terms of path length or clearance. However, real world problems have many different soft constraints, i.e. constraints that we would prefer to satisfy but

that are not required to be met. It is possible to leverage results from the AI planning community to define soft constraints in a standard formalism. The PDDL3 formalism is a commonly used representation because of its simplicity and generality. Using PDDL3, soft constraints are encoded into a cost function which the planner tries to minimize as it plans. This formalism is capable of representing complex preferences over planning trajectories and therefore can be applied to a broad class of problems. Given its common usage the PDDL3 will be adopted in this work but others could also be used with appropriate alternations.

Finally, in sampling-based path planning, the problem of optimizing soft constraints involves optimizing over a collection of possible constraints at various stages in the computation. An efficient and robust mechanism is needed to distribute a constrained optimization budget among a set of optimizers. We will adopt auction-based resource allocation in this work because of its simplicity and effectiveness to overcome obstacles to effective coordination including dynamic events, limited resource and the presence of adversaries. A key problem here, however, is how to identify the utility of a given optimizer for a given task and how to revise these utilities as planning goes on.

3 Path planning with soft constraints

A key question in dealing with soft constraints in path planning involves developing an effective formal representation for these constraints separate from the hard correctness requirements. Given such a formalism it then becomes possible to identify stages for optimizing the soft constraints while maintain correctness. This chapter proposes such a formalism for sampling-based path planning. Within this formalism soft constraints are applied in a systematic and principled manner to sampling-based planners. The chapter ends with the application of the methodology to some simple planning tasks. The development of planning strategies for more complex planning tasks is left until after the introduction of an appropriate resource allocation process in the following chapter.

3.1 Problem statement

Traditionally the robot path planning problem is described as the problem of finding a path for a robot to get from 'here' to 'there' while remaining in the free space

of a static environment. This definition only considers the geometric constraints that arise from collision with obstacles and is often inadequate to describe realistic path planning problems. Specifically this definition fails to distinguish between constraints that **must** be satisfied for the path to be valid, and constraints that *should* be met if possible. Here we introduce a formalism that incorporates both hard and soft constraints. This formalism allows us to address a larger range of practical problems in a principled manner. Informally speaking, given a robot with a description of its kinematics and dynamics, a description of the environment, an initial state, and a goal state, a solution to the path planning with soft constraints problem seeks to find a sequence of control inputs so as to drive the robot from its initial state to a goal state while obeying the hard (also called feasibility) constraints, e.g., not colliding with the surrounding obstacles and staying within joint limits, while optimizing the soft constraints, e.g. maintaining appropriate distance from the obstacles, minimizing rotations of joints, minimizing the path length and so on. This more general version of the problem is formalized below, building upon the traditional definition of path planning and PDDL3 to represent the soft constraints. In the name of completeness, we re-state the definition of configuration space from [69] and then expand upon this definition through an explicit identification of the hard and soft constraints associated with the problem.

Definition 3.1.1. A **configuration** c of a robot $\mathcal{A}$ is a specification of the position

and orientation of $\mathcal{A}$ in its **workspace** $\mathcal{W}$. The **configuration space** of $\mathcal{A}$ is the space $\mathcal{C}$ of all possible configurations of $\mathcal{A}$. The initial configuration c_{init} is an element of $\mathcal{C}$, and the goal region $\mathcal{C}_{goal}$ is an open subset of $\mathcal{C}$.

A configuration of $\mathcal{A}$ is typically represented as a vector of parameters of length n (n is the number of DOFs in $\mathcal{A}$). The path planning problem can be viewed as that of finding a path for a point (i.e. a configuration of the robot) from an initial point to a goal point in the n-dimensional space $\mathcal{C}$. Now we define feasible path planning using the notation of hard constraints. Let $\mathcal{HC}$ be the set of hard constraints that absolutely must be satisfied by the path, which usually includes (but is not necessarily limited to) the following:

- Obstacle collision-free: $\mathcal{A}$ must not collide with any obstacles in $\mathcal{W}$;
- Self collision-free: links of $\mathcal{A}$ must not collide with itself;
- Joint limit: the joint angles of $\mathcal{A}$ must remain within their limits;
- Differential: differential or nonholonomic constraints (if they exist) that must be satisfied for $\mathcal{A}$ to move from one configuration to another.

Definition 3.1.2. (Feasible path) Assume that time is discretized into stages of equal duration. A feasible path $\mathcal{P}$ is a sequence of configurations $(c_0, ..., c_t)$ where t is the number of stages of the path, such that $\mathcal{P}$ satisfies all the hard constraints: $\mathcal{P} \models \mathcal{HC}$.

Definition 3.1.3. (Feasible path planning) Given a path planning problem $(\mathcal{A}, \mathcal{W}, c_{init}, \mathcal{C}_{goal})$, generate a feasible path $\mathcal{P}$ such that $c_o = c_{init}$ and $c_t \in \mathcal{C}_{goal}$, if there exists one. Report failure otherwise.

This definition of feasible path planning mirrors the traditional definition of path planning found in the literature [69]. We now extend this definition through the addition of soft constraints. Here we represent the soft constraints using PDDL3 [40] although other representations would be equally suitable. The syntax for soft constraints includes two parts: (i) the identification of the soft constraints; and (ii) the description of how the satisfaction or level of satisfaction affects the quality of the resulting path. Each soft constraint is associated with a violation penalty weight such that paths that satisfy different subsets of soft constraints can be compared.

The core of sampling-based path planners involves searching for a path by sampling and testing motions connecting the samples. Thus a path is comprised of a sequence of nodes connected using local planners. It is appropriate to define soft constraint cost functions that operate on this underlying representation, such that they can be easily integrated within the different stages in the sampling-based path planning: At the node generation (sampling) level, the sampled configurations have associated soft constraints. At edge generation (node connection for roadmap-based planners or tree extension for tree-based planners) level, the local planner associates a soft-constraint with the transition between nodes. During the graph

search and post-processing phase soft constraints are specified so that among several paths solutions some may be favored over others. Therefore, based on where they are applied in sampling-based planners potential soft constraints are divided into three categories: node-level, edge-level, and global-path-level soft constraints. This structure is used to characterize different soft constraints. After presenting both the formalism and a number of example functions, a range of simple planning examples are presented illustrating the inclusion of soft constraints within the planning process.

3.1.1 Node-level soft constraints

A typical node-level hard constraint is collision avoidance, which requires the robot at the configuration to be free from the obstacles. This hard constraint is applied during node generation to validate a sample. Node-level soft constraints, however, do not check the correctness but instead capture the practicality of a configuration of the robot. Such constraints can help in selecting "good" nodes or eliminating "bad" ones in the roadmap or tree.

Consider a feasible configuration $c \in \mathcal{C}_{free}$, where $\mathcal{C}_{free}$ is the free configuration space; c meets all the hard constraints. Let $cost_s^v$ be the vertex or configuration cost. $cost_s^v : \mathcal{C}_{free} \rightarrow [0,1]$, i.e. $cost_s^v(c) \in [0,1]$ is computed for each $c \in \mathcal{C}_{free}$. The superscript v in $cost_s^v$ indicates that this is a vertex-related cost. This cost

function may be continuous or discrete. In its simplest version, the cost function $cost_s^v$ is binary, which is 0 when the soft constraint is satisfied by c, and 1 when violated. But more generally c may take on any value in $[0, 1]$ with 0 corresponding to a state that meets fully the corresponding soft constraint and 1 corresponding to a state that fully violates the corresponding soft constraint. We now discuss some examples of node-level soft constraints.

Large clearance from obstacles. Let the **clearance** of the robot at configuration c be denoted $Cl(c)$, which is defined as the minimum Euclidean distance between all points on the robot when it is placed at c and all points on all the obstacles. As clearance is one of the key issues account for the safety of mobile robots in real environments, it is often required to ensure some minimum clearance from obstacles as it navigates the environment [38]. A function $cost_{clmax}^v : \mathcal{C} \rightarrow [0, 1]$ that captures this soft constraint is defined as below

$$cost_{clmax}^v(c) = \begin{cases} 1.0 & \text{if } Cl(c) < Cl_{Lower} \\ \epsilon & \text{if } Cl(c) > Cl_{Upper} \\ (1-\epsilon) \cdot \dfrac{Cl_{Upper} - Cl(c)}{Cl_{Upper} - Cl_{Lower}} + \epsilon & \text{otherwise} \end{cases} \tag{3.1}$$

where Cl_{Upper} and Cl_{Lower} are the upper and lower bound of the clearance constraint. Note that in case of the large clearance a small cost ϵ instead of zero is still enforced. This encourages shorter paths while soft constraint costs are com-

pletely satisfied. This philosophy is also used by other soft constraint definitions that follow.

Small clearance from obstacles. Although it is common to desire that the robot stay away from obstacles for safety concerns, it may also be desirable to have the robot remain close to obstacles in some problem domains. For instance, in a hallway environment where there are random people walking by, a servicing mobile robot may be required to move while staying close to the walls to avoid possible interference with people. Similar to $cost^v_{clmax}$ we can minimize the robot's clearance by defining $cost^v_{clmin}$:

$$cost^v_{clmin}(c) = \begin{cases} \epsilon & \text{if } Cl(c) < Cl_{Lower} \\ 1.0 & \text{if } Cl(c) > Cl_{Upper} \\ (1-\epsilon) \cdot \dfrac{Cl(c) - Cl_{Lower}}{Cl_{Upper} - Cl_{Lower}} + \epsilon & \text{otherwise} \end{cases} \tag{3.2}$$

Using this definition the robot will seek to stay as close to obstacles as possible.

Joint limit avoidance. Every joint in a manipulator has travel limits which cannot be exceeded. Although the physical joint limits are hard constraints in terms of path planning, it is generally good practice to remain "away" from these limits when practical. By minimizing the joint displacement from its midpoint, joint travel limits can be avoided. A cost function for joint limit avoidance is given

by

$$cost^{v}_{jla}(c) = \frac{1-\epsilon}{n}\sum_{i=1}^{n}\frac{\mid c_i - c_{i,mid} \mid}{max(c_{i,max} - c_{i,mid}, c_{i,mid} - c_{i,min})} + \epsilon \tag{3.3}$$

where $c_{i,max}$, $c_{i,min}$ and $c_{i,mid}$ are the maximum, minimum and median permissible joint angles and c_i is the current joint angle for the i-the joint of the manipulator.

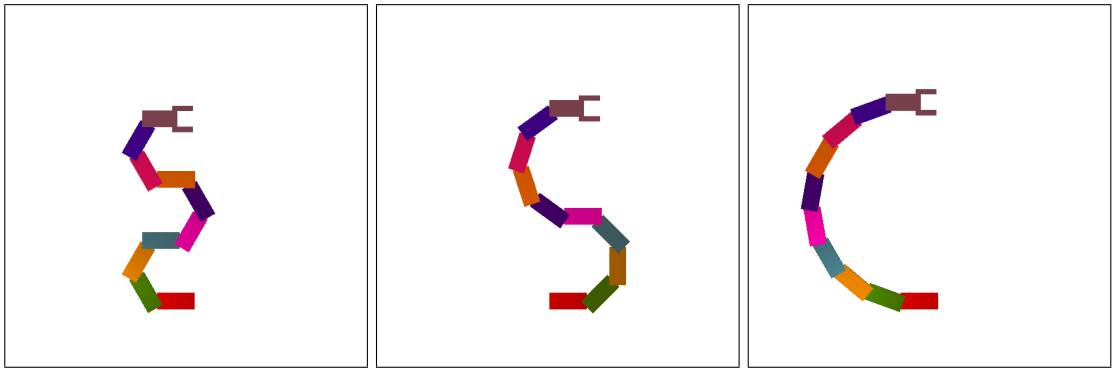
Precision It is often required to ensure the precision with which an articulate robot approaches a point or follows a path defined by the pose of the end-effector [44, 95]. Let $x \in \mathbb{R}^m$ represent the output position vector of the end-effector and $\theta \in \mathbb{R}^n$ the vector of joint angles of the robot. An infinitesimal error in the end-effector position Δx can be mapped from the joint errors Δc through the Jacobian J of the configuration c [71, 81, 62]:

$$\Delta x = J\Delta c \tag{3.4}$$

where J is a geometrically dependent structure relating the joint errors to the output errors. The Euclidean norm of the end-effector error is therefore bounded from below and above by

$$\sigma_{min} \leq \frac{\|\Delta x\|}{\|\Delta c\|} \leq \sigma_{max} \tag{3.5}$$

where σ_{min} and σ_{max} is the Jacobian's minimum and maximum singular value. A configuration with low σ_{max} implies a low error bound on the pose of end-effector. For instance, the corresponding σ_{max} values of the robot at configurations c_1, c_2 and c_3 shown in Figure 3.1 are 712, 754 and 880. This means that it is more likely



(a) $cost^v_{pee}(c_1) = 0.30$ **(b)** $cost^v_{pee}(c_2) = 0.36$ **(c)** $cost^v_{pee}(c_3) = 0.54$

Figure 3.1: Precision of the location of the end effector. A planar tentacle robot with 10 revolute joints in a planar environment is fixed to a point at the bottom of the page and each rigid link is connected by a simple revolute joint. The position and orientation of the end-effector of the robot in the three configurations are the same, but the soft constraint costs of end-effector precision are different.

for the real robot to be able to reach the pose c_1 with a given precision and least likely for it to reach c_3. A soft constraint for the generalized end-effector precision can be defined by minimizing σ_{max}:

$$cost^v_{pee}(c) = \begin{cases} \epsilon & \text{if } \sigma_{max} < \sigma_{Lower} \\ 1.0 & \text{if } \sigma_{max} > \sigma_{Upper} \\ (1-\epsilon) \cdot \dfrac{\sigma_{max} - \sigma_{Lower}}{\sigma_{Upper} - \sigma_{Lower}} + \epsilon & \text{otherwise} \end{cases} \tag{3.6}$$

where σ_{Upper} and σ_{Lower} are the upper and lower bound of the Jacobian's singular values.

Clearly, many such node-level constraints can be defined, and many may be very application specific. Let $\mathcal{SC}^v$ be the set of node-level soft constraints, and let each soft constraint s have an associated importance or violation penalty weight $w_s > 0$. Given $\mathcal{SC}^v$ with associated violation penalty weights, the cost of the feasible configuration is the weighted summation of all the soft constraints at configuration c, defined as $cost^v : \mathcal{C}_{free} \rightarrow \mathbb{R}_{\geq 0}$

$$cost^v(c) = \frac{\sum_{s \in \mathcal{SC}^v} w_s \cdot cost_s^v(c)}{\sum_{s \in \mathcal{SC}^v} w_s} \tag{3.7}$$

Let $\mathbb{P}$ denote the set of all feasible paths. Given a feasible path $\mathcal{P} = (c_0, c_1, ..., c_k)$ its node-level soft constraint cost is given by a function of the cost of the nodes that make up the path. Various combination mechanisms are possible. For example, a max norm would be useful if the soft cost is dependent upon the worst soft cost of the path. A simple sum of the constraints along the path may not be ideal as the stochastic nature of the sampling and edge linking process will produce non-uniform sampling in either configuration or Cartesian space. If path length is not critical in this computation then its soft constraint cost can be approximated as a distance averaged sum of the cost of all the nodes that make up the path to account for different sampling densities, defined as $cost^v : \mathbb{P} \rightarrow [0, 1]$

$$cost^v(\mathcal{P}) = \frac{1}{k+1} \sum_{i=0}^{k} cost^v(c_i) \tag{3.8}$$

In many cases path length needs to be considered, e.g. paths with fewer loops are

preferred over ones with more loops. The node-level soft constraint cost of a path can also be obtained as a discrete approximation of the integral of node-level costs defined as $cost^v : \mathbb{P} \rightarrow [0, 1]$:

$$cost^v(\mathcal{P}) = \frac{l}{l_{max}(k+1)} \sum_{i=0}^{k} cost^v(c_i) \tag{3.9}$$

where l is the length (Euclidean distance) of the path and l_{max} is the upper bound of l. More generally, $cost^v(P)$ is an appropriately weighted cost function f defined over the node $c_i(i = 0...k)$ and its cost $cost^v(c_i)$, i.e. $cost^v(P) = f(cost^v(c_i), c_i)$. Therefore the node-level soft constraint cost of the path is defined only in terms of the nodes along the path and their costs and nothing else.

3.1.2 Edge-level soft constraints

Edge-level soft constraints capture the practicality of a transition from one configuration to another, i.e. an edge connecting an edge (node pairing) of nodes in the roadmap or tree. These soft constraints are used to determine which pairs are preferable when building the roadmap or tree. Sampling-based path planners usually attempt to connect pairs of nodes that are close to each other because the probability that two neighbor configurations can be connected is relatively high. However, short and correct edges may not be practical for the robot to execute. Edge-level soft constraints seek to capture such soft constraints at a reasonable cost.

Consider an edge connecting two close feasible configurations (c, c') and an edge-level soft constraint s, the cost of the $cost_s^e : \mathcal{C}_{free} \times \mathcal{C}_{free} \to [0, 1]$, i.e. $cost_s^e(c, c') \in [0, 1]$ needs to be computed for each pair of configurations. There are many possible ways to define this cost function, which may involve derivatives of any order with respect to many variables, at any point along the edge. In practice it may also be necessary to capture a cost of the transition between two edges that share a node. For instance, it may not be desirable for a robot to make significant changes in trajectory as it passes through a node from one edge to another. For simplicity, in this work we only define the edge-level soft constraint in terms of edge transitions and we only consider the first-order derivative, i.e. constraints on robot velocities.

The robot velocity along an edge depends on the local planner that computes the connections. For robots with only holonomic constraints most sampling-based path planners use a simple straight-line local planner in configuration space for efficiency. For difficult problems or robots with non-holonomic constraints, a straight-line local planner may not be suitable and "smarter" but computationally more expensive local planners may be used instead. Let $\mathcal{SC}^e$ denote the set of edge-level soft constraints. Two examples are discussed below.

Small joint rotations. For free-flying robots or manipulators that possess holonomic constraints only, a straight-line local planner is often chosen to connect the

(a) e_1 **(b)** e_2

Figure 3.2: Moving a piano (rectangle) in a 2D environment. When less rotation is desired, moving the rectangle along the edge e_1 corresponding to (a) should be preferable to moving along the edge e_2 corresponding to (b).

nodes. A typical straight-line planner treats all the DOFs of the robot equally, but in practice it may be desired to minimize rotations of some of robot's joints while moving. Consider the piano mover's problem shown in Figure 3.2, where the piano is simplified as a rectangle. Intuitively we should minimize rotational movements of the piano, as such movements are harder to control than transitional movements for this problem. In the illustrated example, clearly it is more practical to move the piano along the edge e_1 whose corresponding motion is shown in (a) than the edge e_2 whose corresponding motion is shown in (b) because e_1 requires less rotation than e_2. For such problems, the edge-level soft constraint between configurations c and c' can be defined as below:

$$cost^e_{minrot}(c, c') = (1 - \epsilon) \cdot \frac{|c_i - c'_i|}{c_{i,max} - c_{i,min}} + \epsilon \tag{3.10}$$

where c_i is the joint to be minimized, and $c_{i,max}$ and $c_{i,min}$ are the upper and lower limit of the rotation.

Small steering angles. It may be desired for a car-like robot to move in a relatively straight motion rather than to make sharp turns. Let a configuration of the car-like robot be denoted by $c = (x, y, \theta)$, where (x, y) are the Cartisian coordinates of the vehicle, θ is the heading angle. Following [71] dynamics of the simple car is described by the following differential equations:

$$\begin{aligned} \dot{x} &= v \cdot cos(\theta) \\ \dot{y} &= v \cdot sin(\theta) \\ \dot{\theta} &= \tfrac{v}{L} \cdot tan\phi \end{aligned} \tag{3.11}$$

where v is the speed, ϕ is the steering angle, L is the distance between the front and rear axles of the car. If we assume that the speed v of the vehicle is constant then the steering angle ϕ describes the only control input to the vehicle and the edge-level soft constraints can be defined on ϕ. Suppose a steering input ϕ takes the robot from configuration c to c', then an edge-level cost function minimizing the steering angle $cost^e_{\phi min}$ is defined as below:

$$cost^e_{\phi min}(c, c') = (1 - \epsilon) \cdot \frac{\mid \phi \mid}{\phi_{max}} + \epsilon \tag{3.12}$$

where ϕ_{max} is the maximum steering angle of the car and ϕ is the steering angle required for the transition from c to c'.

Similar to the node-level cost functions, a weighted sum of a set of edge-level constraints represents the cost of the local path, i.e. the effort for the robot to move from one configuration to the next. Formally, given a set of edge-level soft constraints $\mathcal{SC}^e$ with associated violation penalty weights w_s, the cost of the local path (c, c') is defined as:

$$cost^e(c, c') = \frac{1}{\sum_{s \in \mathcal{SC}^e} w_s} \sum_{s \in \mathcal{SC}^e} w_s \cdot cost^e_s(c, c') \tag{3.13}$$

Given a feasible path $\mathcal{P} = (c_0, c_1, ..., c_k)$, different combination rules can be used to combine edge-level cost of all the edges along the path. For example, here the edge-level soft constraint cost of a path can be approximated as an averaged cost of the edges that make up the path, defined as $cost^e : \mathbb{P} \rightarrow [0, 1]$:

$$cost^e(\mathcal{P}) = \frac{1}{k} \sum_{i=0}^{k-1} cost^e(c_i, c_{i+1}) \tag{3.14}$$

If path length needs to be considered the edge-level soft constraint cost of a path can also be obtained as a discrete approximation of the integral of cost of the edges that make up the path, defined as $cost^e : \mathbb{P} \rightarrow [0, 1]$:

$$cost^e(\mathcal{P}) = \frac{l}{l_{max} k} \sum_{i=0}^{k-1} cost^e(c_i, c_{i+1}) \tag{3.15}$$

Again, more generally other cost sum functions are possible, with $cost^e(P) = f(cost^e(c_i, c_i + 1), (c_i, c_i + 1))$ being the general form, i.e. the edge-level soft constraint cost of a path is a function of the edges that make up the path and their corresponding cost and nothing else.

3.1.3 Global-path-level soft constraints

Global-path-level soft constraints are defined over the global path from the initial to the goal configuration rather than just the partial elements (nodes, edges) of the path. These constraints are used to select the most practical path among a set of paths, typically in a post-processing phase. Such constraints can be used to measure consumption of critical resources and other parameters, such as energy consumption and path length. Let $\mathbb{P}$ denote the set of all feasible paths. Given a global-path-level soft constraint s, the cost function of the path is $cost_s^p : \mathbb{P} \rightarrow [0, 1]$. Typical examples of global-path-level constraints include the following:

- Short distance for a mobile robot: $\mathcal{A}$ should minimize its total travel distance;
- Short end-effector distance for a manipulator robot: $\mathcal{A}$ should minimize the travel distance of its end-effector;
- Small energy consumption: $\mathcal{A}$ should consume minimum amount of energy;

Note that many of these soft constraints can be approximated using edge and node level soft constraints. It may be desirable to encode such constraints as node, edge and path constraints in order to maximize them more fully. Other global-path-level soft constraints may not be well approximated by node/edge costs, for example, number of loops in the path or similarity to some other path (different homotopic classes).

3.1.4 Putting it all together

Finally the soft constraint cost of the path is the combination of these and possibly other categories of soft constraints:

$$cost(\mathcal{P}) = w_v cost^v(\mathcal{P}) + w_e cost^e(\mathcal{P}) + w_p cost^p(\mathcal{P}) \tag{3.16}$$

Note that it is not necessary to normalize the costs over all of the different weights as we will be applying the same cost function to all potential paths, although one could certainly use weights that sum to 1 if one wanted to. We are now able to incorporate soft constraints and hard constraints within a single definition.

Definition 3.1.4. Path planning with soft constraints Given a path planning problem $(\mathcal{A}, \mathcal{W}, c_{init}, c_{goal})$, a set of soft constraints $\mathcal{SC}$ with corresponding penalty weights and a soft constraint cost function *cost*, generate a feasible path $\mathcal{P}$ such that $cost(\mathcal{P})$ is minimized. Report failure if no feasible path can be found.

This definition transforms the problem of planning a feasible path to planning an optimal path. It is not practical to search through the space of all possible correct paths for the path that minimizes the soft constraint cost $cost(\mathcal{P})$. Rather, we adapt the normal sampling-based algorithms so that at the various stages in the algorithm choices are made that work to optimize portions of $cost(\mathcal{P})$. Indeed the structure of the $cost(\mathcal{P})$ function was chosen so as to aid in this process. We now integrate soft constraints into two commonly used sampling-based path planners -

Algorithm 3.1 Generalized PRM (Roadmap Construction)

1: $V \leftarrow \emptyset$
2: $E \leftarrow \emptyset$
3: **for** $i = 1, ..., n$ **do**
4: $\quad c \leftarrow Sample_i$
5: $\quad V \leftarrow V \cup \{c\}$
6: $\quad Connect(G = (V, E), c)$
7: **end for**
8: return $G = (V, E)$

PRM [60] and its asymptotically optimal variant PRM* [58]. We begin by integrating each category of constraints individually. The process of integrating different categories of constraints within a single optimization framework is presented in the next chapter.

3.2 Probabilistic roadmaps PRM and PRM*

The probabilistic roadmap method (PRM) is designed to answer multiple path queries for a high-DOF robot in cluttered environments. There are several versions of the PRM, but they typically consist of a roadmap construction phase (outlined in Algorithm 3.1), in which a roadmap is constructed by attempting connections among randomly sampled configurations, and a query phase, in which paths connecting initial and goal configurations through the roadmap are sought. In this work we consider two types of PRMs, the basic PRM [60] that aims at quickly

Algorithm 3.2 $Connect_{PRM}(G = (V, E), c)$

1: $X \leftarrow Near(G = (V, E), c, r)$
2: **for all** $x \in X$ in order of increasing $\|x - c\|$ **do**
3: **if** $\neg SameConnectedComponent(c, x) \wedge (c, x) \vDash \mathcal{H}$ **then**
4: $E \leftarrow E \cup \{(c, x)\}$
5: **end if**
6: **end for**

establishing the connectivity of the configuration space and the PRM* [58] that tries to find asymptotically optimal paths.

The difference between the PRM and PRM* is in how a sampled node is connected to the existing roadmap. Assume that all edges are bidirectional. Following [60] the basic form of PRM node connection is outlined in Algorithm 3.2. Connections are attempted between c and other vertices in V within a ball of radius r centered at c, in order of increasing distance from c, using a simple local planner. Successful connections that satisfy the hard constraints are added as new edges to the edge set E. Since the focus of the algorithm is to establish connectivity fast, connections between c and vertices in the same connected component are avoided. Hence, the roadmap constructed by PRM is a forest, i.e. a collection of trees. However, to obtain optimal paths, extra edges within the same connected component can be useful, leading to the development of PRM*.

PRM* is a variant of PRM developed by [58] with the goal of finding asymptotically optimal paths. Outlined in Algorithm 3.3 the PRM* attempts to connect

Algorithm 3.3 $Connect_{PRM^*}(G = (V, E), c)$

1: $X \leftarrow Near(G = (V, E), c, r(n))$
2: **for all** $x \in X$ **do**
3: **if** $(c, x) \vDash \mathcal{H}$ **then**
4: $E \leftarrow E \cup \{(c, x)\}$
5: **end if**
6: **end for**

a sample within a variable radius r, defined as a function of the cardinality of the roadmap n, i.e. $r = r(n)$. The function r must be well defined such that the connection radius decreases with the number of samples and the rate of decay is such that the average number of connections attempted from a roadmap vertex is proportional to $log(n)$. In addition, connections between vertices in the same connected component are allowed. Therefore, the roadmap constructed by PRM* is not a forest for it may contain cycles. This results in a slower performance of PRM* to find *any* path connecting the start and goal, but makes it *almost-sure* to converge to optimal paths [58].

3.3 Sampling with node-level soft constraints

Here we describe the integration of the node-level soft constraints in the sampling phase of the PRM and PRM*. The basic observation here is that PRM/PRM* create potential new nodes to be added to the roadmap in an uninformed manner. Given the information contained in the node cost soft constraint function,

we can bias this process to choose nodes that have lower soft constraint costs over nodes that have higher ones. Instead of choosing completely random configurations, the sampling method with soft constraints (i.e. *SamplingSC* or *SamplingHCSC*) is called (line 4 in Algorithm 3.1) and the new configuration that minimizes the soft constraint cost function is added to the set of vertices V. *SamplingSC* and *SamplingHCSC* are two sample refinement mechanisms that integrate node-level soft constraints within the sampling phase of PRM/PRM*. There are many ways that we could implement the selection of a sample and that minimizes the soft constraint cost function. We augment the node selection algorithm – perhaps the same selection process that has been shown to work well in some specific application – and then refine the choice to optimize its node-level practicality using either *SamplingSC* or *SamplingHCSC*.

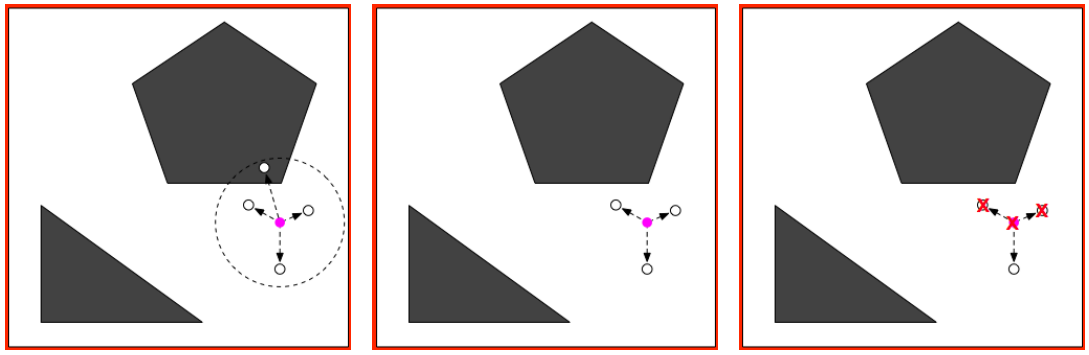
It is observed that for a collision-free node to be useful for path planning it must be part of a connected free region. Within any region we can expect some locations to be more practical than others. Ensuring that more practical nodes are chosen during the seeding process while still sampling the space sufficiently densely to construct paths is likely to improve overall path practicality, at least as measured by the node-based soft constraints. Given that a node was found to be feasible we search within a local region of this node to enhance the practicality of this node. In order to take advantage of this, the planner adjusts a node within

Algorithm 3.4 SamplingSC (random sampling with soft constraints)

1: **repeat**
2: $c_{rand} \leftarrow$ a randomly chosen configuration in $\mathcal{C}$
3: **until** $c_{rand} \models \mathcal{H}$
4: $c_{new} \leftarrow c_{rand}$
5: **for** $i \leftarrow 1, ..., NumAdj$ **do**
6: $d \leftarrow \mathcal{N}(0, r^*)$
7: $c_i \leftarrow$ a random configuration at distance d from c_{rand}
8: **if** $c_i \models \mathcal{H}$ and $cost^v(c_i) < cost^v(c_{new})$ **then**
9: $c_{new} \leftarrow c_i$
10: **end if**
11: **end for**
12: return c_{new}

its free space to states with a reduced soft constraint violation before adding the node to the roadmap. For efficiency reasons, and for generality of the approach, we follow the philosophy of randomness of the PRMs to make the adjustment. The node adjustment strategies, *SamplingSC* and *SamplingHCSC*, are outlined in Algorithms 3.4 and 3.5 respectively.

In *SamplingSC* (illustrated in Figure 3.3) for each randomly generated node c_{new} that satisfies all the hard constraints, the user specifies the number $NumAdj$ of attempts that will be made to adjust c_{new} to reduce the soft cost associated with the current sample. New samples are generated in c_{new}'s neighborhood according to the normal distribution $\mathcal{N}(0, r^*)$, where the scale r^* (similar with the radius r used to in Algorithm 3.2) is chosen based on the assumed local complexity of the configuration space. Each of these new samples is first tested for compliance with



(a) **(b)** **(c)**

Figure 3.3: Random sampling with soft constraints maximizing clearance.

the hard constraints. If the test passes then the soft constraints are applied. The valid node with minimum cost (i.e. it satisfies the soft constraints the most) from this sample is then added to the roadmap.

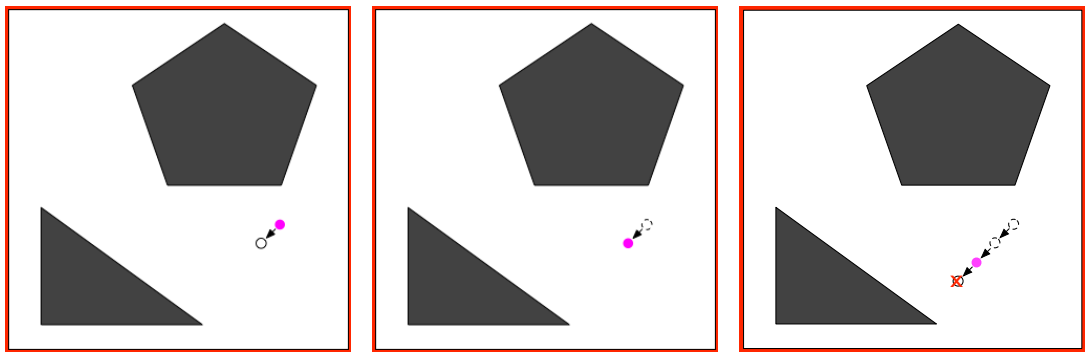
Note that during the node adjustment step $NumAdj$ nodes are not added to the roadmap. Rather, each node is augmented up to $NumAdj$ times while retaining fixed the total number of nodes. Choosing $NumAdj$ is an application-specific issue. On the one hand, $NumAdj$ should not be too small, because we want to give our planner a good chance to make an improvement. On the other hand, making $NumAdj$ too large increases the running time unnecessarily. In essence we assume that within some radius (defined by r^*) of a node, there exists a common homotopic path. We may assume a constant r^* but clearly it would be possible to set $r^* = f(c)$ for complex non-homogeneous environments or to set $r^* = g(n)$ according

Algorithm 3.5 SamplingHCSC (random sampling with hill-climbing soft constraint satisfaction)

```
1: repeat
2:    c_rand ← a randomly chosen configuration in C
3: until c_rand ⊨ H
4: c_new ← c_rand
5: u ← a random direction
6: for i ← 1, ..., NumAdj do
7:    c_i ← move c_new in the direction of u by step size d_step
8:    if c_i ⊨ H and cost(c_i) < cost(c_new) then
9:       c_new ← c_i
10:   else
11:      return c_new
12:   end if
13: end for
14: return c_new
```

to the density of the sampling. Similar to the connection radius function used in PRM* [58], r^* can be chosen as a function of n, the number of nodes in the roadmap, i.e. $r^* = \gamma(log(n)/n)^{1/d}$, where $\gamma > 2(1+1/d)^{1/d}(\mu(\mathcal{C}_{free})/\zeta_d)^{1/d}$, d is the dimension of the space $\mathcal{C}$, $\mu(\mathcal{C}_{free})$ is the Lebesgue measure (i.e., volume) of the obstacle-free space. This way, the adjustment radius decreases with the number of samples.

SamplingHCSC (illustrated in Figure 3.4) is a greedy strategy that can be considered as an alternative to *SamplingSC*. Instead of attempting to reduce the cost of a sample once, the *SamplingHCSC* iterates a maximum of $NumAdj$ steps toward a random direction u until a hard constraint is violated or a local minimum is



(a) **(b)** **(c)**

Figure 3.4: Random sampling with hill-climbing soft constraint satisfaction maximizing clearance.

reached. The random direction u incorporates all of the degrees of freedom of the robot.

Once the roadmap $\mathcal{R}$ has been constructed, finding a path between c_{init} and c_{goal} involves connecting these points to $\mathcal{R}$. The approach then follows the basic PRM algorithm. Given the nature of the **cost** function of paths, it is suitable to use Dijkstra's algorithm [63] to find the minimum cost path in $\mathcal{R}$ from c_{init} to c_{goal}.

3.3.1 An illustrative example

The environmental setup is shown in Figure 3.5, which comprises a point robot that is translating in a 2D maze. It also shows the probabilistic roadmap constructed by the basic PRM. The generated nodes are scattered throughout the free space

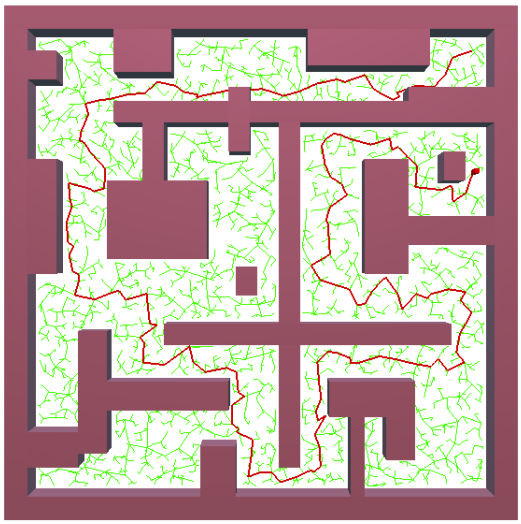
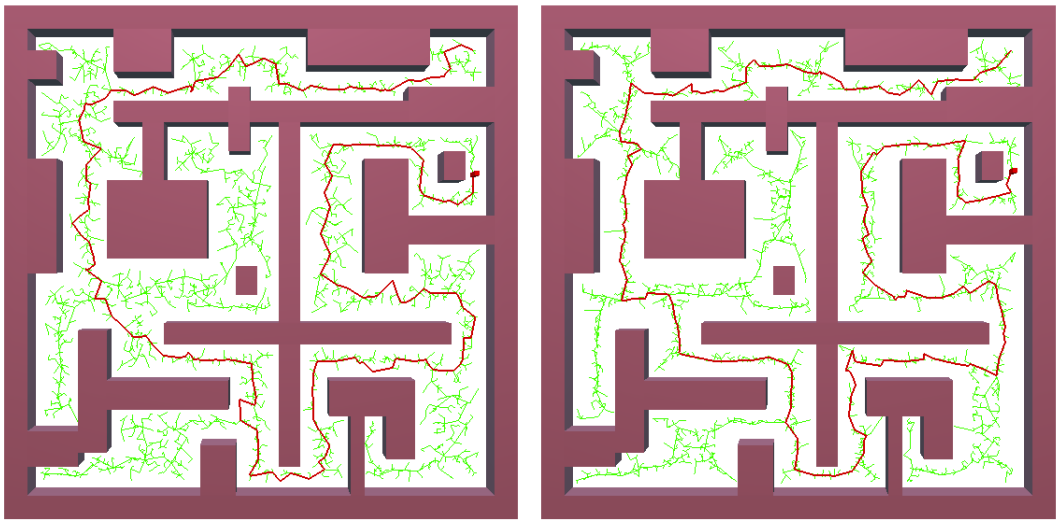


Figure 3.5: The 2D maze problem and its solution utilizing the basic PRM algorithm. This solution is to be contrasted with those shown in Figures 3.6 and 3.7 which integrate soft constraints in the solution. The roadmap and the path are computed by basic PRM. The starting location is indicated by the small rectangular square while the goal location is in the upper right portion of the image. Note the random nature of the roadmap explored by the algorithm.

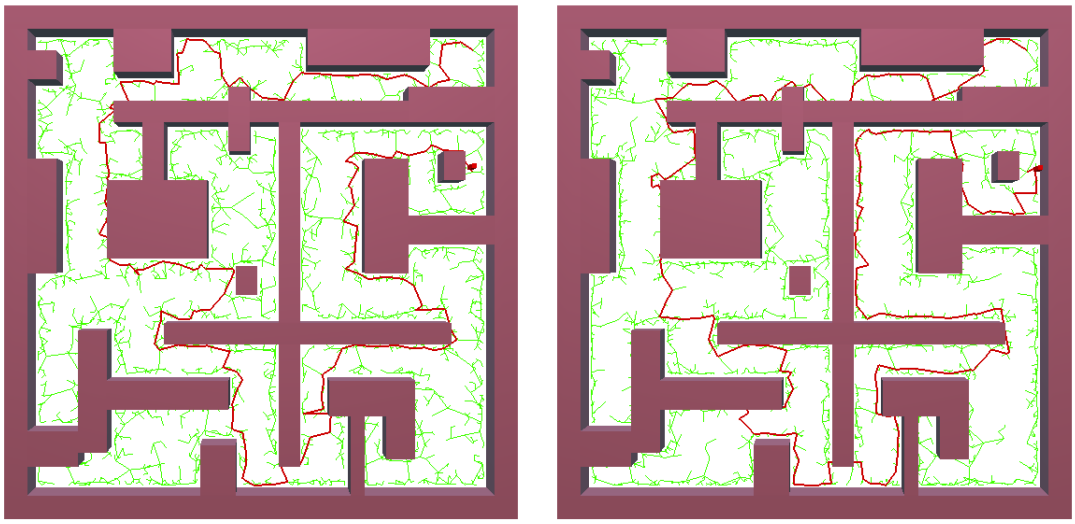
and the path is in some locations quite close to the boundaries of the free space while in other portions the path is in the middle of the free space. Figures 3.6 and 3.7 show the probabilistic roadmap and the path computed by the basic PRM with two sampling adjustment methods (SamplingSC and SamplingHCSC) with two soft constraints ($cost^v_{clmax}$ and $cost^v_{clmin}$). Both algorithms were run with the same sample sequence as the basic PRM shown in Figure 3.5. Consequently, the



(a) SamplingSC, $cost^{v}_{clmax}(P) = 0.41$ **(b)** SamplingHCSC, $cost^{v}_{clmax}(P) = 0.33$

Figure 3.6: $cost^{v}_{clmax}$ solutions identified by a soft-constraint-based PRM algorithm. The roadmap and path are computed by PRM with (a) SamplingSC and (b) SamplingHCSC. Note that the roadmap is biased more towards the center of the free space. This is especially true in (b) where the hill climbing nature of the optimization has pushed solutions towards the Voronoi graph of the space.

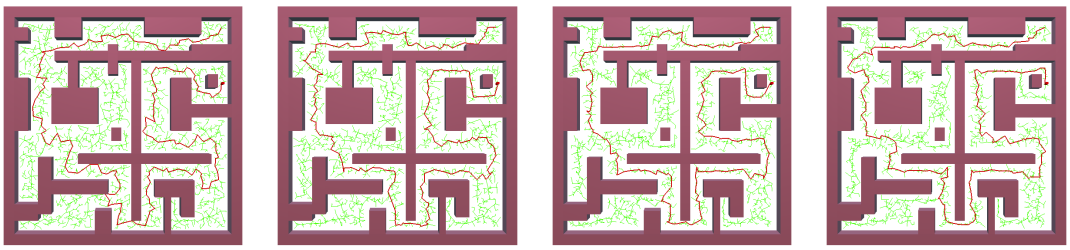
roadmaps have the same number of nodes. Only the node positions differ due to the node adjustments. Compared to the basic PRM, the roadmaps constructed by SamplingSC and SamplingHCSC using $cost^{v}_{clmax}$ tend to lie close to the medial axis of the free space, and the resulting roadmaps of SamplingSC and SamplingHCSC using $cost^{v}_{clmin}$ tend to lie close to the boundaries of obstacles.



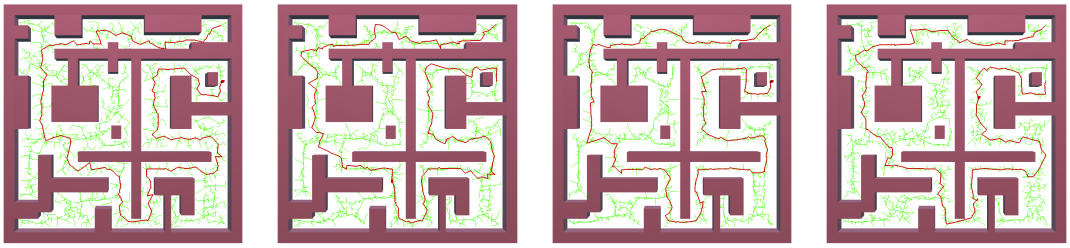
(a) SamplingSC, $cost^{v}_{clmin} = 0.48$ **(b)** SamplingHCSC, $cost^{v}_{clmin} = 0.35$

Figure 3.7: $cost^{v}_{clmin}$ solutions identified by a soft-constraint-based PRM algorithm. The roadmap and path are computed by PRM with (a) SamplingSC and (b) SamplingHCSC. Observe that the roadmap is biased towards the boundaries of the free space and that the path found remains close to workspace boundaries.

Figures 3.8 and 3.9 compare the effectiveness of SamplingSC and SamplingHCSC with different numbers of node adjustment attempts to replace a free node with one of its $NumAdj$ neighbours with most soft constraint satisfaction. As the number of adjustments is increased the found path becomes more and more localized to the Voronoi diagram.



(a) NumAdj = 2 **(b)** NumAdj = 4 **(c)** NumAdj = 6 **(d)** NumAdj = 8

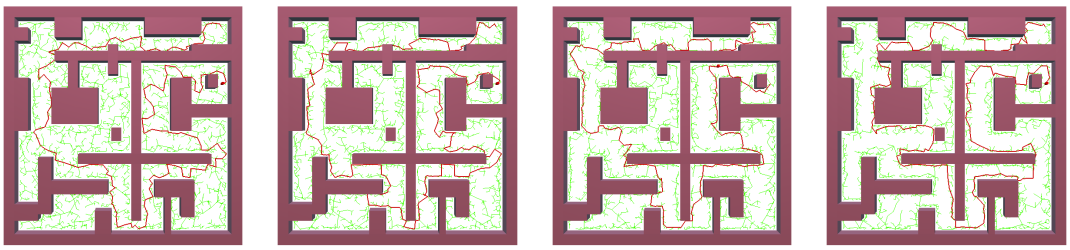


(e) NumAdj = 2 **(f)** NumAdj = 4 **(g)** NumAdj = 6 **(h)** NumAdj = 8

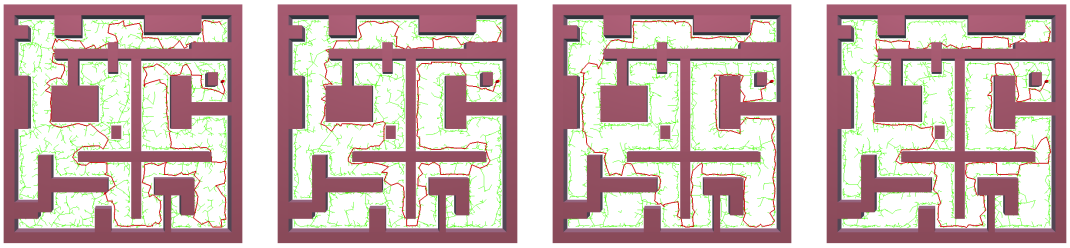
Figure 3.8: SamplingSC (first row) and SamplingHCSC (second row) using $cost^{v}_{clmax}$ with 2, 4, 6, and 8 node adjustments. The algorithms were run with the same sample sequence.

3.4 Node connection with edge-level soft constraints

This section describes the integration of the edge-level soft constraints in the node connection phase of the PRM and PRM*. Traditional node connection strategies use the Euclidean distance between two nodes as a metric to choose candidate pairs of nodes to apply the local planner. However, as discussed in Section 3.1.2, short and correct edges may not be practical for the robot to execute. Edge-level soft



(a) NumAdj = 2 **(b)** NumAdj = 4 **(c)** NumAdj = 6 **(d)** NumAdj = 8



(e) NumAdj = 2 **(f)** NumAdj = 4 **(g)** NumAdj = 6 **(h)** NumAdj = 8

Figure 3.9: SamplingSC (first row), SamplingHCSC (second row) using $cost^{v}_{clmin}$ with 2, 4, 6, and 8 node adjustments. The algorithms were run with the same sample sequence.

optimizations may be applied to enhance the practicality of the resulting paths. Here we augment the two node connection algorithms used by PRM and PRM* such that edges with a reduced soft constraint cost are added to the set of edges E in preference to edges with higher soft constraint costs.

To connect a sample c to the roadmap the basic PRM sorts the vertices that will be considered for connection in order of increasing distance from c (Algorithm 3.2

Algorithm 3.6 $ConnectSC_{PRM}(G = (V, E), c)$

1: $X \leftarrow Near(G = (V, E), c, r)$
2: **for all** $x \in X$ in order of increasing $cost^e(x, c)$ **do**
3: **if** $\neg SameConnectedComponent(c, x) \wedge (c, x) \vDash \mathcal{H}$ **then**
4: $E \leftarrow E \cup \{(c, x)\}$
5: **end if**
6: **end for**

Algorithm 3.7 $ConnectSC_{PRM^*}(G = (V, E), c)$

1: $X \leftarrow Near(G = (V, E), c, r(n))$
2: **for all** $x \in X$ **do**
3: **if** $(c, x) \vDash \mathcal{H}$ and $cost^e(c, x) < \beta$ **then**
4: $E \leftarrow E \cup \{(c, x)\}$
5: **end if**
6: **end for**

line 2). This makes sense because shorter paths are usually less costly to check for collision, and they also have a higher likelihood of being collision-free. However, such a design ignores edge-level soft constraints. Algorithm 3.6 outlines a modified PRM connection method that incorporates such constraints. In order to minimize the edge-level soft constraints, the connection process orders the potential connections in terms of their edge-level soft constraint cost (Line 2). This ensures that either edges with low cost or edges that can increase the connectivity of the roadmap will be added.

Algorithm 3.7 outlines a modified PRM* connection method that takes edge-level soft constraints into consideration. It is observed that the connection attempts of PRM* to all the nodes within a radius to c can result in an unnecessary com-

putational burden especially when many of these edges have a high soft constraint cost. To address this, an upper bound β is imposed on the edge-level cost of the candidate edges in $ConnectSC_{PRM^*}$ (Line 3), i.e. correct edges that are within distance $r(n)$ and with cost less than β will be added to the roadmap. The upper bound β controls the degree of the soft constraints applied to the problem. One way to define the bound as a variable is to follow a strategy similar to the radius $r(n)$ introduced in PRM* such that β decreases with the number of samples in the roadmap.

3.4.1 An illustrative example

This section illustrates the PRM with edge-level soft constraint algorithm described in the previous section. We consider a rectangle robot that moves in the 2D environment shown in Figure 3.10. A configuration of the robot is denoted by $c = (x, y, \theta)$, where (x, y) are the Cartisian coordinates of centre of the rectangular robot and θ is the heading angle. In this example we search for paths that minimize the change in θ by utilizing $cost^e_{minrot}$ defined in Eq. 3.10. Only edge-level optimizations are considered here.

Figure 3.11 illustrates the comparison of the basic PRM with different node connection methods $Connect_{PRM}$ and its soft-constraint-based variant $ConnectSC_{PRM}$ in the example. $Connect_{PRM}$ tries to connect pairs of nodes that are close to each

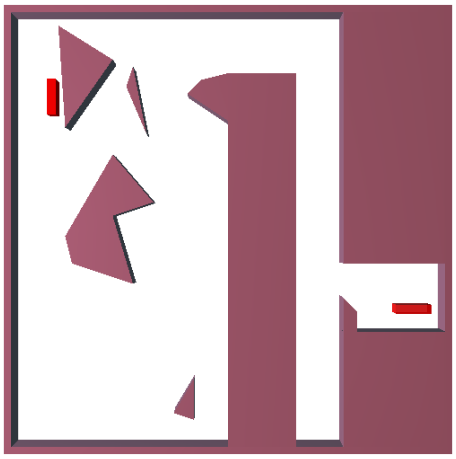
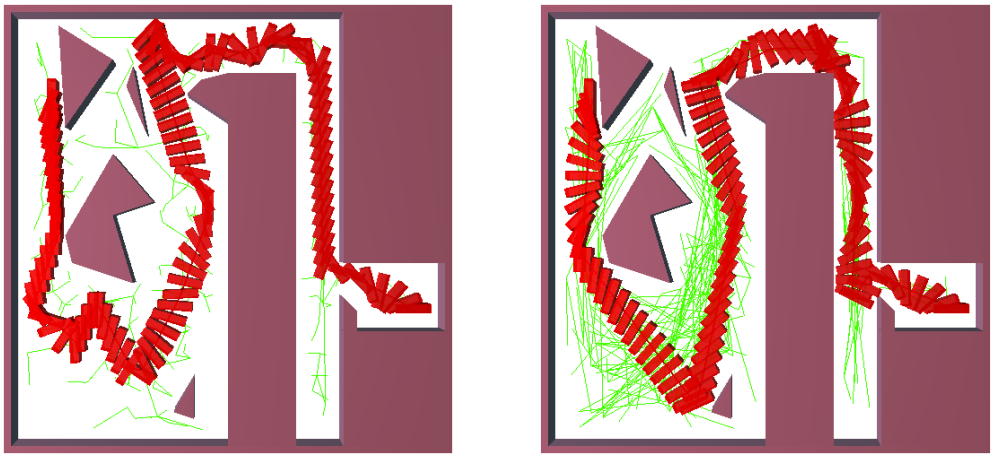


Figure 3.10: A path planning problem for a rectangle robot to move from the upper left corner to the right side in a 2D environment.

other before nodes that are farther away, so the constructed roadmap contains relatively short edges. $ConnectSC_{PRM}$ tries to connect pairs of nodes that results in a smaller change in robot orientation before those that results in a greater change in orientation. Therefore it may find longer paths than the basic PRM, but the paths shown have fewer rotations of the robot.

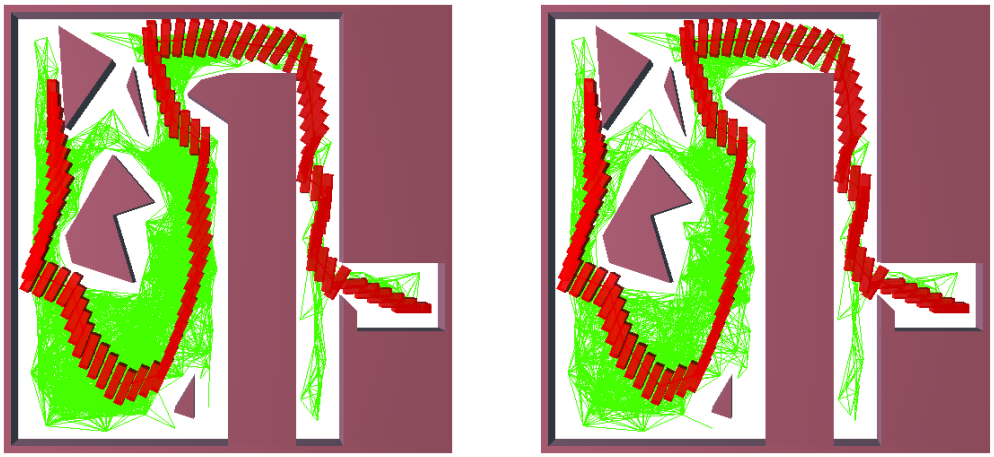
Figure 3.12 compares the PRM* with its original connection method $Connect_{PRM^*}$ and its soft-constraint-based variant $ConnectSC_{PRM^*}$. In this illustrative example, the two algorithms were run with the same sample sequence and generated the same number of nodes. With the edge-level soft constraint applied the $ConnectSC_{PRM^*}$ is able to eliminate about half of the edges that have high cost (here the upper



(a) $Connect_{PRM}$, $cost^{e}_{minrot}(P) = 0.11$ **(b)** $ConnectSC_{PRM}$, $cost^{e}_{minrot}(P) = 0.04$

Figure 3.11: Snapshots of the execution of the PRM with $Connect_{PRM}$ and $ConnectSC_{PRM}$ minimizing robot rotations. Note that the $ConnectSC_{PRM}$ algorithm results in a path with substantially smoother changes in the orientation of the rectangle robot.

bound β is set to a fixed value 0.5). Assume that the path cost is defined as the sum of the edge-level soft constraint cost and the two PRM*'s use Dijkstra's algorithm to find a minimum cost path once the roadmap is constructed, then the same path is found. Note that $ConnectSC_{PRM^*}$ may take longer to construct a connected roadmap than $Connect_{PRM^*}$ because $ConnectSC_{PRM^*}$ may miss finding some high-cost edges that could increase the connectivity of the roadmap. How-



(a) $Connect_{PRM^*}$, $cost^e_{minrot}(P) = 0.03$ **(b)** $ConnectSC_{PRM^*}$, $cost^e_{minrot}(P) = 0.03$

Figure 3.12: Constructed roadmap and path of the PRM* with $Connect_{PRM^*}$ and $ConnectSC_{PRM^*}$ minimizing robot rotations. (a) The roadmap contains 420 nodes and 15004 edges; (b) the roadmap contains 420 nodes and 7876 edges. Note that the two methods result in paths with the same soft constraint cost but $Connect_{PRM^*}$ constructs a more dense graph than $ConnectSC_{PRM^*}$.

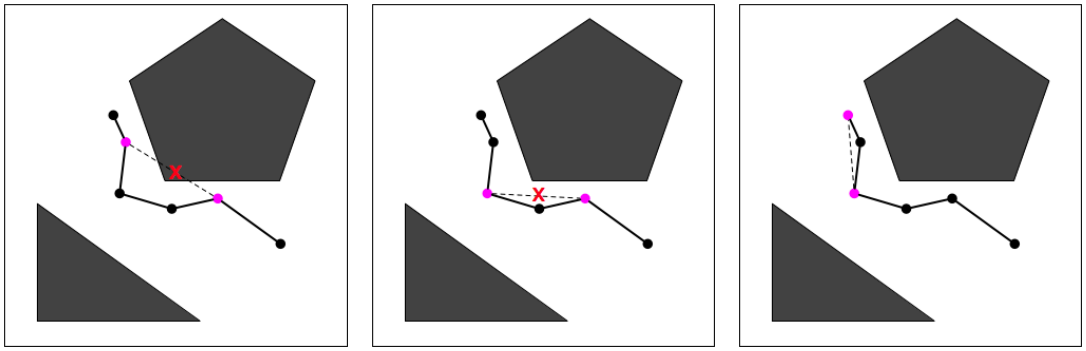
ever, the low-cost edges of the two roadmaps are exactly the same (assuming the same sampling sequence), so once $ConnectSC_{PRM^*}$ finds a path it is guaranteed to have the same cost as the original PRM* (assuming both algorithms use Dijkstra's algorithm to find a minimum cost path).

Algorithm 3.8 Shortcut with Soft Constraints (discrete path $\mathcal{P} = c_0, c_1, ...c_{m-1}$)

for $i \leftarrow 1, ..., NumAdj$ **do**
  $a, b \leftarrow$ two random indices in $[0, m)$ and $(a < b)$
  $\mathcal{P}_1 \leftarrow c_0, ...c_{a-1}$
  $\mathcal{P}_2 \leftarrow c_a, ...c_b$
  $\mathcal{P}_3 \leftarrow c_{b+1}, ...c_{m-1}$
  $\mathcal{P}_2' \leftarrow$ local path connecting c_a and c_b
  if $\mathcal{P}_2' \vDash \mathcal{H}$ and $cost^p(\mathcal{P}_2') < cost^p(\mathcal{P}_2)$ **then**
    $\mathcal{P} \leftarrow \mathcal{P}_1 \cup \mathcal{P}_2' \cup \mathcal{P}_3$
  end if
end for

3.5 Postprocessing with soft constraints

Path pruning and shortcut heuristics are common path smoothing techniques for creating shorter and smoother paths. The goal of the traditional shortcutting method is to find a shorter path that is in the same homotopy class of an existing path but that is shorter. However, such shortcuts can bring the robot close to an obstacle or violate other soft constraints. We augment the shortcut algorithm with soft constraints (outlined in Algorithm 3.8) which compares the cost of the new local path and the original part of the path before replacement. An example of the algorithm maximizing clearance along an existing path is illustrated in Figure 3.13. We expect that this method will be slower than the original heuristic as the cost comparison takes extra computing time. However, we expect that the resulting path will be more practical.



(a) Shortcut violating hard constraints is rejected

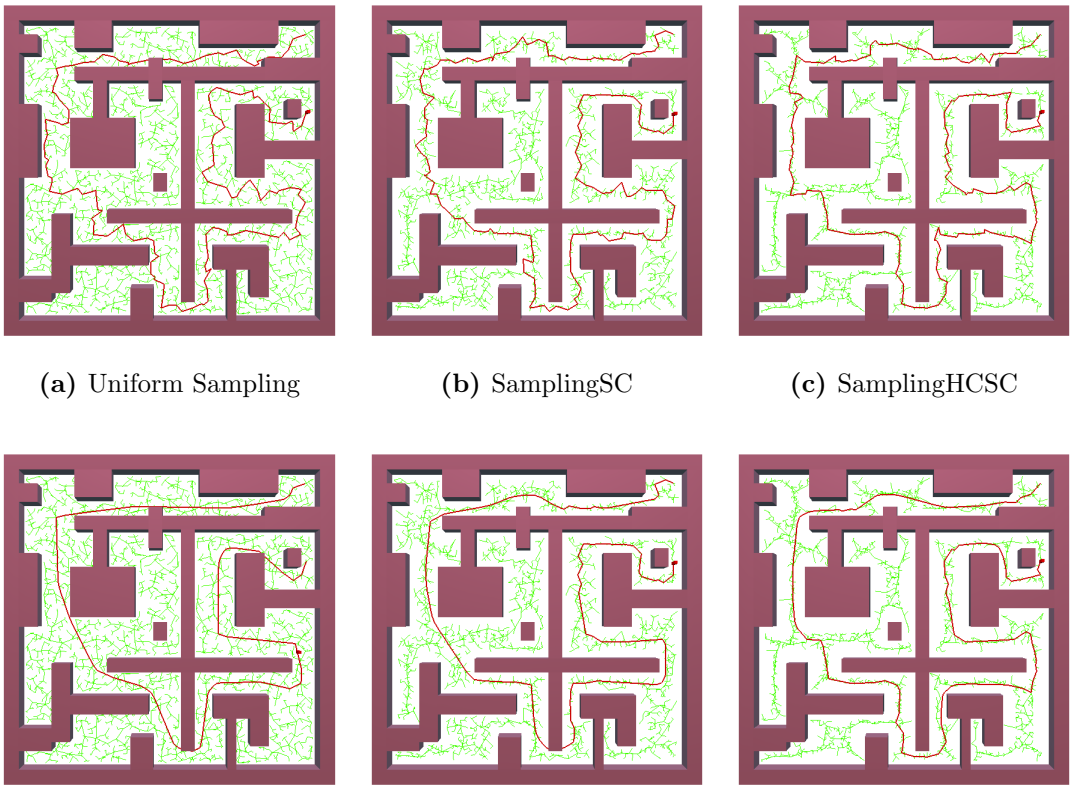
(b) Shortcut violating soft constraints is rejected.

(c) Shortcut satisfying both hard and soft constraints replaces the original sub-path.

Figure 3.13: Shortcut satisfying the hard constraints but with reduced soft constraint cost replaces the original sub-path.

3.5.1 An illustrative example

This section shows an example that applies the soft-constraint-based shortcut method to the path constructed by the PRM and its soft-constraint-based variants to see how much their path can be improved. The same environment setting is used as the one in Section 3.3.1. Figures 3.14 and 3.15 show the smoothed paths following basic PRM with uniform sampling, *SamplingSC* and *SamplingHCSC* after applying the *ShortcutSC* method with two clearance-based soft constraints, which are the discrete approximations of the integral of the node-level soft constraints $cost^v_{clmax}$ and $cost^v_{clmin}$ respectively. It shows that the ShortcutSC shortens the path by re-

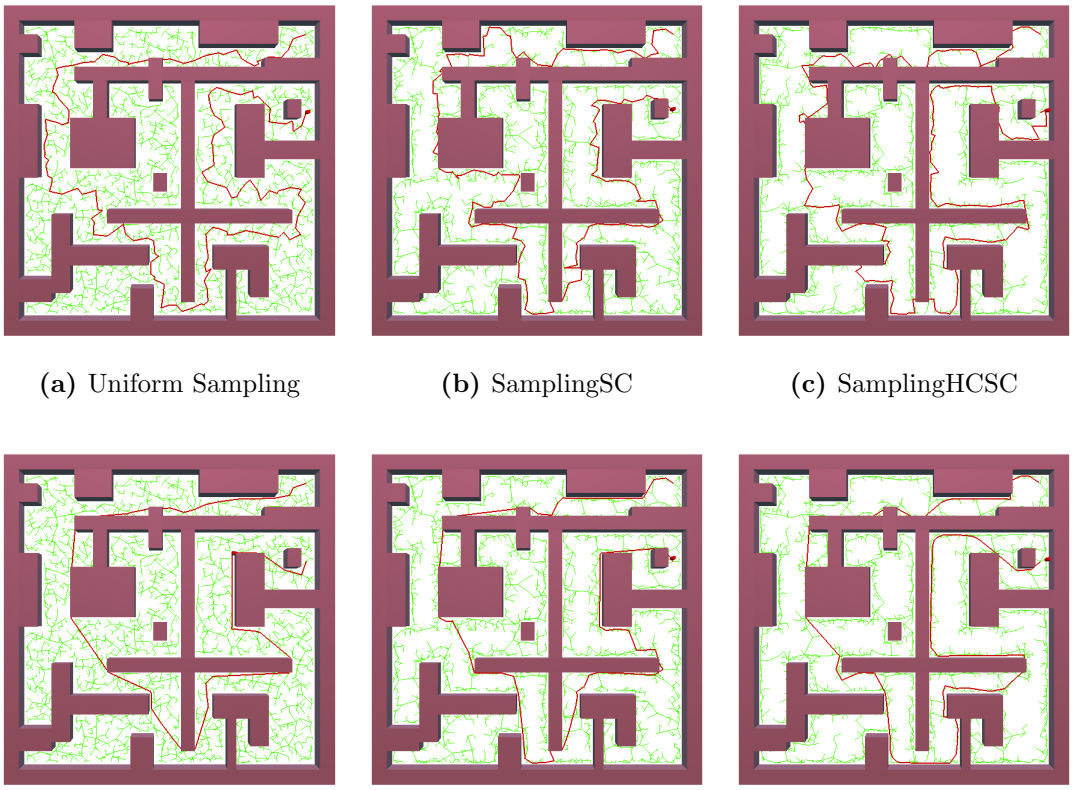


(a) Uniform Sampling **(b)** SamplingSC **(c)** SamplingHCSC

(d) Uniform Sampling followed by ShortcutSC **(e)** SamplingSC followed by ShortcutSC **(f)** SamplingHCSC followed by ShortcutSC

Figure 3.14: Shortcut with soft constraints ($cost^{v}_{clmax}$) on the path obtained by the PRM with uniform sampling, SamplingSC and SamplingHCSC. The upper row shows the original paths and the lower row shows the corresponding paths followed by ShortcutSC.

moving the extra zig-zag motions and reduces the soft constraint cost. In the two scenarios when we try to maximize/minimize the clearance, the smoothed paths

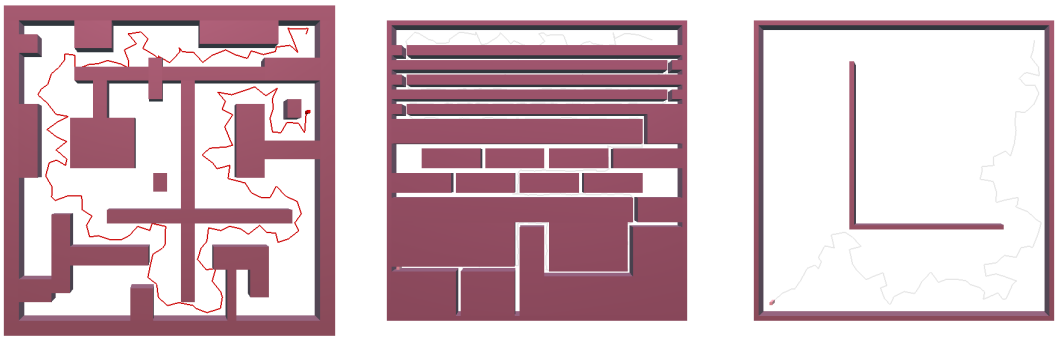


(a) Uniform Sampling **(b)** SamplingSC **(c)** SamplingHCSC

(d) Uniform Sampling followed by ShortcutSC **(e)** SamplingSC followed by ShortcutSC **(f)** SamplingHCSC followed by ShortcutSC

Figure 3.15: Shortcut with soft constraints ($cost^{v}_{clmin}$) on the path obtained by the PRM with uniform sampling, SamplingSC and SamplingHCSC. Upper row shows the original paths and the lower row shows the corresponding paths followed by ShortcutSC.

are able to stay away/close from/to the obstacles. Note that the effectiveness of ShortcutSC depends on the soft constraint considered and the original paths before



(a) Maze **(b)** Hard **(c)** Easy

Figure 3.16: Point robot in 2D environments (a) maze; (b) hard; (c) easy.

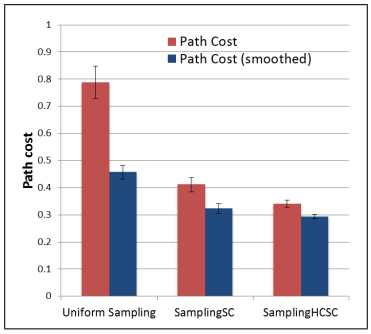
applying the shortcut optimization.

3.6 Experimental results and discussions

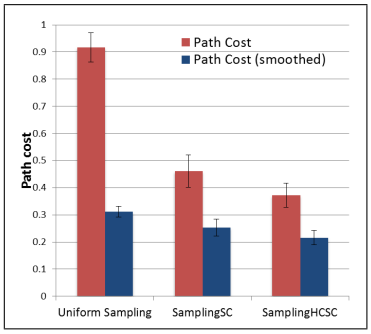
This sections presents experiments that validate the soft-constraint-based algorithms described in this chapter. The algorithms are implemented within Lavalle's Motion Strategy Library [71]. All experiments were run on a Mac running OS X with 3.06 GHz Intel Core 2 Duo processor and 2 GB memory. We show the generality of the proposed algorithms for a variety of configuration spaces and then investigate the extent to which these algorithms can improve the quality of the path by considering soft constraints. The performance results summarized in the tables and charts are values averaged over twenty independent runs.

We first present examples illustrating a point robot moving in environments that vary in their geometrical complexity (Figure 3.16). Three environments are considered, a maze with many corridors, a hard environment with narrow passages, and an easy environment with a large open space. Two node-level soft constraints (described in Section 3.3.1) are considered: $cost^v_{clmax}$ and $cost^v_{clmin}$.

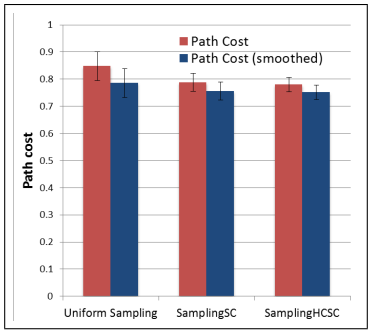
Figure 3.17 compares the average soft constraint cost of paths obtained by PRM with uniform sampling, SamplingSC and SamplingHCSC (followed by ShortcutSC) for the same amount of samples for the two soft constraints of the three 2D problems. From the figures, it is clear that the proposed SamplingSC and SamplingHCSC have improved the cost of the solution path of the PRM in all the cases. The level of the improvement depends on the complexity of the environment as well as the soft constraints. For instance, while large clearance is preferred for the 2D maze problem (Figure 3.17(a)), SamplingSC is able to compute paths with 50% less cost and SamplingHCSC gets paths with about 60% less cost, compared to the original path computed by the uniform sampling. The smoothing technique ShortcutSC reduces the cost of uniform sampling path by about 40%, SamplingSC by 20%, and SamplingHCSC by 15%. However, the smoothed uniform sampling path cost is still higher than the other two algorithms, which means the smoothing technique is not as effective as the sample adjustment technique in this scenario. While small clearance is preferred (Figure 3.17(b)), SamplingSC is also able to com-



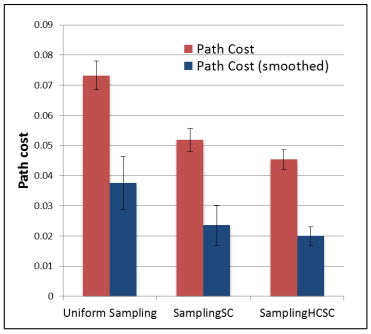
(a) 2D maze, $cost^v_{clmax}$



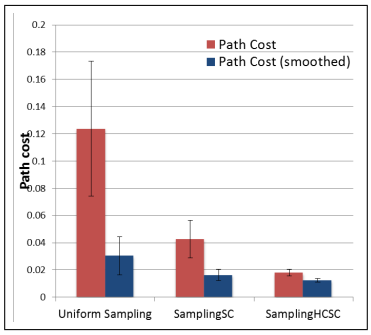
(b) 2D maze, $cost^v_{clmin}$



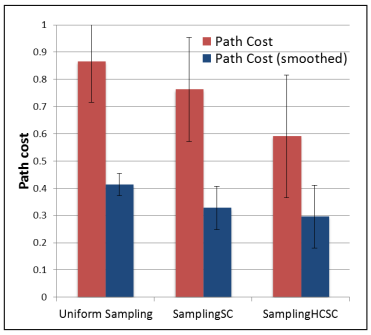
(c) 2D hard, $cost^v_{clmax}$



(d) 2D hard, $cost^v_{clmin}$



(e) 2D easy, $cost^v_{clmax}$



(f) 2D easy, $cost^v_{clmin}$

Figure 3.17: Path cost comparison for the three 2D problems. The planners' running time is 5s (maze), 30s (hard) and 2s (easy) followed by 2s post-processing time. Standard deviations are shown.

Initial seeds	N = 3000		N = 5000		N = 7000		N = 9000	
	Suc(%)	T(s)	Suc(%)	T(s)	Suc(%)	T(s)	Suc(%)	T(s)
Uniform sampling	75	1.7	95	4.6	100	8.8	100	14.5
SamplingSC, $cost^v_{clmax}$	60	2.0	80	5.2	90	9.7	95	15.6
SamplingHCSC, $cost^v_{clmax}$	35	2.1	80	5.2	95	9.7	100	15.6
SamplingSC, $cost^v_{clmin}$	75	2.1	100	5.2	100	9.8	100	15.7
SamplingHCSC, $cost^v_{clmin}$	80	2.0	100	5.2	100	9.7	100	15.9

Table 3.1: Comparative results for the 2D maze problems.

Initial seeds	N = 30000		N = 40000		N = 50000		N = 60000	
	Suc(%)	T(s)	Suc(%)	T(s)	Suc(%)	T(s)	Suc(%)	T(s)
Uniform Sampling	20	7.9	60	13.7	75	21.4	90	30.8
SamplingSC, $cost^v_{clmax}$	15	8.5	45	14.7	65	22.6	70	32.2
SamplingHCSC, $cost^v_{clmax}$	15	8.7	40	15.1	50	23.3	55	33.5
SamplingSC, $cost^v_{clmin}$	20	8.7	65	15.1	75	23.5	95	34.1
SamplingHCSC, $cost^v_{clmin}$	30	8.7	80	15.1	85	23.3	90	33.7

Table 3.2: Comparative results for the 2D hard problems.

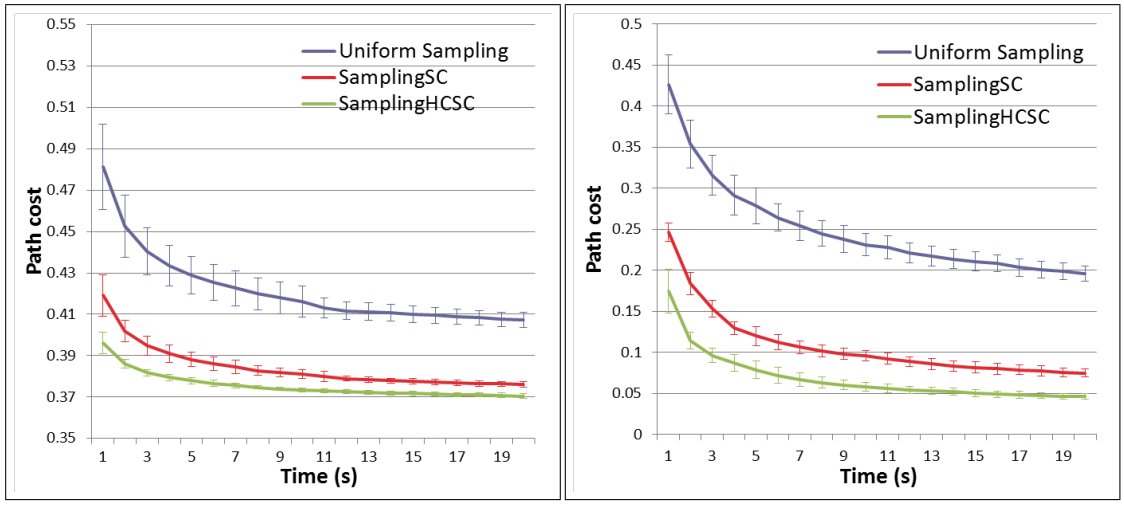
pute paths with 50% less cost and SamplingHCSC obtains paths with about 60% less cost, compared to the original path computed by uniform sampling. Note that ShortcutSC works better to minimize the clearance than to maximize it, since it reduces the path cost of the uniform sampling by about 65%, 45% and 40% for the three environments. The ShortcutSC is more effective than the sample adjustment method to find paths in small clearance.

Tables 3.1, 3.2 and 3.3 show the percentage of successful connections of the

Initial seeds	N = 1000		N = 2000		N = 3000		N = 4000	
	Suc(%)	T(s)	Suc(%)	T(s)	Suc(%)	T(s)	Suc(%)	T(s)
Uniform Sampling	75	0.4	100	1.6	100	3.5	100	6.1
SamplingSC, $cost^v_{clmax}$	90	0.5	100	1.8	100	3.8	100	6.6
SamplingHCSC, $cost^v_{clmax}$	95	0.5	100	1.8	100	3.8	100	6.6
SamplingSC, $cost^v_{clmin}$	60	0.5	100	1.8	100	3.8	100	6.7
SamplingHCSC, $cost^v_{clmin}$	70	0.5	100	1.8	100	3.9	100	6.7

Table 3.3: Comparative results for the 2D easy problems.

start and goal to the same connected component of the constructed roadmap and the average running time of roadmap construction. In general the SamplingSC and SamplingHCSC algorithms take more time to construct roadmaps than the uniform sampling approach since their node adjustment step involves an increased number of collision checks. However, the success rates of finding a path by SamplingSC and SamplingHCSC are impacted by the loss of high-cost nodes and the addition of low-cost nodes. For instance, in the 2D maze problem and the 2D hard problem while maximizing node clearance, SamplingSC and SamplingHCSC have a lower success rate than uniform sampling. The node adjustment strategies of SamplingSC and SamplingHCSC could push the nodes away from the "narrow passages" to open regions for higher clearance, so the roadmap could be disconnected due to the loss of such key nodes. However, because of the stochastic nature of the node adjustment, the algorithm may eventually discover nodes in the narrow passages as it generates



(a) $cost^v_{clmax}$ **(b)** $cost^v_{clmin}$

Figure 3.18: Path cost comparison of PRM* and PRM* with soft constraint sampling versus the running time for the 2D maze problem. The plot shows average path cost with standard deviation denoted by error bars. Results averaged over 20 runs.

more samples, just as the uniform sampling algorithm does. While minimizing clearance, both SamplingSC and SamplingHCSC have higher (or equal) success rate than the uniform sampling because they push nodes close to the boundaries of obstacles and therefore result in better connectivity of the roadmap.

It is instructive to apply the soft-constraint-based sampling methods to the PRM* [58] that has been proven asymptotically optimal. Figure 3.18 compares the path's soft constraint cost of the PRM* with uniform sampling, SamplingSC and

SamplingHCSC versus the computational time for the 2D maze problem. This plot shows that like the original PRM*, PRM* with SamplingSC and SamplingHCSC also converges to the optimal solution as the planning time goes to infinity. The soft-constraint-based sampling approaches generally return solutions with lower cost than the uniform sampling approach at the same time step.

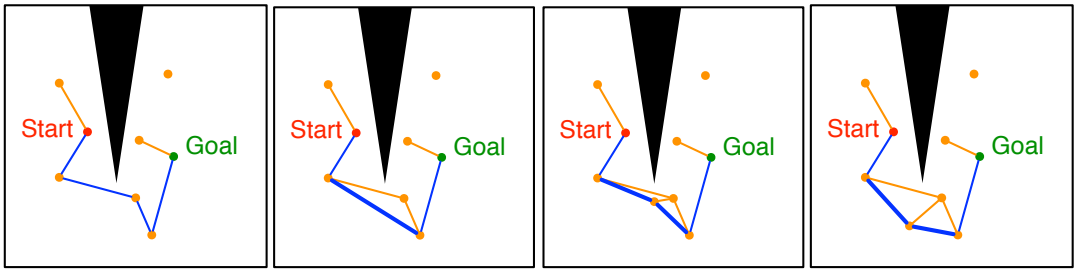
3.7 Summary

Based on where they are applied in sampling-based planners potential soft constraints are divided into three categories: node-level, edge-level, and global-path-level soft constraints. This structure is used to characterize different soft constraints. Each of the categories of soft constraints leads to a corresponding optimization within the PRM algorithm. Soft constraints defined at the vertex level can be optimized during the node generation phase. Soft constraints defined at the edge level can be optimized during the edge linking phase, while optimizations defined at the path level can be optimized at the post processing phase. Optimization strategies in each of these phases are well understood. Here we have adapted three: a node generation process that generates additional nodes to minimize node soft costs, an edge linking process that links nodes based on a cost function that incorporates soft costs associated with edges, and a global path smoothing process that incorporates soft costs. Given a set of weighted soft constraint penalty terms

and a group of corresponding optimization strategies, the key question becomes which optimizations should be performed and in what order? This is the problem considered in the next chapter.

4 Coordinating multiple optimizers: an auction-based approach

Given a specific set of soft constraints, there typically exists more than one optimization method that can be applied at a given time. Given a constrained optimization budget, how should the various optimization methods be applied to the problem? To make this problem more concrete, consider the example illustrated in Figure 4.1, where the goal is to optimize the found path to maintain a safe clearance from obstacles. As described in the previous chapter there are a number of different optimizations that could be used to make this path more practical. We could apply a shortcut method that attempts to shorten the path, or we could apply a random or hill-climbing sampling method that attempts to generate samples with higher clearance, or we could do something else. Although all of these optimizers share the same goal of efficient path optimization, they each have their own optimization strategies. For example, the shortcut method optimizes the path by adding more edges (shortcuts), while the sampling method optimizes the path by adding more



(a) **(b)** **(c)** **(d)**

Figure 4.1: A 2D example of path optimization with a safe clearance soft constraint. (a) A graph (roadmap or tree) with a connected path from start to goal; (b) The shortcut with soft constraints method is applied; (c) The sampling with soft constraints method is applied; (d) The sampling with hill-climbing soft constraints method is applied. How should we decide which optimization to apply and in what order should we apply them?

nodes. Which one(s) should we choose to apply and in what order? How do we choose to invest in the optimization alterations? Each optimization mechanism has a cost, and resources should be distributed amongst these optimizers so that an optimal (or sufficiently good) solution can be found efficiently.

Among a set of optimizers, only one can be applied to a given solution at a time. Once an optimizer is applied, the solution may change and thus so does the problem that is to be optimized in the next iteration. This optimization process can be viewed as a multi-agent sequential decision-making problem [23], where the

global objective is to maximize benefit (soft constraint path cost reduction) accumulated over time while minimizing resource consumption (computational effort). Addressing this resource allocation issue can be decomposed into two main issues that must be addressed: which optimizer to be given the resources to run at a given time; and how to ensure that the system can adapt to deal with uncertainties related to the relative performance of a given optimizer.

4.1 Multiple-round sequential optimizations

In an ideal world it would be possible to try all possible optimizers, in all possible combinations and then choose the sequence that best solves the problem. However, such an approach is computationally infeasible and thus it is necessary to choose a sequence of optimizations with imperfect knowledge of the effect of each optimization. In order to make this problem more tractable, let us decompose the global task into rounds of sequential computations, during each of which one optimizer is chosen to run for a certain amount of time. Suppose we have a collection of n optimizations $O_i (1 \leq i \leq n)$, each capable of performing a unit cost optimization on the current (world, path). Our goal here is to choose from a set of possible optimizations $\mathcal{O} = \{O_1, O_2, O_1, O_3, ...\}$ a sequence of unit-cost optimizations $O_1, O_2, O_1, O_3, ...$ that approximates the best possible optimization of the path. With a total optimization budget of T optimization rounds then there are

n^T possible optimizations that can be performed within the optimization budget. How can we approximate this optimization cost in a tractable manner?

A greedy approximation would be to try each of the n optimizers in each round, and then for that round to choose the optimization that has the best effect on the resulting path. The cost of such an approach would be $T \cdot n$. The key problem with this approach is the cost of testing each optimizer before choosing one to run. How can we improve over this greedy approach? Suppose that each optimizer O_i has an associated utility oracle $U_i(world, path)$ that estimates the optimization value obtained when optimizer O_i is run on the corresponding $(world, path)$ and that U_i's computational cost is extremely low relative to the cost of the optimization. (We imagine that the oracle's cost is negligible.) We can then imagine a multiple-round greedy bidding optimization process within which the various optimizers use their oracle to bid on the current $(world, path)$, with the winning optimizer being chosen to optimize the current $(world, path)$.

You cannot get something from nothing and it must be recognized that the utility function U_i provides only an approximation of the optimization that O_i will obtain. If the oracle functions are correct then the oracle-based greedy optimization process will perform identically to the greedy optimization process but at a substantially reduced cost. But how can we deal with the fact that the various oracles are unlikely to be perfect? The oracles may over- or under-estimate the true value

Algorithm 4.1 Dynamic optimization

1: **for** $t = 1, ..., T$ **do**
2: **for all** $O_i \in \mathcal{O}$ **do**
3: $U_i \leftarrow O_i.PredictUtility$
4: **end for**
5: $O_w \leftarrow O_i$ with highest utility U_i
6: O_w.Optimize
7: O_w.CorrectUtility
8: **end for**

of performing the corresponding optimization. How can we deal with this error? For the chosen (winning) optimizer O_w we know the actual optimization that was obtained and we can compare this improvement to the improvement predicted by the oracle. We can use this information continually to tune the various U functions, always cognizant of the requirement that U's computational cost (and the learning or refinement cost) be low.

A dynamic optimization algorithm that coordinates multiple optimizers along these lines is outlined in Algorithm 4.1. Auctions take place among the set of optimizers $\mathcal{O} = \{O_1, O_2, ..., O_n\}$ for T rounds. In each round, optimizers submit their predicted utility for a unit of optimization and the optimizer with the highest expected utility is declared the winner. After the winner O_w performs its unit-cost optimization on the current solution, the optimizers correct their expected utility U_i based on the new information acquired. The key problem of course is estimating the expected utility of each optimizer, given the current system state, and properly

recalibrating these utility estimates throughout the path finding process.

The modularity of this approach allows for flexibility in both the number and types of the optimizers available, including the pre-processing, post-processing and customized-learning techniques. This general approach is likely to capture the strength of more effective optimizers while mitigating the effects of other optimizers that are less effective for the problem. Furthermore, as the relative performance of the various optimizers changes during the overall optimization process, this can be reflected in changes in the estimated utility oracles.

4.2 Auction mechanism

At a decision point t, the objective of the system coordinator is to choose the optimizer leading to a new path with lowest soft constraint cost. Formulation of the problem in an auction system transforms the scenario from one in which each optimizer wants to use the resource to one in which an individual optimizer is allocated the resource only if his predicted utility is maximal. This mechanism accepts as input optimizers' bids of the expected utility given the resource and allocates the resource to the optimizer with the highest bid.

4.2.1 Utility function

The utility function characterizes the optimization value of each optimizer. Because the auction acts to minimize the practical path cost within budget, the utility function must closely correspond to actual progress toward this goal. There are many ways to define the utility function. One way to measure the utility of an optimizer is to measure the path soft cost improvement the optimizer achieves. It is desirable that this measure be a selective measure as we can expect substantial changes in scale of the absolute cost and improvement during the optimization process. In this work, a utility function that is equal to the expected percentage of the path cost reduction from the old path to the new path obtained by the optimizer, i.e. $U = (cost(P_{old}) - cost(P_{new}))/cost(P_{old})$ is used. The oracle does not have access to this utility function, of course, but rather produces an inexpensive "best guess" as to how the investment of a unit of optimization effort is expected to improve the solution. In this work, the expected utility is denoted as $\hat{U}$ and the true utility is denoted as U. The goal then becomes estimating the sequence of $\hat{U}$ for each optimizer in each optimization round given the winning observations.

The expression of expected utility presents a formal approach to specifying and evaluating a decision maker's preference regarding optimizers with non-deterministic outcomes. At each decision point, a rational decision should be made to maximize

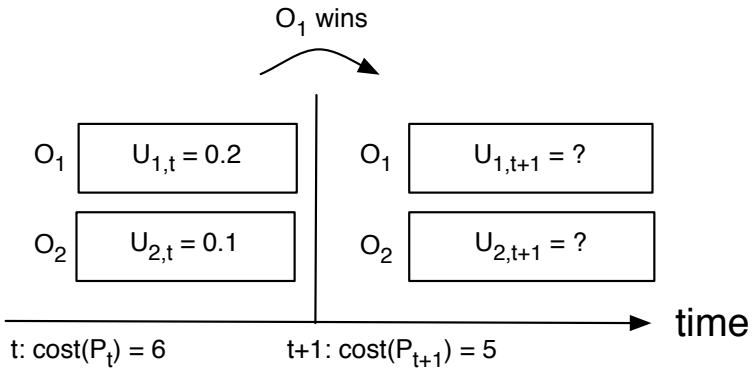


Figure 4.2: Illustration of a 2-optimizer 2-period decision problem. A decision about which optimizer to execute at t and again at $t+1$ needs to be made. The time-step "ticks forward" when an optimizer is chosen and executed.

the expected utility. Perhaps the most straightforward approach is to choose the optimizer with the largest expected utility. In order to illustrate this consider the simple problem with two optimizers O_1 and O_2 illustrated in Figure 4.2. Each optimizer has an expected utility of the optimization value of the current $(world, path)$ should it be applied in the current round. At the beginning of the problem when the cost of the current best path is 6, the initial expected utilities of each optimizer are: $\hat{U}_{1,t} = 0.2$, $\hat{U}_{2,t} = 0.1$, that is O_1 is expected to make a 20% improvement over the old path cost and O_2 is expected to make a 10% improvement. At decision point t we can simply choose O_1 for it has the best expected utility. After O_1 is executed the problem reaches time $t+1$, optimization O_1 has been performed and we have a new best path cost (suppose it is 5) although the expected utility of O_1

was 20% – which would correspond to a path cost after optimization of 4.8. We can now tune $\hat{U}_1$ based on this new information. Although O_2 was not chosen to run, its utility should also be adjusted since its augment $(world, path)$ is changed by the optimization performed by O_1. Note that the optimization may change the world - it may introduce new nodes or edges, change the way edges are constructed, etc., and the best path through the world may change as a result. Furthermore, each optimizer may be able to exploit the way in which the operation of optimization $O_1(world, path)$ performed in terms of estimating it own utility function. Now at time $t + 1$ a decision needs to be made again to choose between O_1 and O_2 based on their new utilities.

4.2.2 Bidding dynamics

The process of optimizing and updating the utility functions must be efficient. Each of the n optimizers that participate in the auction at time t places a bid. Let $\hat{U}(t) = (\hat{U}_1(t), \hat{U}_2(t), ..., \hat{U}_n(t))'$ be the $n \times 1$ vector containing the estimation of the true utility of the optimizers. Suppose an optimizer O_w wins the auction and is chosen to execute, its utility distribution $U_w(t)$ is partly revealed via the observed cost of the new path, whereas the true utility of all other optimizers remain hidden because they did not execute at time t. We now have $U_w(t)$ for the winning optimizer O_w and $\hat{U}_i$ for all of the optimizers. How should we estimate $\hat{U}_i$

at $t+1$ efficiently? We seek a straightforward way of estimating $\hat{U}_i(t)$ given the sequence of measured utility values $U_{w_t}(t)$ the winning optimizers. Let us make the strong assumption that the error between $\hat{U}_{w_i}(t)$ and $U_{w_i}(t)$ is $N(0, \sigma_i)$ and that we can treat the optimization of $U(t)$ as a recursive least squares estimation process. Following the notation of Kalman filters [57] we define the state vector by U which evolves following a linear plant model.

$$U(t) = F(t)U(t-1) + B(t)d(t-1) + w(t) \tag{4.1}$$

where $F(t)$ is the transition model (here $F(t) = I$), $B(t)$ is the control input model which is applied to the control vector $d(t-1)$. In our case we assume that $B(t) = I$ and that $d(t)$ is a term to capture the expected reduction in utility as the optimization process proceeds. Specifically $d(t)$ represents the drift rate that is expected to be negative, which can be constant or time-varying. $w(t)$ is the process noise which is assumed to be drawn from a zero mean multivariate normal distribution with covariance $Q(t)$.

At time t an observation is made of the true utility of the winning optimizer according to

$$z(t) = H(t)x(t) + v(t) \tag{4.2}$$

Here $H(t)$ is a matrix with one non-zero diagonal element corresponding to the chosen optimizer. $v(t)$ is the observation noise. $v(t)$ is assumed to be zero mean

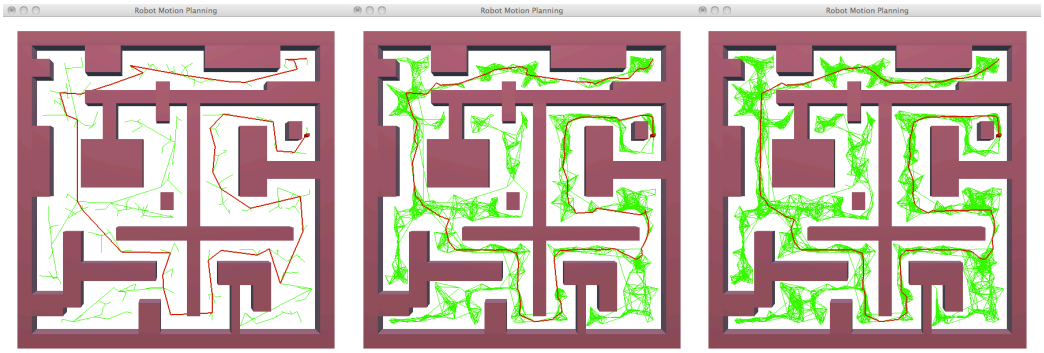
Gaussian with covariance $R(t)$. Theoretically $R(t)$ is zero, but here is implemented as a small non-zero value to model numerical errors. Under this assumption we can follow standard recursive least-squares mechanisms (e.g., a Kalman [57] or Particle [31] filter) to implement the optimization process. Here we choose to utilize a Kalman updating process.

In practice it is likely that $F(t) = I$, which provides considerable computationally efficiencies in terms of the computation of the state covariance matrix – it may prove to be diagonal under reasonable assumptions – allowing the Kalman filter to be separated for each optimizer.

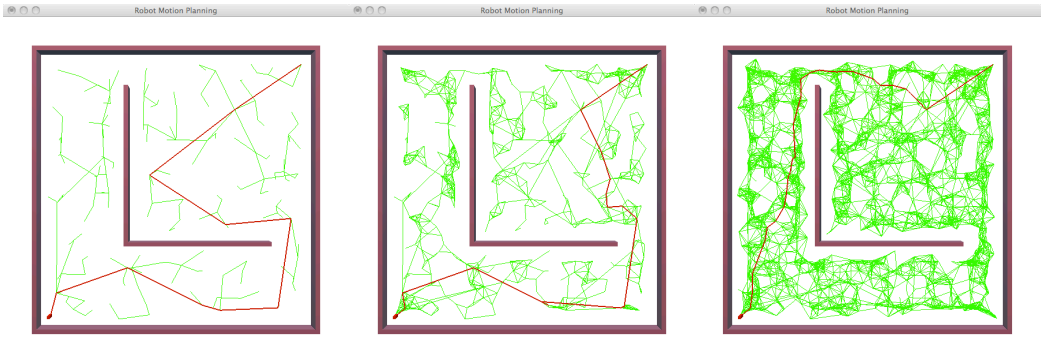
4.3 Experimental validation

Simulations are presented to show the performance of the auction-based optimization technique when planning under a time budget. Note that the overall performance of the auction-based optimization depends on two factors: the individual optimizers and the strategy for combining them. The experiments in this section are not meant to compare any two individual optimizers as the previous chapter has shown that one optimizer may be better suited for a specific soft constraint or an environment. Rather we demonstrate the benefits of the proposed auction-based strategy to combine multiple optimizers.

Experiments are reported utilizing the three optimizers introduced in Chapter 3:



(a) $cost(P) = 0.81$ **(b)** $cost(P) = 0.60$ **(c)** $cost(P) = 0.58$



(d) $cost(P) = 0.60$ **(e)** $cost(P) = 0.46$ **(f)** $cost(P) = 0.45$

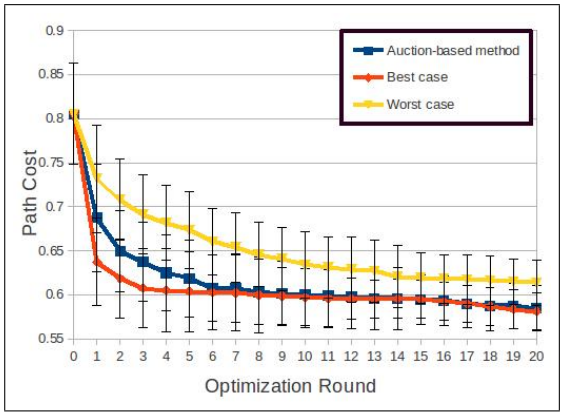
Figure 4.3: Illustration of the auction-based optimization on the 2D maze and 2D easy problems at the initial, middle and final step of the optimizations. The soft constraints considered are equally weighted $cost^v_{clmax}$ and $cost^p_{len}$.

PRM* with SamplingSC, PRM* with SamplingHCSC and ShortcutSC. Note that here we choose PRM* [58] rather than the basic PRM [59] as a base planner to integrate the soft constraint-based optimization strategies. This is because neither PRM nor PRM with soft constraints is asymptotically optimal. However, PRM*

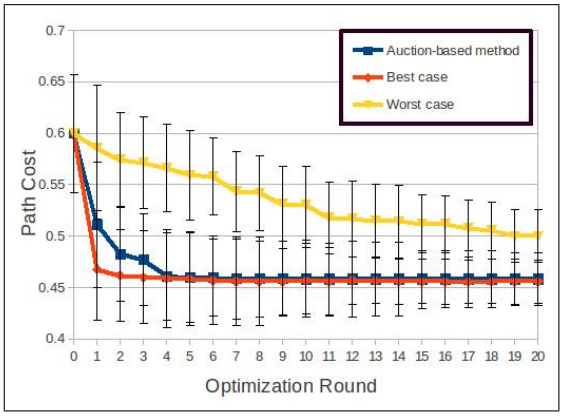
and PRM* with soft constraints expect to make continuous improvement towards the solution as they progress. As the auction-based approach is formulated as a multiple-round optimization process that runs for a range of run-times, PRM* and its soft-constraint-based variants fit better within this framework.

The efficiency of the auction-based optimization is evaluated in instances of a point robot in three 2D environments shown in Figure 3.16 and described in Chapter 3. Three proposed optimization techniques are combined in the auction-based framework: the PRM* with SamplingSC (PRM*SC) that perturbs each randomly sampled new node in its neighborhood for lower cost before adding it to the roadmap and then connects it to its neighbors with a varying connection radius; the PRM* with SamplingHCSC (PRM*HCSC) that is a greedy version of PRM*SC; and the ShortcutSC.

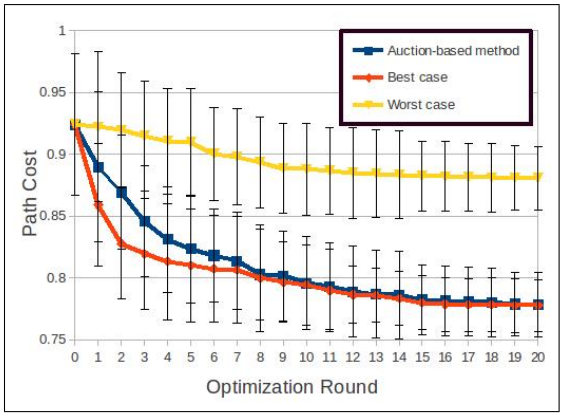
Two different types of soft constraints are considered in these experiments, a node-level soft constraint $cost^{v}_{clmax}$ and a global-path-level soft constraint $cost^{p}_{len}$. Each soft constraint needs a numerical weight representing its importance with respect to the other one. Depending on the requirements of the problem the weights should be specified properly by the user. Here let the weights for both of them be 0.5. Figure 4.3 illustrates the auction-based optimization in action on the maze and easy problems with different computational budgets. The constructed roadmap and computed path are shown at the initial, middle and final steps during the



(a) 2D Maze



(b) 2D Easy



(c) 2D Hard

Figure 4.4: Performance of the auction-based approach in the three 2D environments. Upper and lower bounds are shown. The plot shows the average solution soft constraints costs with standard deviation versus the optimization rounds. Results averaged over 20 runs.

optimization. As the optimization progresses, more nodes and edges are added to the roadmap and the path is refined to be shorter and further away from obstacles.

Problem	Resource distribution (ShortcutSC: PRM*SC : PRM*HCSC)
2D Maze	56 : 20 : 24
2D Easy	71 : 12 : 17
2D Hard	36 : 33 : 31

Table 4.1: Resource distribution among optimizers by the auction-based approach in the 2D problems. Average percentages of times for each optimizer to be selected as a winner are compared.

Figure 4.4 compares the performance of the auction-based approach with the best case and the worst case in each round of the optimization. The best/worst case is computed by trying all the three participating optimizers at every round before choosing the one that improves the path cost the most/least. This increased the computation time to three times of the original time, but provides us an idea of how well our auction-based approach performs. These charts plot the average path cost with standard deviations in each round of the optimization. The results show that the auction-based approach performs significantly better than the worst case and quickly converges to the best case as the optimization progresses. It also means that the utility function defined in the auction-based approach well approximates the right decisions.

In addition, the statistics of the resource distribution among the optimizers in the auction is shown in Table 4.1. In the "maze" world and the "easy" world, ShortcutSC obviously outperforms PRM*SC and PRM*HCSC because ShortcutSC is more effective in discovering short and high-clearance paths in environments with many open spaces. In the "hard" world, however, there is no absolute winner among the three optimizers, because it is equally difficult for them to make any improvement.

An important issue is when to terminate the algorithm. Clearly the user can specify a time budget based on the nature of the problem and run the algorithm till the end of the time runs out. However, when such a budget is not clear we would want some mechanism to decide when to terminate the algorithm based on its performance. In each optimization round of the algorithm, we only execute one optimizer that has the highest expected utility. This can lead to an update of the $(world, path)$. Even though an optimizer may not make an adequate improvement of a solution at one point, it may make a big improvement in the future after the $(world, path)$ changes. For example, a shortcut method may stop improving a path after a while, but it will work again once another optimizer finds a path in a different homotopy class. Therefore, we terminate the algorithm when the improvement of the solution cost in k consecutive rounds is smaller than some threshold δ and every optimizer O_i has been selected at least $k_i (1 \leq k_i \leq k)$ times in these rounds.

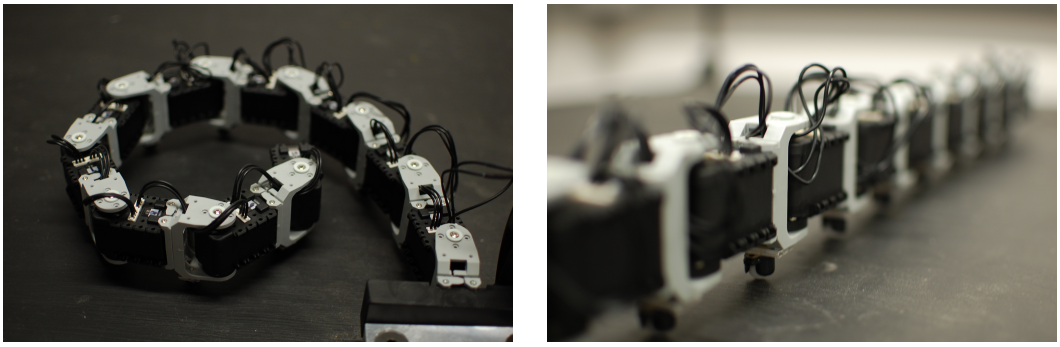
4.4 Summary

Many optimization strategies have unique strengths in addressing different types of soft constraints in robot path planning, but it is difficult to determine which optimizer is most suitable at a certain time for a certain problem. This chapter presents a multiple-round auction-based approach that allows multiple optimizers to compete in a 'market' for computational resources. This approach is online and adaptive and the optimizer with the best predicted performance is dynamically selected as the process goes on. Specifically, each optimizer adjusts its expected utility distribution downwards based on the utility observed thus far across all the optimizers and not just its own utility. The auction-based approach was shown to be effective at distributing computational resource among multiple optimizers on a point robot in various 2D environment. The following chapter applies the algorithm to more complex scenarios involving two types of high-DOF tentacle robots.

5 Path planning and tentacle robots

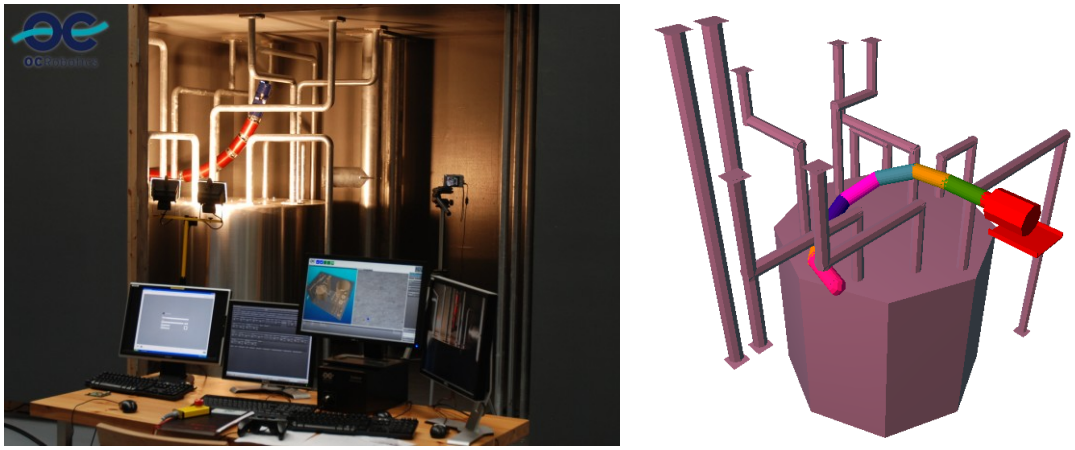
Chapters 3 and 4 developed a framework within which plans can be developed for high DOF robotic systems within a computational budget that meet the hard constraints of path feasibility and at the same time seek to reduce the cost of domain-specific soft constraints. This framework is intended to be robot independent, but in this chapter the approach will be grounded in the capabilities and tasks associated with tentacle robots. In particular, this chapter provides details of the application of the theoretical framework on a planar tentacle robot built from Dynamixel servo motors (Figure 5.1) and a simulated tentacle robot performing a nuclear inspection task (Figure 5.2). All algorithms were implemented in C++ within Lavalle's Motion Strategy Library (MSL) [71] and run on a computer with 2.0GHz x 4 CPU and 7.7 GiB memory running the Ubuntu operating system.

In the problem of path planning for tentacle robots, the hard constraints include: (1) each joint of the robot must stay within their limits; (2) collisions of the robot and the obstacles must be avoided; (3) self-collision configurations where two links of



(a) **(b)**

Figure 5.1: A planar tentacle robot created from ten Dynamixel AX-12 servos.



(a) **(b)**

Figure 5.2: Tentacle robot in a hot cell in (a) real mock environment and (b) simulation. The picture (a) is ©OC Robotics and is used with permission.

the robot intersect each other must be avoided. The need to properly represent the soft constraints is particularly important for tentacle robots. For these devices the

high number of DOFs provides the opportunity to deal with complex environments and to produce solutions that are not only correct (e.g., they grasp the object through free space, for grasping tasks) but also optimize other requirements of the problem space. This chapter describes experiments of our path planning approach with soft constraints on simulated and real tentacle robots. After identifying the soft constraints to be considered in the problem, proper optimization strategies are chosen and combined in the auction-based system. It demonstrates that this approach is an effective enhancement to the basic probabilistic planner with uniform sampling to find practical paths.

5.1 Planning for a real planar tentacle robot among obstacles

5.1.1 Experimental setup

These experiments are related to a planar tentacle robot built from ten Robotis Dynamixel AX-12 servos shown in Figure 5.1. The robot is approximately 67cm long when it lies straight. One end of the robot is fixed and rollers have been installed below each joint to reduce the friction between the robot and the table top. A configuration of the robot can be described as a vector of its revolute joints $(\theta_1, \theta_2, ...\theta_{10})$. Each revolute joint has defined an internal joint limit $[-90°, 90°]$,

which expresses the range of the possible orientations that the corresponding link can take relative to its previous one. Experiments were conducted with this robot running the Robot Operation System (ROS) as well as a simulation of the robot.

The goal of these experiments is to demonstrate the execution of the real robot following the computed paths with soft constraints applied. In an ideal world a robot should remain in the free space of the environment while executing a computed collision free path, but this may not be the case in reality because of various uncertainties (refer to Section 2.1.1). Uncertainty in the pose of the end effector given the state of the device is an issue in real devices. Given this reality it is desirable to be able to create paths that do not move the robot through the free space of the environment but it is also desirable that the robot will be able to follow these paths accurately. We assume that the environment is static and known, i.e. there is no uncertainty in the robot's environment. Given these assumptions the main uncertainty with this tentacle robot comes from the robot's joints, including position sensor error, control error, and deflection due to joint compliance. Further error is introduced by the nature of the motor controller. An independent joint position controller is used to control the robot since the main purpose of this experiment is to test the path planner not the controller. The controller does not take into account the dynamics of the robot links and treats each joint as an independent, uncoupled system.

5.1.2 Optimizing individual soft constraints

There are a number of different soft constraints that are appropriate for the tentacle robot. We begin by considering three such constraints individually and then apply the full soft-constraint algorithm to all three constraints concurrently. The first soft constraint maximizes the clearance from obstacles ($cost^v_{clmax}$ defined in Eq. 3.1) and the second avoids joint limits ($cost^v_{jla}$ defined in Eq. 3.3). Beyond these soft constraints it is critical to consider precision of the pose of the end effector. Lack of precision is a major problem for the planar tentacle robot because each AX-12 servo is expected to have some amount of error and the errors of servos propagate through the links to the end-effector. Minimizing the effect of this error propagation is a critical issue for the robot to accomplish tasks and this precision issue becomes critical when the robot has ten joints. Here we also consider the precision soft constraint ($cost^v_{pee}$ defined in Eq. 3.6) and try to find paths that the robot can follow more precisely. The difference with respect to $cost^v_{pee}$ is obvious in the case shown in Figure 5.3 where there is no obstacle in the scene. Lacking such soft constraints the basic planner computes the path shown in Figure 5.3(a) that identifies the direct path from the start to the goal. Considering $cost^v_{pee}$ results in a more complex solution as shown in Figure 5.3(b). The effectiveness of the method was then executed on the real tentacle robot. See Figure 5.4 for snapshots from the execution

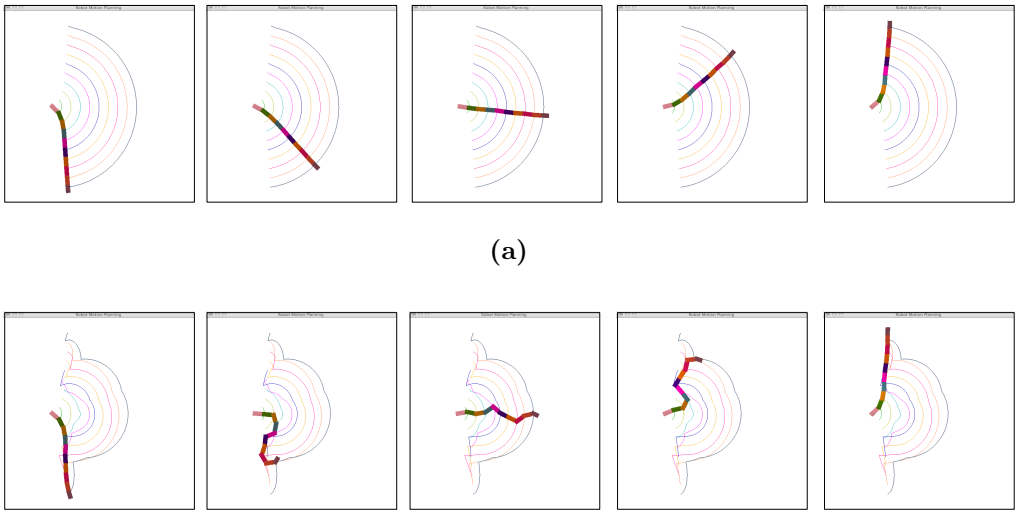


Figure 5.3: Path planning considering the precision of the end effector. (a) shows the solution identified by the basic planner; (b) shows the path found when the precision of the end-effector is considered in planning. In both cases the environment is considered to be free of obstacles.

of these different paths.

In the first experiments only one single soft constraint is considered at a time. Comparisons between the basic path planning and path planning with soft constraints are shown in Figure 5.5. The scene shows a simulation of the tentacle robot operating in a workspace comprised of two obstacles. The path planners compute paths that take the robot from the lower space to the upper one while avoiding obstacles. Figure 5.5(a) shows a path computed by the traditional PRM

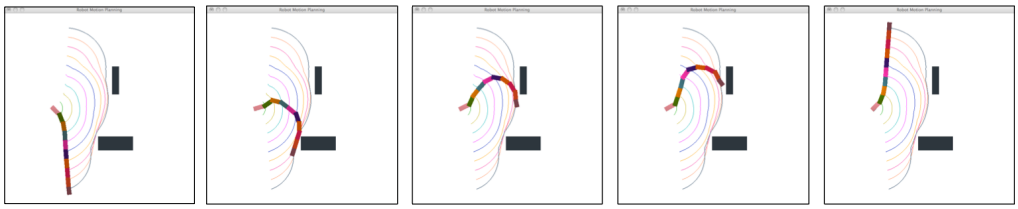
(a)

(b)

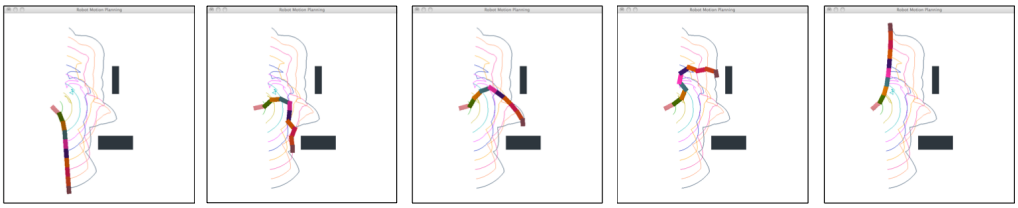
Figure 5.4: Snapshots of the tentacle robot executing the paths computed by: (a) basic path planning and (b) Path planning with soft constraints maximizing the precision of end-effector.

with uniform sampling followed by the traditional shortcut method. The path is correct and is relatively short. Figure 5.5(b) shows a path computed by the soft-constraint path planner minimizing $cost^v_{clmax}$, which encourages to keep the robot away from the obstacles. Figure 5.5(c) shows a path minimizing $cost^v_{jla}$, which tries to minimize the deviation of each joint. Figure 5.5(d) shows a path minimizing $cost^v_{pee}$.

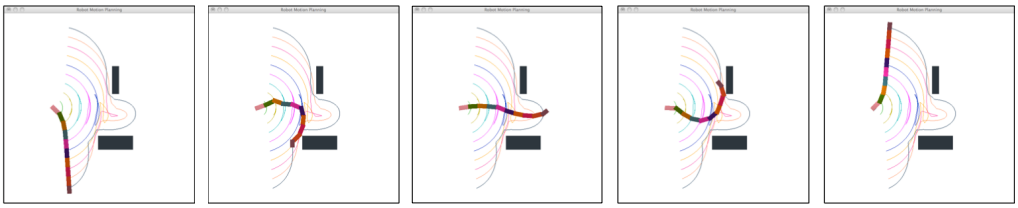
To test the performance of the optimization strategies developed in Chapter 3 on the individual soft constraints, we compared results obtained using individual soft constraints against those obtained with the basic PRM. Specifically, we compared



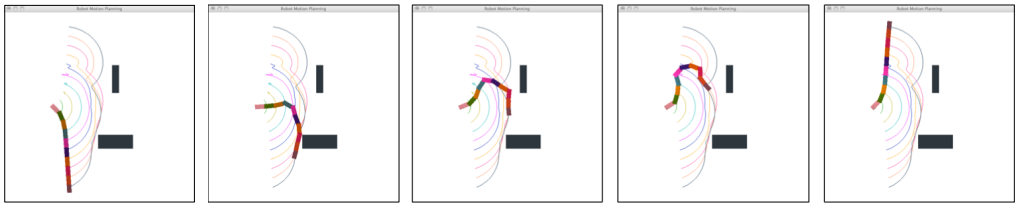
(a)



(b)



(c)

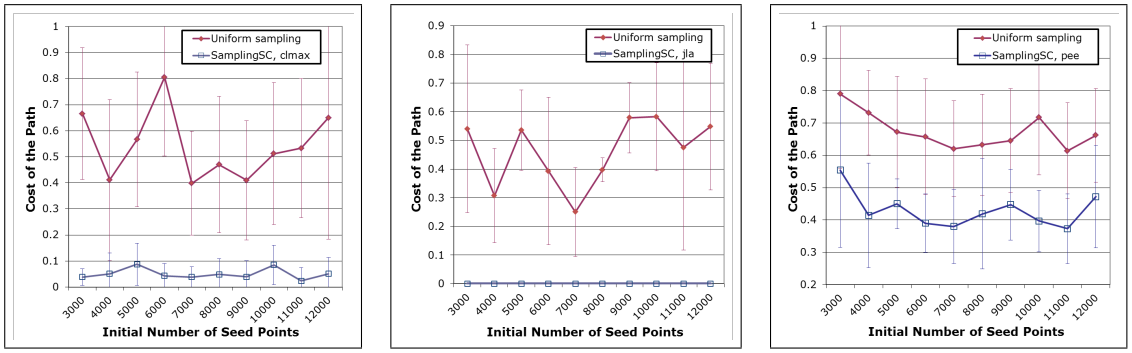


(d)

Figure 5.5: Path planning in a simulated workspace with two rectangular obstacles. Comparison between path planning (a) with no soft constraint; (b) keeping safe clearance from obstacles; (c) avoiding joint limits; (d) maximizing precision of the end-effector.

the *SamplingSC* algorithm with a single soft constraint applied against the PRM with uniform sampling. Results are provided in Figure 5.6. By sampling with soft constraints the average cost of the path is decreased when compared to the basic PRM in all three cases. The *SamplingSC* method also shows more stability (measured in terms of the cost standard deviations) than the uniform sampling in the path quality they achieve. Figure 5.7 compares the running time associated with constructing the PRM with uniform sampling and the roadmap with different soft constraints for the test model. Roadmap generation with soft constraints does take more time to compute than the uniform sampling since the node adjustment involves an increased number of collision checks and computation to check the soft constraints.

The performance of optimization strategies developed in Chapter 3 varies depending on the degree of optimization applied, such as the number of adjustments attempts in the sampling phase (i.e., the number k) and the number of shortcut attempts taken by the shortcut procedure. Figure 5.8 shows results of the three algorithms (*SamplingSC*, *SamplingHCSC*, and *ShortcutSC*) on the simulated planar tentacle robot in a workspace consisting of two obstacles. Figure 5.8(a), (c) and (e) show that the average soft cost of the path as measured in terms of violations from soft constraints is reduced when the degree of optimization increases, i.e. more node adjustment or shortcut attempts are made. The optimization methods reduce



(a) **(b)** **(c)**

Figure 5.6: Cost comparison of the paths found by PRM with uniform sampling and *SamplingSC*, when the soft constraints are: (a) $cost^v_{clmax}$; (b) $cost^v_{jla}$; (c) $cost^v_{pee}$. Cost of the path is measured as a discrete approximation of the integral of the node-level soft cost of the configurations along the path. Results are averaged for 20 independent runs for each case. Standard deviations are shown.

the path cost quickly at first, while further optimization efforts have a reduced rate of return. Clearly the shortcut method outperforms the two sampling methods in reducing path cost. This is because the shortcut method takes a given path as an input so its task is more focused than sampling methods which try to optimize the nodes of the entire roadmap. Therefore, the sampling methods aim at generating a "better" roadmap, which can be beneficial for multiple queries of paths.

Fig. 5.8(b), (d) and (f) show the running time of the three optimization methods. Clearly the time increases with the degree of optimization because of the computa-

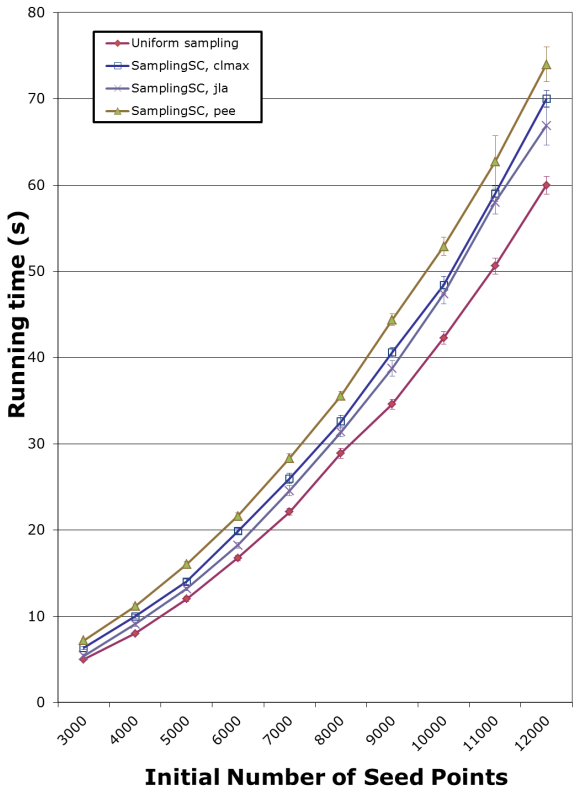
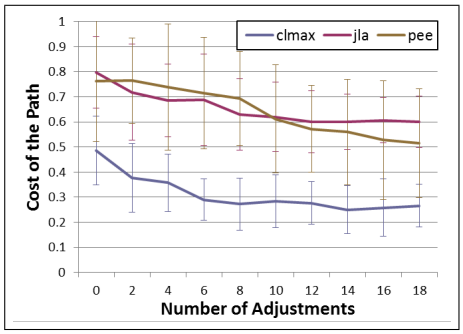
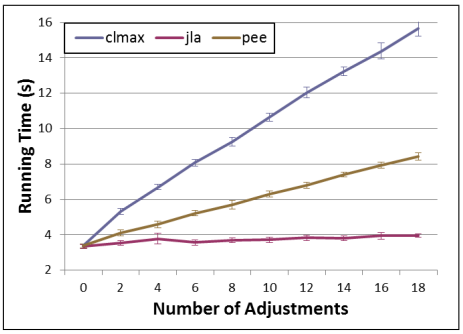


Figure 5.7: Comparison of the running time of the PRM with uniform sampling and *SamplingSC* with soft constraints being $cost^v_{clmax}$, $cost^v_{jla}$ and $cost^v_{pee}$. Results are averaged for 20 independent runs for each case. Standard deviations are shown.

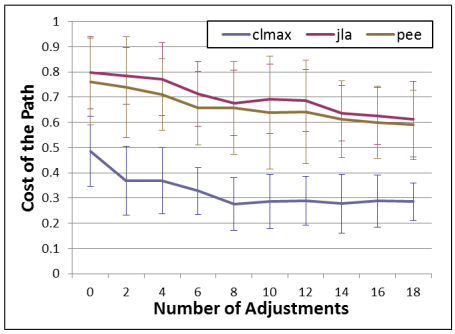
tion required by the optimization, and the amount of additional time depends on the computation of the soft constraints. In this example, computing $cost^v_{clmax}$ is expensive as it requires the distance to be computed between each robot link and the obstacles, and computations of $cost^v_{jla}$ and $cost^v_{pee}$ are relatively less expensive as they do not require examining geometry of the workspace. $cost^v_{pee}$ involves matrix



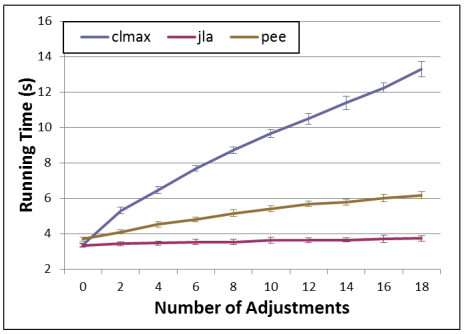
(a)



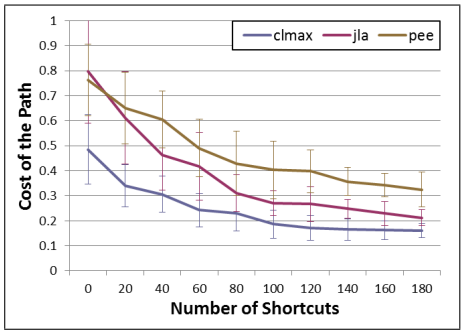
(b)



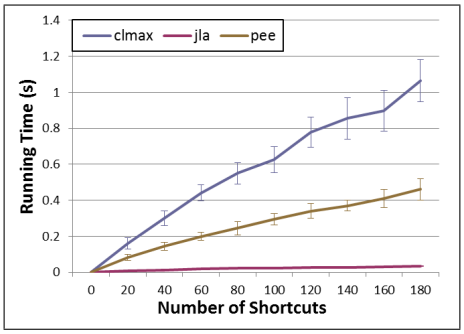
(c)



(d)



(e)



(f)

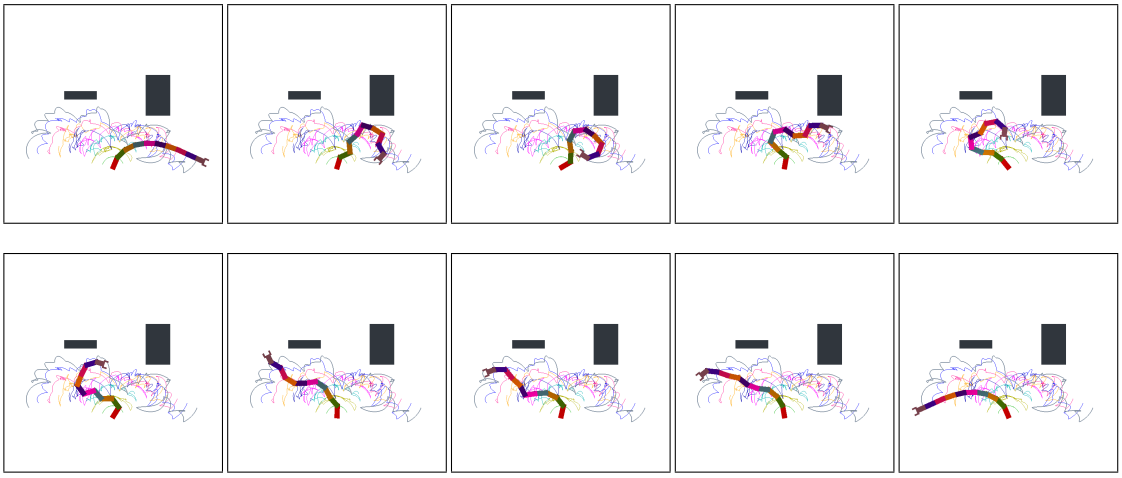
Figure 5.8: Soft cost of the paths found by and running time of (a-b) *SamplingSC*; (c-d) *SamplingHCSC*; and (e-f) *Shortcut with Soft Constraints*. Results are averaged for 20 independent runs for each case. Standard deviations are shown.

computation, so it is more expensive to compute than $cost^v_{jla}$.

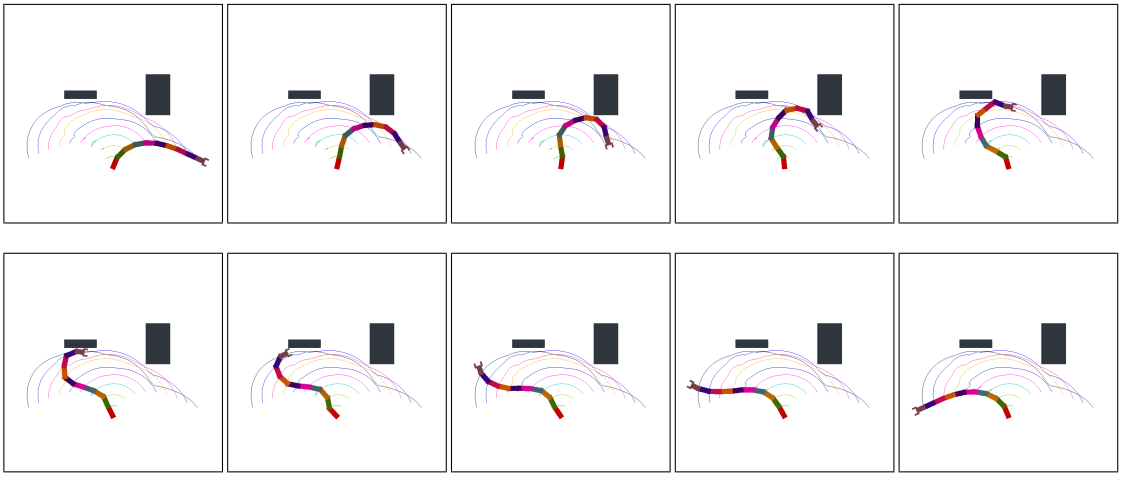
5.1.3 Optimizing multiple soft constraints

Here we test our auction-based optimization algorithm's performance using the full version of the algorithm that integrates multiple soft constraints. Assume the importance weights for the three involved soft constraints are equal. Figure 5.9(a) shows the paths computed by the traditional PRM. The original PRM path contains many extra motions (zig-zags and unnecessary rotations of the links of the robot) that result from the stochastic nature of the planner. Because path length is typically required to be short in practice we also show the smoothed path in Figure 5.9(b) computed by the traditional PRM followed by a traditional Shortcut method and compare it with our soft constraint path. The smoothed path is short but often causes the robot to approach very close to obstacles in the environment. Figure 5.10 shows the corresponding paths optimized by the auction-based approach. The robot tends to stay away from obstacles and bend its joints for maximum end-effector precision during navigation.

Figure 5.11 shows the combined and individual costs of the three considered soft constraints of the solution paths for the planning problem shown in Figure 5.9 and 5.10. The optimization strategy reduced the total soft constraint cost of the PRM path by about 45% and that of the smoothed path by about 30%. Further-



(a)



(b)

Figure 5.9: Traditional path planning results for the planar tentacle robot by (a) Traditional PRM; (b) Traditional PRM followed by a traditional Shortcut method.

more, it shows that our optimization method is able to reduce each soft constraint cost of the PRM path. Note that the soft constraint path has a slightly higher cost

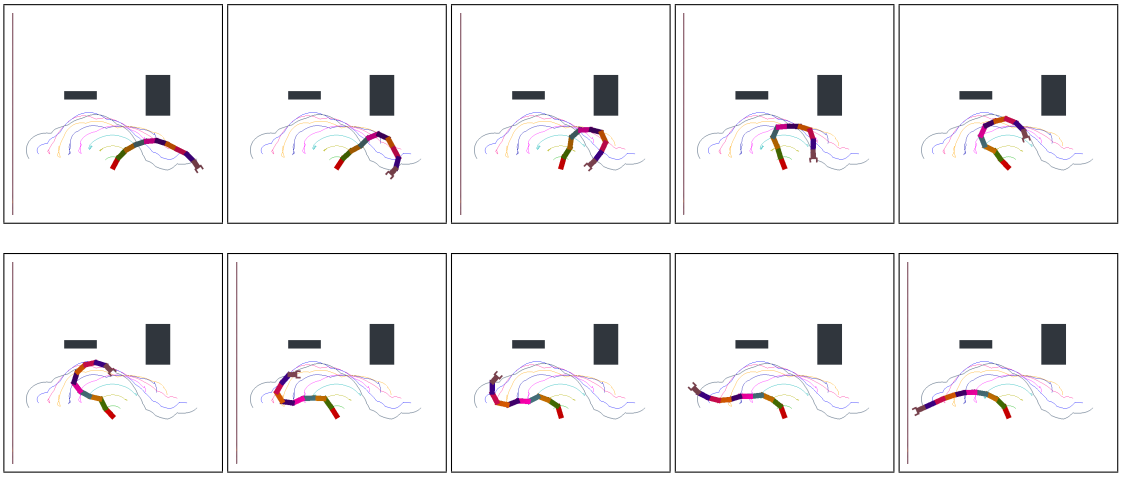


Figure 5.10: Auction-based path planning with soft constraints maximizing the clearance, avoiding joint limit and maximizing end-effector precision.

of $cost^v_{jla}$ than the smoothed path. This is due to the conflict between different soft constraints. Here, the reduction of $cost^v_{clmax}$ and $cost^v_{pee}$ results in an increase of $cost^v_{jla}$.

Figure 5.12 shows the performance of the auction-based approach combining the three participating optimizers. The total combined soft constraint costs of the paths versus the rounds of optimizations are shown. The best/worst case in each round is computed by trying all of the participating optimizers before choosing the one that improves the path cost the most/least. This increased the computational time to three times the original budget, but provides an idea of how well our auction-based approach performs relative to a system that is more exhaustive in

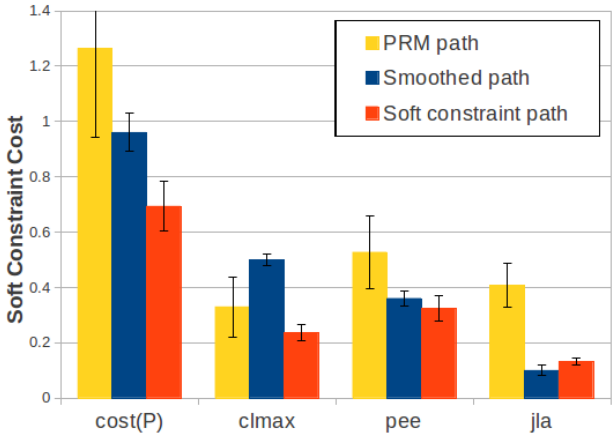


Figure 5.11: Soft constraint cost comparison between the PRM with uniform sampling, the PRM with uniform sampling followed by shortcut, and auction-based path optimization with soft constraints. The plot shows the combined and individual soft constraints costs with standard deviation. All values are taken over 20 independent runs.

terms of trying the various optimizations. These charts plot the average path cost with standard deviations in each round of the optimization. The results show that the auction-based approach performs significantly better than the worst case and quickly converges to the best case as the optimization progresses. It also means that the utility function defined in the auction-based approach approximates the right decisions well.

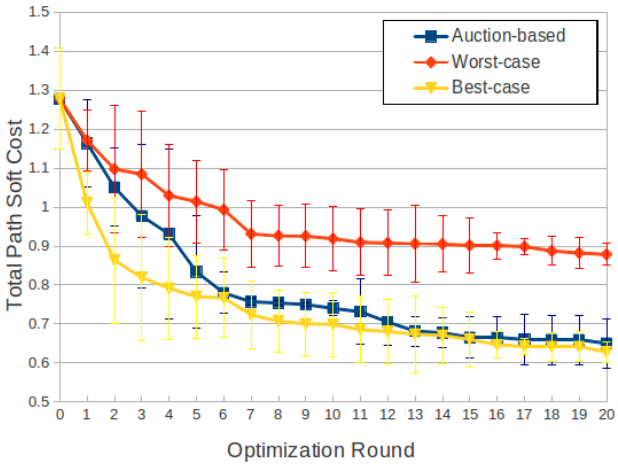


Figure 5.12: Performance of the auction-based path optimization. Upper and lower bounds are shown. The plot shows the average solution soft constraints costs with standard deviation.

5.2 Planning for a tentacle robot in a hot cell

The previous section examined the performance of the planning with soft constraints approach on a robot operation on the plane. Here we apply the algorithm to the simulation of a larger scale device that is designed to operate in radioactively hot regions in nuclear power plants.

5.2.1 Experimental setup

A kinematic model of the snake robot designed and built by OC Robotics, shown in Figure 5.2 was constructed (Figure 5.13). The robot is modeled as a collection of

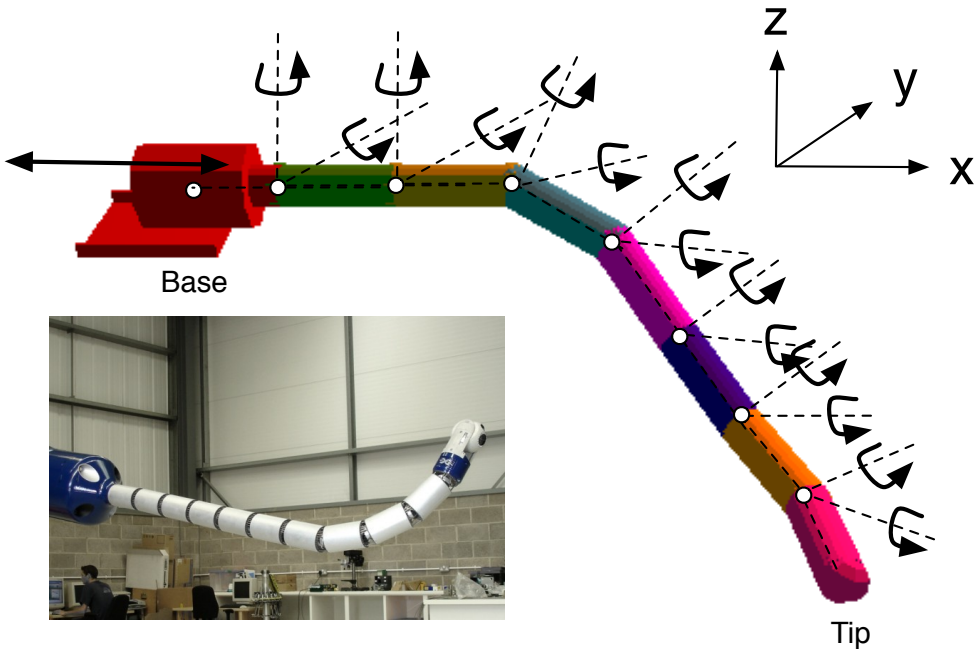


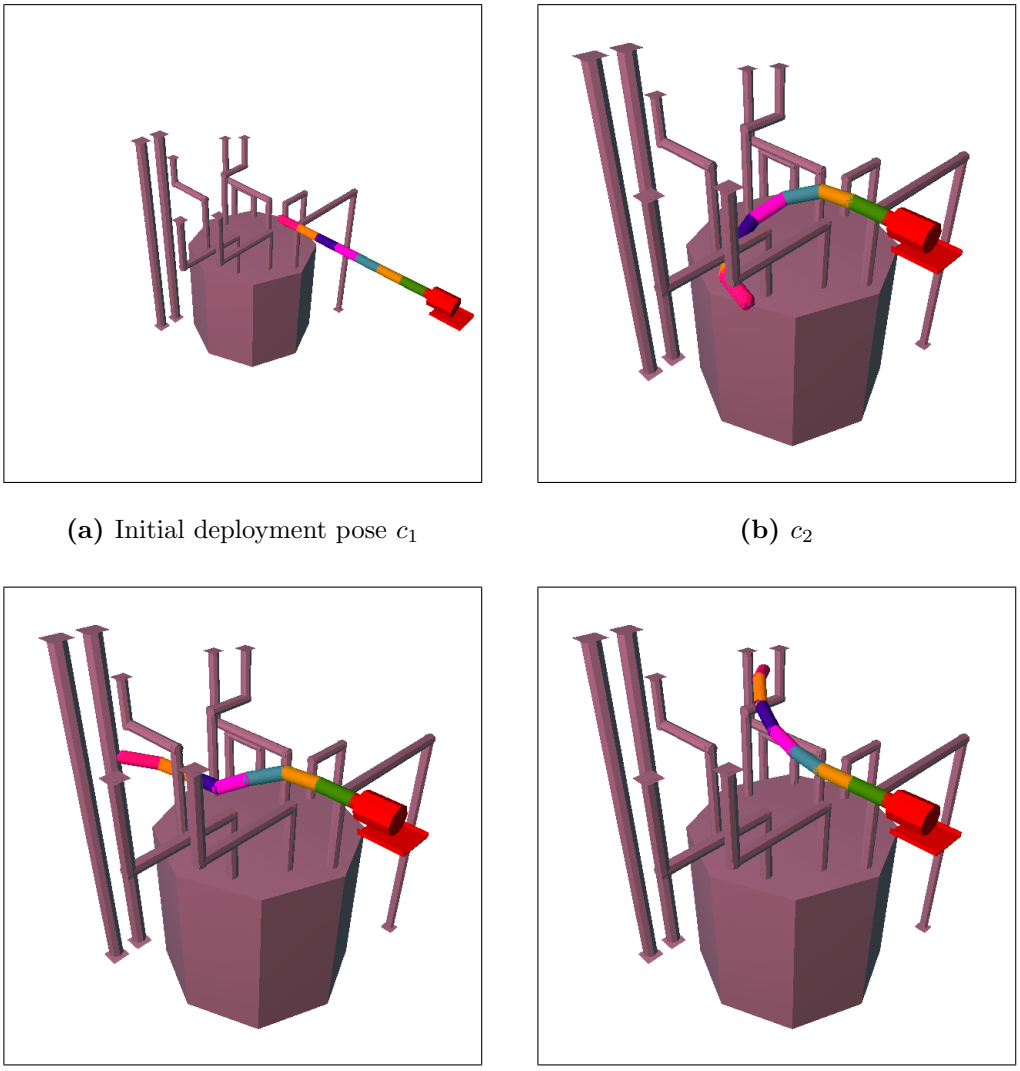
Figure 5.13: The tentacle robot consists of a mobile base that translates in one dimension and seven joints that each consists of two DOFs. The origin of the coordinate frame is defined as the location of the base when $x_{base} = 0$.

rigid links connected via simple 2-DOF joints, and the tentacle robot is modeled using traditional techniques from the robot manipulator literature [62, 69]. The robot itself is mounted on a mobile base that translates in one dimension. Formally, the robot can be described in terms of seven joints, each consisting of two DOFs - pitch and yaw - plus one translational joint for a total of 15 DOFs. The configuration of the robot is written as a vector $c = (x, \alpha_1, \beta_1, \alpha_2, \beta_2, \alpha_3, \beta_3, \alpha_4, \beta_4, \alpha_5, \beta_5, \alpha_6, \beta_6, \alpha_7, \beta_7)$, where x is the distance that the base of the robot moves along the x-axis. Each joint

has limits in terms of joint angles. Specifically, $\alpha_i \in [-60°, 60°], \beta_i \in [-60°, 60°]$ for $i = 1, ..., 7$ are the yaw and pitch angles of each link of the robot. The translational link is restricted to the range $x \in [0, 100]$.

Tentacle robots find application for in-situ inspections and repairs in nuclear power plants, as these sites are often inaccessible for humans and can contain extremely confined spaces. Here we model a nuclear decommissioning project completed by OC Robotics for Sellafiled Ltd, a British company responsible for safely delivering decommissioning, reprocessing, nuclear waste management and fuel manufacturing activities [3]. The site contains a wide range of ponds, vaults and hot cells, often containing unquantified radiation risks. Shown in Figure 5.2, the environment we test is a simulated hot cell mock-up, representing a dry processing cell containing pipework and vessels. It is a representative of confined-space challenges found in contaminated environments across the Sellafield site.

Suppose the robot makes its entry into the cell via a horizontal access hole. The initial configuration is shown in Figure 5.14(a). The robot must move a tool attached to its end effector to a range of different locations within the cell. To demonstrate our approach's ability to meet the challenges in confined space we consider three difficult spots among the pipes as our goals for the robot to reach as shown in Figure 5.14(b-d). In the following experiments we investigate the construction of practical paths from c_1 to c_2, c_2 to c_3, and c_3 to c_4.



(a) Initial deployment pose c_1 **(b)** c_2

(c) c_3 **(d)** c_4

Figure 5.14: Four goal configurations of the tentacle robot considered in the experiments reported here, labelled as c_1, c_2, c_3 and c_4.

5.2.2 Identifying soft constraints

There are a number of different properties the tentacle robot should exhibit to optimize its performance for this hot cell task, but safety is perhaps the most important issue. First, we choose two node-level soft constraints that are commonly associated with tentacle robots, including safe clearance avoidance $cost^v_{clmax}$ defined in Eq. 3.1 and joint limit avoidance $cost^v_{jla}$ defined in Eq. 3.3. These two types of soft constraints are defined in Chapter 3. Short paths are often desired in robot path planning, but this requires a proper definition of a distance metric. There are many ways to define a "short" path for tentacle robots, such as one with small swept volume by the links, and one with small rotations by the joints. In the nuclear task we consider here, the robot is typically required to carry a tool by its end-effector, so it is desired that the robot's end-effector moves along a short path. Therefore we define a global-path-level soft constraint cost for shortening the distance travelled by the end-effector $cost^p_{lenee}$.

Optimization of multiple soft constraints requires that each soft constraint be assigned a weight based upon its relative importance. Each soft constraint cost describes "how much" the soft constraint is violated, and its associated weight describes "how important" it is with respect to other soft constraints. The user can specify a higher weight to a soft constraint that he/she thinks is important. In

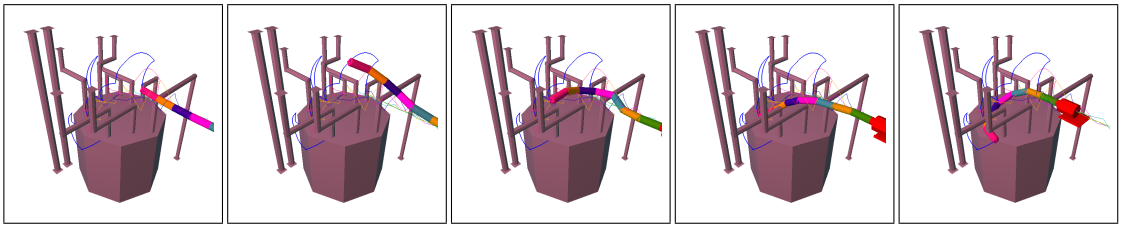
our example, as safety is the most important concern for nuclear tasks, we assign the weights as $w_{clmax} = 5, w_{jla} = 1, w_{lenee} = 1$, so that the planner first attempts to optimize clearance rather than optimizing distance travelled by the end-effector and joint limit avoidance.

5.2.3 Optimizers

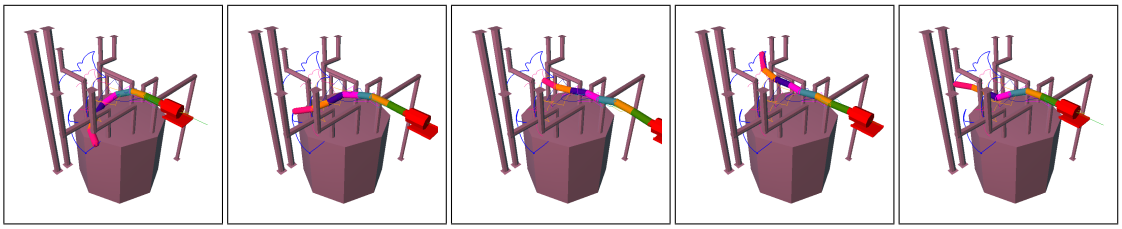
Both node-level and global-path-level soft constraints are involved in this experiment, so optimization strategies designed for these two types of soft constraints are suitable here. PRM*SC and PRM*HCSC optimizers are used for the node-level soft constraints, and a ShortcutSC optimizer is used for the global-path-level soft constraint. We set up the initial bidding price for the optimizers based on several test runs, i.e. given a constructed roadmap and a found path, how much improvement on average they can make within one unit of time. The initial bidding prices are: $p_{PRM*SC} = 0.08$, $p_{PRM*HCSC} = 0.09$, $p_{ShortcutSC} = 0.21$.

5.2.4 Results

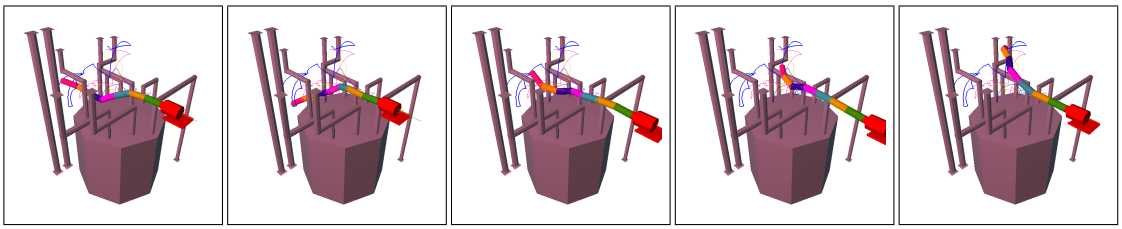
Figures 5.15 and 5.16 illustrate the practicality of the paths computed by the auction-based optimization. Figure 5.15 shows the paths computed by the traditional PRM (baseline case). The paths contain many extra motions (zig-zags and unnecessary rotations of the links of the robot) that result from the stochastic



$c_1 \rightarrow c_2$



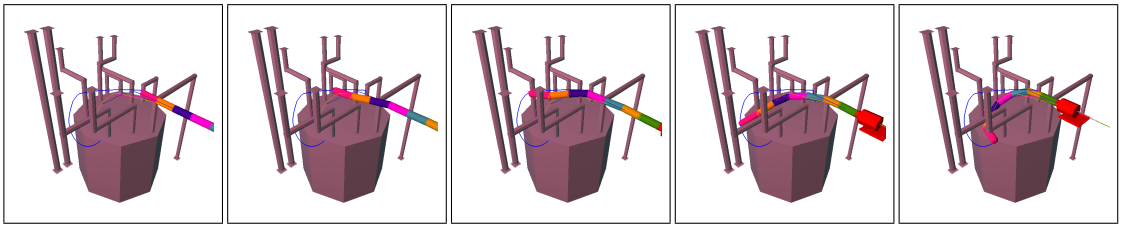
$c_2 \rightarrow c_3$



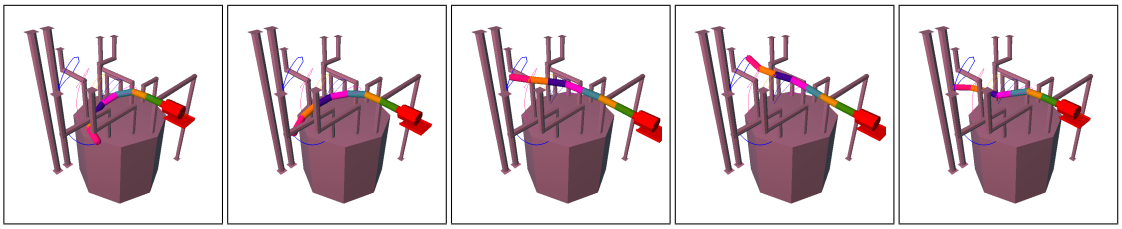
$c_3 \rightarrow c_4$

Figure 5.15: Traditional PRM path planning results for a tentacle robot in the hot cell.

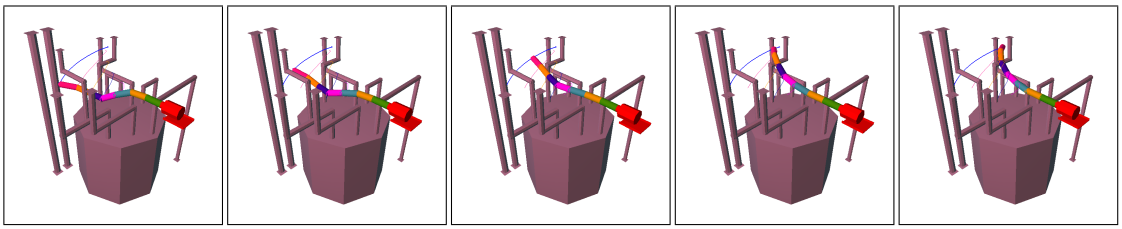
nature of the planner. In addition, the planned paths often cause the robot to approach very close to obstacles in the environment. Figure 5.16 shows the corresponding paths optimized by the auction-based approach. The paths tend to be short and smooth. Note that the robot tends to stay away from obstacles during



$c_1 \rightarrow c_2$



$c_2 \rightarrow c_3$

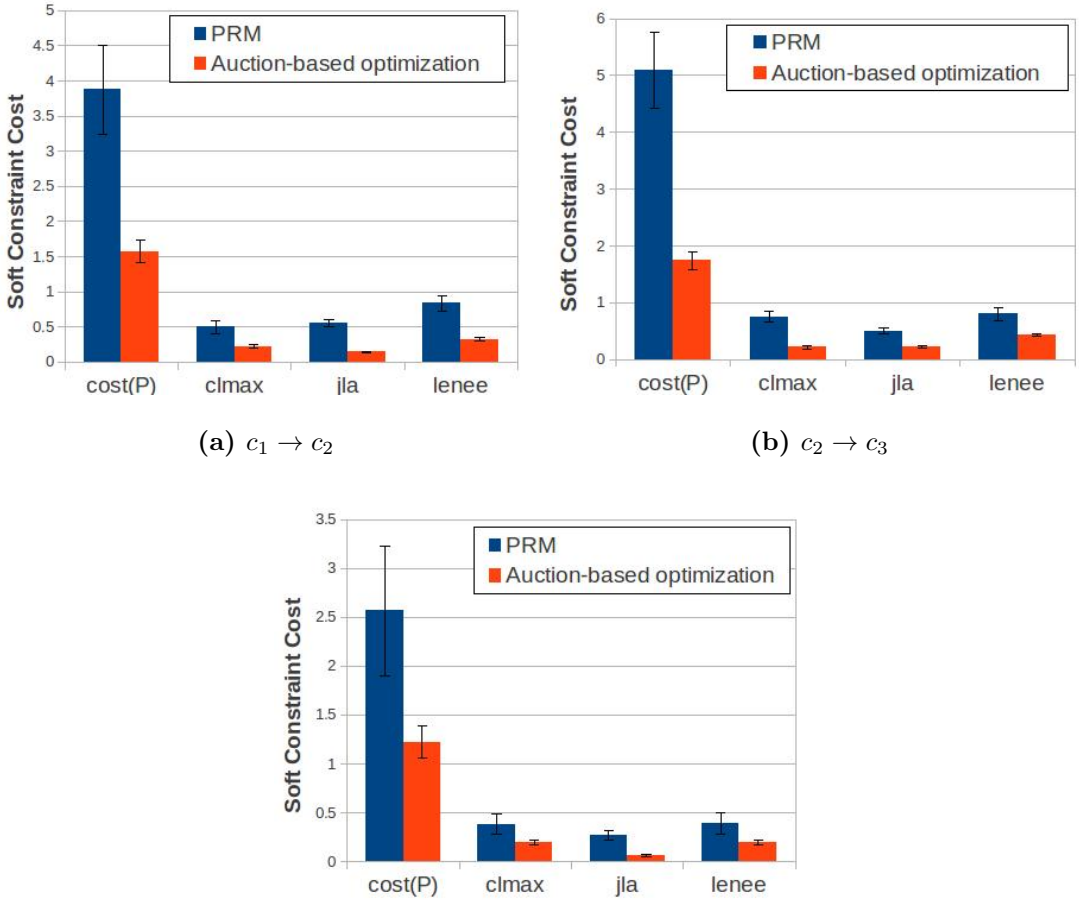


$c_3 \rightarrow c_4$

Figure 5.16: Auction-based path planning with soft constraints maximizing the clearance, avoiding joint limit and minimizing the end-effector's travelling distance.

navigation, but it has to violate the safe clearance soft constraint in order to reach the goals which may be close to the obstacles.

Figure 5.17 shows the soft constraint cost of the paths computed by the basic PRM algorithm and the auction-based optimization. Both algorithms were run



(a) $c_1 \rightarrow c_2$

(b) $c_2 \rightarrow c_3$

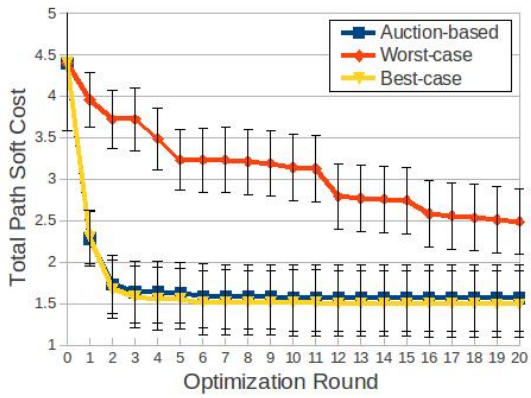
(c) $c_3 \rightarrow c_4$

Figure 5.17: Soft constraint cost comparison between the PRM with uniform sampling and auction-based path optimization with soft constraints for the poses shown in Figure 5.14. The plot shows the average solution soft constraints costs with standard deviations. All values are taken over 20 independent runs.

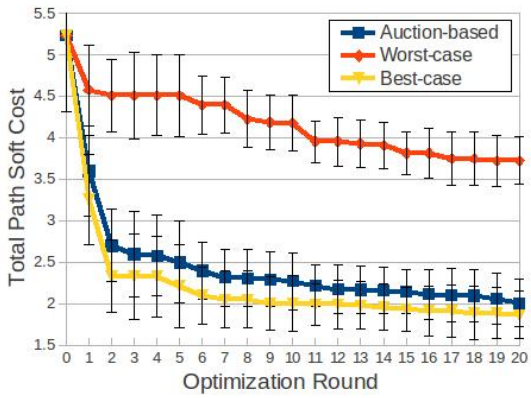
for 20 units of time. The optimization strategy reduced the cost of the total soft constraint cost by at least 50%. Furthermore, the optimization strategy reduced the cost of each individual soft constraint by at least 50%, although the improvement for each soft constraint varies in the different scenarios. For example, to move the robot from c_2 to c_3 (shown in Figure 5.17(b)) the optimization is able to discover a path with relatively larger clearance, resulting $cost^v_{clmax}$ to decrease by about 70%. To move the robot from c_1 to c_2 and from c_3 to c_4 (shown in Figure 5.17(a) and (c)), the optimization is able to reduce $cost^v_{jla}$ by about 75%.

Figure 5.18 compares the performance of the auction-based approach with the best case and the worst case in each round of the optimization. The total combined soft constraint costs of the paths are compared. The best/worst case in each round is computed by trying all of the participating optimizers before choosing the one that improves the path cost the most/least. This increased the computational time to three times the original budget, but provides an idea of how well our auction-based approach performs relative to a system that is more exhaustive in terms of trying the various optimizations. These charts plot the average path cost with standard deviations in each round of the optimization. The results show that the auction-based approach performs significantly better than the worst case and quickly converges to the best case as the optimization progresses.

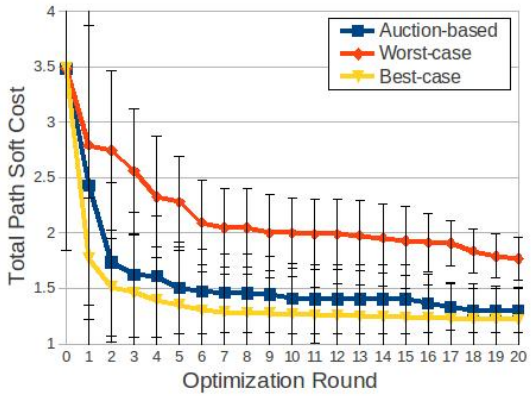
Although the auction-based optimization aims at reducing the combined soft



(a) $c_1 \rightarrow c_2$



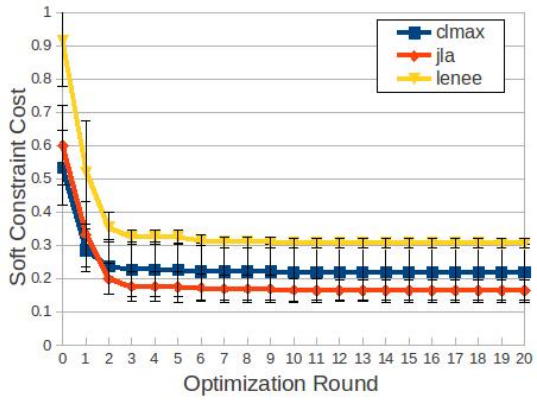
(b) $c_2 \rightarrow c_3$



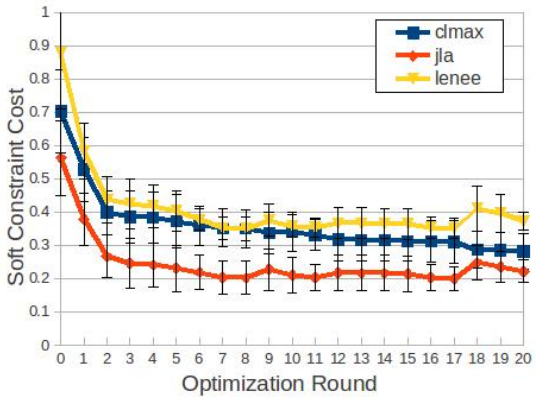
(c) $c_3 \rightarrow c_4$

Figure 5.18: Performance of the auction-based path optimization. Upper and lower bounds are shown. The plot shows the average solution soft constraints costs with standard deviation.

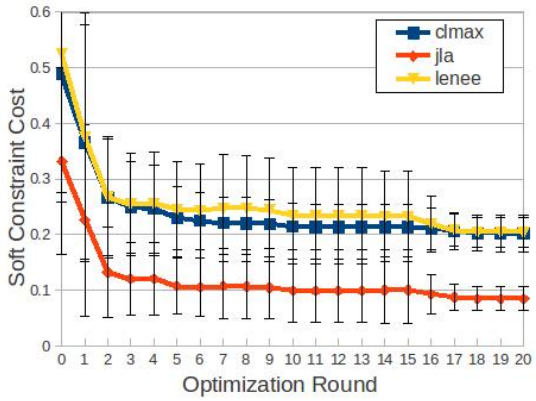
costs of the solution, it is interesting to investigate its effects on each individual soft constraint. Figure 5.19 plots the solution cost of each soft constraint achieved



(a) $c_1 \rightarrow c_2$



(b) $c_2 \rightarrow c_3$



(c) $c_3 \rightarrow c_4$

Figure 5.19: Effects of the auction-based path optimization with respect to computational budget. The plot shows the individual soft constraints costs averages with standard deviation. All values are taken over 20 independent runs.

by the auction-based optimization given a specific planing budget. Note that it is not the case that each soft constraint cost is monotonically decreasing. This is due

to the conflict between different soft constraints. For example, in Figure 5.19(b) at some time during the optimization $cost_{jla}$ and $cost_{lenee}$ increased while $cost_{clmax}$ decreased. This is a consequence of the safe clearance soft constraint having a higher importance weight than joint limit avoidance and length of the end-effector, i.e. $w_{clmax} > w_{jla}$ and $w_{clmax} > w_{lenee}$.

5.3 Summary

The need to properly represent and use soft constraints is particularly important for redundant DOF robots such as tentacle devices. In this chapter our path optimization approach introduced in Chapter 3 and 4 is grounded in the capabilities and tasks associated with tentacle robots. Experiments with both a 2D tentacle robot in the lab and a 3D tentacle robot in a nuclear hot cell environment were presented. The soft-constraint-based path planner has shown effective improvement over the path computed by the basic PRM and PRM* within the same computational budget.

6 Discussions and future work

Path planning is an important but difficult problem in robot planning with high numbers of DOFs. Sampling-based path planning algorithms are successful in solving high-dimensional problems. However, their ability to find paths that meet certain soft constraints is still limited. This thesis presents a framework within which plans can be developed for high DOF robotic systems within a time budget that meet the hard constraints of path feasibility and at the same time seek to reduce the cost of a collection of domain-specific soft constraints. Soft constraints are formalized within the PDDL3.0 and divided into three categories: node-level, edge-level, and global-path-level soft constraints. Each of the categories have a corresponding optimization within the PRM algorithm. Soft constraints defined at the vertex level can be optimized during the node generation phase. Soft constraints defined at the edge level can be optimized during the edge linking phase, while optimizations defined at the path level can be optimized at the post processing phase. Optimization strategies in each of these phases are well understood.

An important contribution in this work is the development of a multiple-round auction-based approach that combines and coordinates multiple optimizations so that their strengths are preserved and computational resource is allocated dynamically among them. The optimization task is decomposed into rounds of sequential computations, during each of which one optimizer is chosen to run for a unit of time. Each optimizer adjusts its expected utility downwards based on the utility observed thus far across all the optimizers and the optimizer with the best predicted performance is dynamically selected as the process goes on.

This framework is intended to be robot independent, but in this thesis the approach is grounded in the capabilities and tasks associated with tentacle robots. We have shown the effectiveness of our approach using both simulated and real tentacle robots. Several soft constraints are used separately and together in the test model. Although the resultant path is not necessarily optimal due to the stochastic nature of the planner, the approach presented here shows significant improvement over the path computed by the basic PRM and PRM* within the same time budget.

There are a number of important issues that have not been addressed in this thesis. First, in choosing *a priori* a set of soft constraints to be optimized, we have constrained the system to consider only soft constraints specified by the user. This assumption requires the user to know the robot and the environment well enough to

define proper soft constraint cost functions and the associated importance weights. Besides, some real soft constraints do not fall into any of the three categories defined in the thesis, so it is unknown when and where to optimize them in the sampling-based path planning.

The developed optimization strategies also have limitations. Their performance depends upon the original sampling-based path planners and the soft constraints that need to be optimized. While the auction-based system provides a dynamic way to coordinate multiple optimizers, there is no guarantee that the optimization can find the optimal solution.

6.1 Limitations and directions for future work

Our current formalization of soft constraints is based on PDDL3, which assumes a known cost function in [0,1] and a known importance weight for each soft constraint. The cost of the path is obtained by summing together the weighted costs of all the soft constraints. This allows multiple soft constraints to be considered simultaneously in path planning but it also limits the generality of the approach. This assumption has limited the robustness of the algorithm. In practice, weighting of individual terms is not a trivial task, since different soft constraints usually have different units and scaling. An extension to our assumption of a pre-defined constant weighting method is the automatic generation of importance weights after

the user specifies their preference ordering of the soft constraints.

This work categorizes soft constraints into node-level, edge-level and global-path-level. It is of great practical interest to address the path practicality problems subject to more complex constraints that do not fall into these three categories. For example, in practice it may not be desirable for a robot to make significant changes in trajectory as it passes through a node from one edge to another. Therefore it is necessary to capture a cost of the transition between two edges that share a node. In addition, it is of interest to consider practical planning problems in the presence of temporal/logic constraints on the paths, expressed using formal specification language such as Linear Temporal Logic (see [10] for more details on such constraints). For example the robot may be required to visit region X before reaching the goal or to visit $X1$, then $X2$, then the goal.

While integrating soft constraints in the path planning this thesis focuses mainly on roadmap-based probabilistic path planners, i.e. PRMs. However, it would also be useful to explore ways of integrating soft constraints in the tree-based path planners, such as RRTs and ESTs. These path planners are designed to solve a single query as fast as possible and are known for dealing with non-holonomic constraints that exist in many mobile devices. For example, it is desirable that the planner does not waste time for exploring regions of the configuration space that has high soft constraint cost and grows the tree towards a potential final path

with low soft constraint cost. Various categories of soft constraints are likely to be applied in the tree expansion, merging and post-processing stages.

This work developed an auction-based system that combines multiple optimization strategies to deal with multiple soft constraints within a computational budget. The simplest kind of auction, a single-item first-price auction, is adopted to determine the winner optimizer to execute in each round. More complex auction mechanisms may also be used. For example, using combinatorial auctions [29] multiple rounds of computations are offered and each optimizer can bid on any combination of subsets of these rounds. This allows the optimizer to explicitly express the synergies between rounds of computations. Consider the shortcut method as a participating bidder, which often stops making improvement of a given solution within one round, it can express the negative synergy between two rounds that are close together by bidding only slightly higher for the bundle of these two rounds than for either round individually.

Currently the auction-based selection of optimizers is compared to the best and worst cases. Future work would include comparisons with other selection approaches. For example, rather than to select the highest bidder in each round it may select a bidder with some probability, such that highest bidders are often but not always selected. Other probabilistic selection mechanisms are also possible, and this would certainly be an interesting area for further research.

Future work should provide more quantitative analysis of the approach. For example, the developed optimization strategies could be compared to other optimization techniques rather than just the basic path planners. Although most existing optimization techniques are designed for specific robotic systems and soft constraints and our approach is more general, such numerical comparison could provide insight to the advantages and limitations of our approach.

This thesis presented experiments of the algorithm on two types of tentacle robots in a simple planar environment and a simulated 3D hotcell environment. In particular the method was applied on a real planar tentacle robot. Executions of the paths computed by our optimization technique on the robot were demonstrated. It would be desirable to provide a more systematic and quantitative analysis of the results in the future. Such analysis can be accomplished in many ways. For example, the accuracy with which the real robot follows the desired path and reaches the goal could be measured. Further experiments on real tentacle robot operating in confined space and requiring high path practicality would also be useful. Applications such as nuclear reactor inspection and heart surgery might provide suitable environments to further evaluate the algorithms and approaches presented here.

Finally, making path planning algorithms capable of planning practical movements for robot in dynamic real-world environments will significantly advance the state of art in robotics. Although this work assumes a static and known environ-

ment it has presented a general framework for path planning. Due to its general nature, it can be adapted further to solve the dynamic environment problems.

Bibliography

[1] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo. OBPRM: an obstacle-based PRM for 3D workspaces. In *Proceedings of the International Workshop on Algorithmic Foundations of Robotics (WAFR)*, pages 155–168, Natick, MA, USA, 1998.

[2] Anonymous. MDA performs critical inspection of Pickering nuclear reactor. urlhttp://sm.mdacorporation.com/news/pr23082010.html, August 2010.

[3] Anonymous. Nuclear decommissioning case study: Sellafield. `http://www.ocrobotics.com/applications--solutions/nuclear/nuclear-case-study--sellafield/`, 2014.

[4] R. Anscombe, R. Buckinham, A. Graham, N. Parry, and M. Lichon. Snake-arm robots conduct nuclear maintenance. In *Proceedings of the International Youth Nuclear Congress (IYNC)*, Stockholm - Olkiluoto, 2006.

[5] J. A. Baier, F. Bacchus, and S. A. McIlraith. A heuristic search approach to planning with temporally extended preferences. *Artificial Intelligence*, 173(5-6):593–618, 2009.

[6] J. Barraquand and J. C. Latombe. Robot motion planning: A distributed representation approach. *International Journal of Robotics Research*, 10(6):628–649, 1991.

[7] O. B. Bayazit. *Solving Motion Planning Problems by Iterative Relaxation of Constraints*. PhD thesis, Texas A&M University, 2003.

[8] K. E. Bekris and L. E. Kavraki. Greedy but safe replanning under kinodynamic constraints. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, pages 704–710, Roma, Italy, April 2007.

[9] P. Bessière, J. manuel Ahuactzin, E. ghazali Talbi, and E. Mazer. The Ariadne's clew algorithm: Global planning with local methods. In *Proceedings of*

the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), volume 2, pages 1373–1380, Tokyo, Japan, 1993.

[10] A. Bhatia, L. E. Kavraki, and M. Y. Vardi. Sampling-based motion planning with temporal goals. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, Anchorage, Alaska, USA, May 2010.

[11] P. Bhattacharya and M. L. Gavrilova. Geometric algorithms for clearance based optimal path computation. In *Proceedings of the ACM International Symposium on Advances in Geographic Information Systems (GIS)*, pages 1–4, New York, NY, USA, 2007.

[12] S. Bistarelli. *Semirings for Soft Constraint Solving and Programming.* Springer-Verlag New York, Inc., 2004.

[13] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint satisfaction and optimization. *Journal of the ACM*, 44(2):201–236, 1997.

[14] R. Bohlin and L. E. Kavraki. Path planning using Lazy PRM. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, volume 1, pages 521–528, San Fransisco, CA, USA, April 2000.

[15] R. Brafman and Y. Chernyavsky. Planning with goal preferences and constraints. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 182–191, Monterey, CA, 2005.

[16] J. Bruce and M. Veloso. Real-Time Multi-Robot Motion Planning with Safe Dynamics. In *Multi-Robot Systems: From Swarms to Intelligent Automata*, volume 3, pages 159–170, 2005.

[17] R. Buckinham and A. Graham. SAFIRE - a robotic inspection system for CANDU feeders. In *Proceedings of the International Conference on CANDU Maintenance*, 2011.

[18] B. Burns. *Exploiting Structure: A Guided Approach to Sampling-Based Robot Motion Planning.* PhD thesis, The University of Massachusetts, 2007.

[19] B. Burns and O. Brock. Sampling-based motion planning using predictive models. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, pages 3120–3125, Barcelona, Spain, April 2005.

[20] J. F. Canny. *The Complexity of Robot Motion Planning.* MIT Press, Cambridge, MA, USA, 1988.

[21] R. Cassady. *Auctions and Auctioneering*. California University Press, Berkeley, 1967.

[22] A. R. Cassandra, L. P. Kaelbling, and M. L. Littman. Acting optimally in partially observable stochastic domains. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI)*, volume 2, pages 1023–1028, Menlo Park, CA, USA, 1994.

[23] R. Cavallo. *Social Welfare Maximization in Dynamic Strategic Decision Problems*. PhD thesis, Harvard University, 2008.

[24] P. Cheng, G. Pappas, and V. Kumar. Decidability of motion planning with differential constraints. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, pages 1826–1831, Roma, Italy, 2007.

[25] H. Choset, K. M. Lynch, S. Hutchinson, W. B. G. Kantor, L. E. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, 2005.

[26] H. Choset, S. Walker, K. Eiamsa-ard, and J. Burdick. Sensor-based exploration: Incremental construction of the hierarchical generalized Voronoi graph. *International Journal of Robotics Research*, 19(2):126–148, 2000.

[27] K. Cleary. Incorporating multiple criteria in the operation of redundant manipulators. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, pages 618–624, Cincinnati, OH, USA, 1990.

[28] D. A. Cohn, Z. Ghahramani, and M. I. Jordan. Active learning with statistical methods. *Journal of Artificial Intelligence Research*, 4:129–145, 1996.

[29] M. B. Dias, R. Zlot, N. Kalra, and A. Stentz. Market-based multirobot coordination: A survey and analysis. In *Proceedings of the IEEE*, volume 94, pages 1257–1270, 2006.

[30] C. Domshlak, S. Prestwich, F. Rossi, K. B. Venable, and T. Walsh. Hard and soft constraints for reasoning about qualitative conditional preferences. *Journal of Heuristics*, 12(4-5):263–285, 2006.

[31] A. Doucet, N. Freitas, and N. Gordon. *Sequential Monte Carlo Methods in Practice*. Springer-Verlag, 2001.

[32] R. V. Dubey, S. McGhee, and T. F. Chan. Probability-based weighting of performance criteria for redundant manipulators. *Journal of Intelligent and Robotic Systems*, 19:89–103, 1997.

[33] S. Edelkamp and J. Hoffmann. PDDL2.2: The language for the classical part of the 4th international planning competition. Technical Report 195, University of Freiburg, 2004.

[34] D. Ferguson, N. Kalra, and A. Stentz. Replanning with RRTs. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, Florida, Orlando, USA, 2006.

[35] J. E. Gallardo, C. Cotta, and A. J. Fernandez. Solving weighted constraint satisfaction problems with memetic/exact hybrid algorithms. *Journal of Artificial Intelligence Research*, 35:533–555, 2009.

[36] M. Garber and M. C. Lin. Constraint-based motion planning using Voronoi diagrams. In *Proceedings of the International Workshop on Algorithmic Foundations of Robotics (WAFR)*, Nice, France, 2002.

[37] R. Geraerts and M. Overmars. On improving the clearance for robots in high-dimensional configuration spaces. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 679–684, Edmonton, Canada, August 2005.

[38] R. Geraerts. *Sampling-based Motion Planning: Analysis and Path Quality*. PhD thesis, Utrecht University, 2006.

[39] R. Geraerts and M. H. Overmars. Creating high-quality paths for motion planning. *International Journal of Robotics Research*, 26(8):845–863, 2007.

[40] A. Gerevini and D. Long. Plan constraints and preferences in PDDL3 - the language of the 5th International Planning Competition. Technical report, University of Brescia, 2005.

[41] S. Ghosh and D. M. Mount. An output sensitive algorithm for computing visibility graphs. *SIAM Journal on Computing*, 20:888–910, 1991.

[42] E. Giunchiglia and M. Maratea. Planning as satisfiability with preferences. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI)*, pages 987–992, Vancouver, British Columbia, 2007.

[43] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic detrmination of minimum cost paths. *IEEE Transactions of Systems Science and Cybernetics*, 4(2):100–107, 1968.

[44] B. Hill and D. Tesar. *Design of Mechanical Properties for Serial Manipulators*. PhD thesis, University of Texas at Austin, 1997.

[45] K. E. Hoff III, J. Keyser, M. Lin, D. Manocha, and T. Culver. Fast computation of generalized Voronoi diagrams using graphics hardware. *Computer Graphics*, 33(Annual Conference Series):277–286, 1999.

[46] J. E. Hopcroft, J. T. Schwartz, and M. Sharir. On the complexity of motion planning for multiple independent objects: PSPACE-hardness of the "warehouseman's problem". *International Journal of Robotics Research*, 3(4):76–88, 1984.

[47] J. E. Hopcroft and G. T. Wolfong. Reducing multiple object motion planning to graph searching. *SIAM Journal on Computing*, 15(3):768–785, 1986.

[48] D. Hsu. *Randomized Single-query Motion Planning in Expansive Spaces*. PhD thesis, Stanford University, May 2000.

[49] D. Hsu, L. E. Kavraki, J.-C. Latombe, R. Motwani, and S. Sorkin. On finding narrow passages with probabilistic roadmap planners. In *Proceedings of the International Workshop on Algorithmic Foundations of Robotics (WAFR)*, pages 141–153, Natick, MA, USA, 1998.

[50] D. Hsu, G. Sanchez-Ante, and Z. Sun. Hybrid PRM sampling with a cost-sensitive adaptive strategy. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, pages 3874–3880, Barcelona, Spain, April 2005.

[51] Y. Huang and K. Gupta. Collision-probability constrained PRM for a manipulator with base pose uncertainty. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1426–1432, St. Louis, Missouri, USA, 2009.

[52] L. Hurwicz. The design of mechanisms for resource allocation. *The American Economic Review*, 63(2):1–30, 1973.

[53] P. Isto. Constructing probabilistic roadmaps with powerful local planning and path optimization. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 2323–2328, Lausanne, Switzerland, 2002.

[54] L. Jaillet, J. Cortes, and T. Simeon. Sampling-based path planning on configuration-space costmaps. *IEEE Transactions on Robotics*, 26(4):635–646, 2010.

[55] L. Jaillet and T. Simeon. A PRM-based motion planner for dynamically changing environments. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 2, pages 1606–1611, Sendai, Japan, 2004.

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

[57] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME – Journal of Basic Engineering*, (82):35–45, 1960.

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

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

[60] L. E. Kavraki, J.-C. Latombe, R. Motwani, and P. Raghavan. Randomized query processing in robot path planning. *Journal of Computer and System Sciences*, 57(1):50–60, 1998.

[61] E. C. Kerrigan and J. M. Maciejowski. Soft constraints and exact penalty functions in model predictive control. In *Proceedings of the UKACC International Conference (Control 2000)*, 2000.

[62] W. Khalil and E. Dombre. *Modeling, Identification and Control of Robots*. Hermes Penton Ltd., 2002.

[63] J. Kim, R. A. Pearce, and N. M. Amato. Extracting optimal paths from roadmaps for motion planning. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, volume 2, pages 2424–2429, Taipei, Taiwan, September 2003.

[64] J. T. Klosowski, M. Held, J. S. B. Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k-DOPs. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):21–36, March 1998.

[65] M. Kobilarov and G. S. Sukhatme. Near time-optimal constrained trajectory planning on outdoor terrain. In *Proceedings of the IEEE International*

Conference on Robotics & Automation (ICRA), pages 1833–1840, Barcelona, Spain, April 2005.

[66] I. Kolmanovsky and D. Filev. Stochastic optimal control of systems with soft constraints and opportunities for automotive applications. In *Proceedings of the IEEE International Conference on Control Applications*, St.Petersburg, Russia, 2009.

[67] H. Kurniawati. *Workspace-based Sampling for Probabilistic Path Planning*. PhD thesis, National University of Singapore, 2008.

[68] H. Kurniawati, D. Hsu, and W. S. Lee. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Proceedings of Robotics: Science & Systems IV*, Zurich, Switzerland, 2008.

[69] J.-C. Latombe. *Robot Motion Planning*. Kluwer, 1991.

[70] J.-C. Latombe. Motion planning: a journey of robots, molecules, digital actors, and other artifacts. *International Journal of Robotics Research*, 18(11):1119–1128, 1999.

[71] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu.

[72] S. M. LaValle and J. J. Kuffner. Rapidly-exploring random trees: Progress and prospects. In *Proceedings of the International Workshop on Algorithmic Foundations of Robotics (WAFR)*, Hanover, NH, USA, 2000.

[73] D. Lee. *Proximity and reachability in the plane*. PhD thesis, University of Illionis, Urbana, 1978.

[74] P. Leven and S. Hutchinson. Toward real-time path planning in changing environments. In *Proceedings of the International Workshop on Algorithmic Foundations of Robotics (WAFR)*, Hanover, NH, USA, 2000.

[75] T.-C. Liang, J.-S. Liu, G.-T. Hung, and Y.-Z. Chang. Practical and flexible path planning for car-like mobile robot using maximal-curvature cubic spiral. *Robotics and Autonomous Systems*, 52:312–335, 2005.

[76] J.-M. Lien, S. Thomas, and N. Amato. A general framework for sampling on the medial axis of the free space. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, volume 3, pages 4439–4444, Taipei, Taiwan, 2003.

[77] M. Likhachev. *Search-based Planning for Large Dynamic Environments*. PhD thesis, Carnegie Mellon University, 2005.

[78] T. Lozano-Pérez and M. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM (CACM)*, 22(10):560–570, 1979.

[79] V. Lumelsky and A. Stepanov. Path planning strategies for point mobile automation moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, 2:403–430, 1987.

[80] R. Luna, I. A. Sucan, M. Moll, and L. E. Kavraki. Anytime solution optimization for sampling-based motion planning. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, pages 5053–5059, Karlsruhe, Germany, May 2013.

[81] R. Manseur. *Robot Modeling and Kinematics*. Da Vinci Engineering Press, 2006.

[82] E. Masehianand, M. Admin-Naseri, and S. Khadem. Online motion planning using incremental construction of medial axis. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, pages 2928–2933, Taipei, Taiwan, 2003.

[83] R. McAfee and J. McMillan. Auctions and bidding. *Journal of Economic Literature*, (25):699–738, 1987.

[84] D. V. McDermott. PDDL – the planning domain definition language. Technical Report TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.

[85] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[86] M. Morales. *Metrics for Sampling-based Motion Planning*. PhD thesis, Texas A&M University, 2007.

[87] W. Moss, M. C. Lin, and D. Manocha. Constraint-based motion synthesis for deformable models. In *Proceedings of the Conference on Computer Animation and Virtual Worlds (CASA)*, volume 19, pages 421–431, 2008.

[88] C. Nielsen and L. E. Kavraki. A two-level fuzzy PRM for manipulation planning. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 1716–1722, Takamatsu, Japan, November 2000.

[89] D. Nieuwenhuisen and M. H. Overmars. Useful cycles in probabilistic roadmap graphs. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, volume 1, pages 446–452, New Orleans, LA, USA, April-May 2004.

[90] C. Nissoux, T. Simeon, and J.-P. Laumond. Visibility based probabilistic roadmaps. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 1316–1321, Kyongju, Korea, 1999.

[91] C. O'Dunlaing and C. Yap. A retracting method for planning the motion of a disc. *Journal of Algorithms*, 6:104–111, 1982.

[92] J. M. O'Kane. *A Theory for Comparing Robot Systems*. PhD thesis, University of Illinois, 2007.

[93] J. Peng and S. Akella. Coordinating multiple robots with kinodynamic constraints along specified paths. *International Journal of Robotics Research*, 24:295–310, 2005.

[94] J. M. Phillips, N. Bedrossian, and L. E. Kavraki. Guided expansive spaces trees: A search strategy for motion- and cost-constrained state spaces. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, pages 3968–3973, New Orleans, LA, USA, 2004.

[95] C. Pholsiri. *Task-based Decision Making and Control of Robotic Manipulators*. PhD thesis, University of Texas at Austin, 2004.

[96] J. M. Porta, N. Vlassis, M. T. Spaan, and P. Poupart. Point-based value iteration for continuous POMDPs. *Journal of Machine Learning Research*, 7:2329–2367, 2006.

[97] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa. CHOMP: Gradient optimization techniques for efficient motion planning. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, Kobe, Japan, May 2009.

[98] B. Raveh, A. Enosh, and D. Halperin. A little more, a lot better: Improving path quality by a path merging algorithm. *IEEE Transactions on Robotics*, 27(2):365–371, 2011.

[99] J. Reif and M. Sharir. Motion planning in the presence of moving obstacles. *Journal of the ACM*, 41(4):764–790, 1994.

[100] J. H. Reif. Complexity of the mover's problem and generalizations. In *Proceedings of the Annual Symposium on Foundations of Computer Science (SFCS)*, pages 421–427, 1979.

[101] G. Sáchez and J. C. Latombe. Using a PRM planner to compare centralized and decoupled planning for multi-robot systems. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, pages 2112–2119, Washington, DC, USA, May 2002.

[102] M. Saha. *Motion Planning with Probabilistic Roadmaps*. PhD thesis, Standford University, 2006.

[103] J. Schwartz and M. Sharir. On the Piano Movers' Problem: II. General techniques for computing topological properties of algebraic manifolds. *Advances in Applied Mathematics*, 12:298–351, 1983.

[104] Z. Shiler and S. Dubowsky. On computing the global time-optimal motions of robotic manipulators in the presence of obstacles. *IEEE Transactions on Robotics and Automation*, 7:785–797, 1991.

[105] D. E. Smith. Choosing objectives in over-subscription planning. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 393–401, Whistler, Canada, 2004.

[106] T. C. Son and E. Pontelli. Planning with preferences using logic programming. In *Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, LNCS, pages 247–260, 2004.

[107] G. Song, S. Miller, and N. M. Amato. Customizing PRM roadmaps at query time. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, pages 1500–1505, Seoul, Korea, 2001.

[108] A. Stentz. Optimal and efficient path planning for partially-known environments. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, volume 4, pages 3310–3317, San Diego, CA, USA, May 1994.

[109] A. Stentz. The focused D* algorithm for real-time replanning. In *Proceedings of the International Joint Conference of Artificial Intelligence*, August 1995.

[110] P. Svestka. A probabilistic approach to motion planning for car-like robots. Technical Report RUU-CS-93-18, Department of Computer Science, Utrecht University, Utrecht, The Netherlands, April 1993.

[111] S. Thomas, M. Morales, X. Tang, and N. M. Amato. Biasing samplers to improve motion planning performance. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, pages 1625–2630, Roma, Italy, 2007.

[112] C. Urmson and R. Simmons. Approaches for heuristically biasing RRT growth. In *Proceedings of the IEEE International Conference of Intelligent Robot and Systems (IROS)*, volume 2, pages 1178–1183, Las Vegas, Nevada, USA, 2003.

[113] J. van den Berg, P. Abbeel, and K. Goldberg. Lqg-mp: Optimized path planning for robots with motion uncertainty and imperfect state information. *International Journal of Robotics Research*, 30(7):895–913, 2011.

[114] G. van den Bergen. Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools (JGT)*, 2(4):1–14, 1997.

[115] M. van den Briel, S. Kambhampati, and T. Vossen. Planning with preferences and trajectory constraints through integer programming. In *Proceedings of the ICAPS Workshop on Planning with Preferences and Soft Constraints*, pages 19–22, Ambleside, U.K., 2006.

[116] M. van den Briel, R. S. Nigenda, M. B. Do, and S. Kambhampati. Effective approaches for partial satisfaction (over-subscription) planning. In *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI)*, pages 562–569, 2004.

[117] J. van den Burg. *Path Planning in Dynamic Environment*. PhD thesis, Utrecht University, 2007.

[118] J. Vleugels and M. Overmars. Approximating generalized Voronoi diagrams of convex sites in any dimension. *International Journal of Computational Geometry and Applications*, 8:201–221, 1998.

[119] R. Wein, J. P. van den Berg, and D. Halperin. The visibility-Voronoi complex and its applications. In *Proceedings of the Annual Symposium on Computational Geometry (SCG)*, pages 63–72, New York, NY, USA, 2005.

[120] E. Welzl. Constructing the visibility graph for n line segments in $O(n^2)$ time. *Information Processing Letters*, 20:107–171, 1985.

[121] S. Wilmarth, N. Amato, and P. Stiller. MAPRM: A probabilistic roadmap planner with sampling on the medial axis of the free space. In *Proceedings*

of the IEEE International Conference on Robotics & Automation (ICRA), pages 1024–1031, Detroit, Michigan, USA, 1999.

[122] E. Wolfstetter. Auctions: An introduction. *Journal of Economic Surveys*, 10:367–420, 1996.

[123] D. T. Wooden. *Graph-based Path Planning for Mobile Robots*. PhD thesis, School of Electrical and Computer Engineering, December 2006.

[124] A. Yahja, A. Stentz, S. Singh, and B. Brumitt. Framed-quadtree path planning for mobile robots operating in sparse environments. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, pages 650–655, Leuven, Belgium, May 1998.

[125] J. Yang, R. Codd-Downey, P. Dymond, J. Xu, and M. Jenkin. Planning practical paths for tentacle robots. In *Proceedings of the International Conference on Agents and Artificial Intelligence (ICAART)*, pages 128–137, Barcelona, Spain, 2013.

[126] J. Yang, P. Dymond, and M. Jenkin. Hierarchical probabilistic estimation of robot reachable workspace. In *Proceedings of the 6th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, pages 60–66, Milan, Italy, 2009.

[127] J. Yang, P. Dymond, and M. Jenkin. Practicality-based probabilistic roadmaps method. In *Proceedings of the Canadian Conference on Computer and Robot Vision (CRV)*, pages 102–108, St Johns, Newfoundland, Canada, 2011.

[128] J. Yang, P. Dymond, and M. Jenkin. Integrating multiple soft constraints for planning practical paths. Submitted for publication in the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Chicago, U.S.A, 2014.

[129] M. Zefran, V. Kumar, and X. Yun. Optimal trajectories and force distribution for cooperating arms. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, pages 874–879, San Diego, CA, USA, 1994.

[130] M. N. Zeilinger, C. N. Jones, and M. Morari. Robust stability properties of soft constrained MPC. In *Proceedings of the IEEE Conference on Decision and Control*, pages 5276–5282, 2010.

[131] S. Zickler. *Physics-Based Robot Motion Planning in Dynamic Multi-Body Environments*. PhD thesis, Carnegie Mellon University, 2010.