“ A walk through Motion Planning Algorithms and MoveIt! ”

Chandana Darapaneni, IIT Jodhpur

ug201313006@iitj.ac.in

Index

1. Introduction
2. Background
   a) Motion planning methods
      (i) Exact and approx. cell decomposition
      (ii) Control based methods
      (iii) Potential fields
      (iv) Randomized techniques
      (v) Sampling based methods
3. Sampling based methods
   a) Probabilistic RoadMapping
   b) Tree-based planners
   c) What can we expect from sampling based planning
4. Planners available in MoveIt!
   a) OMPL
   b) STOMP
   c) CHOMP
   d) SBPL
5. Conclusion & Remarks
6. References

1.Introduction

The classic problem of motion planning, the infamous '*Piano Movers Problem'* goes like: **Go from the start to goal positions without colliding with any of the obstacles in the PSPACE** (in an optimal way added optionally). In a computational point of view it has been proven that solving this is very hard. The setup contains a piano of 6DOF, 3-D environment, known set of obstacles and 6 different values are to be calculated for every manipulation to be done minding the constraints. [1][2][3] This document is a journey through different motion planning approaches and the corresponding algorithms available, Also a briefing about planners/planning libraries available in MoveIt! ROS. Paying more attention to Open-source Motion Planning Library (OMPL).

2.Background

2.a.Motion Planning Methods

A brief overview of methods that were proposed to solve the motion planning problem are as follows chronologically[4].

2.a.i.Exact and Approximate Cell decomposition

In some instances, it is possible to partition the workspace into discrete cells corresponding to the obstacle free portion of the environment. This decomposition can then be modeled as a graph (roadmap), where the vertices represent the individual cells and edges indicate adjacency among the cells. Utilizing this graph, the problem of moving a point robot becomes a classical search from the cell with the starting position to the cell containing the goal position.

Such a formulation also works well in "controllable" systems (e.g., omnidirectional base). There are cases though, for example when the robot has complex non-linear dynamics, where it is not clear how to move the system from one cell to an adjacent cell. It should also be noted that the motion of robots are computed in states spaces that are high-dimensional. Partitioning these spaces into free (or approximately free) cells is difficult and impractical in most cases.

2.a.ii. Control based methods

Control-based methods attempt to model the equations of motion of the system and apply ideas from control theory in order to navigate a system along a specified trajectory. These approaches operate in the continuous space, and typically employ a feedback loop in order to effectively maneuver the system with minimal error. Using a control-based approach to navigate a robot is very fast and can be done in an online manner, which is necessary in many applications. Computing a desirable and feasible trajectory can be very difficult, however, in systems with complex dynamics and/or cluttered environments, which heavily restrict the valid motions

2.a.iii. Potential Fields

Conceptually, potential fields are a simple and intuitive idea. The classical potential field involves computing a vector at each point in the workspace by calculating the sum of an attractive force emanating from the goal, and a repulsive force from all of the obstacles. The point robot can then navigate using gradient descent to follow the potentials to the goal. Potential fields have been generalized to systems with a rigid body by considering multiple points on the robot, and can operate in real time.

Navigating a system using a potential field, however, can fail due to local minima in the field itself, which stem from the heuristic combination of the forces in the workspace. Ideally the field would be constructed in the state space of the system, but this is equivalent to solving the original problem. Some approaches consider a navigation function where the potential field is guaranteed to have a single minimum, but computing such a function is possible only in low-dimensional spaces and is non-trivial.

2.a.iv. Randomized Planning

Randomization in otherwise deterministic planners has shown to be very effective. In potential fields, for example, Brownian motions in which a random action is applied for a specific amount of time have been shown to be highly effective in guiding a system out of a local minima. Randomized methods have also been applied to non-holonomic systems, particularly in the field of sampling-based planning. Sampling-based methods were inspired by randomization, and the use of samples, random or deterministic, when planning are particularly effective for high degree of freedom systems or those with complex dynamics.

2.a.v. Sample based planning

Sampling-based motion planning is a powerful concept that employs sampling of the state space of the robot in order to quickly and effectively answer planning queries, especially for systems with differential constraints or those with many degrees of freedom. Traditional approaches to solving these particular problems may take a very long time due to motion constraints or the size of the state space. Sampling arises out of the need to quickly cover a potentially large and complex state space to connect a start and goal configuration along a feasible and valid path.

The need to reason over the entire continuous state space causes traditional approaches to breakdown in high-dimensional spaces. In contrast, sampling-based motion planning reasons over a finite set of configurations in the state space. The sampling process itself computes in the simplest case a generally uniform set of random robot configurations, and connects these samples via collision free paths that respect the motion constraints of the robot to answer the query. Most sampling-based methods provide probabilistic completeness. This means that if a solution exists, the probability of finding a solution converges to one as the number of samples reasoned over increases to infinity. Sampling-based approaches cannot recognize a problem with no solution.

3.Sampling Based Methods

Here, we discuss two types of sampling-based planners, from which many of the state-of-the-art techniques can be derived from. It should be noted that many sampling strategies exists, but our discussion is constrained to the existing ones. Terminology to mind throughout the document:

Workspace: The physical space that the robot operates in. It is assumed that the boundary of the workspace represents an obstacle for the robot.

State space: The parameter space for the robot. This space represents all possible configurations of the robot in the workspace. A single point in the state space is a state.

Free state space: A subset of the state space in which each state corresponds to an obstacle free configuration of the robot embedded in the workspace.

Path: A continuous mapping of states in the state space. A path is collision free if each element of the path is an element of the free state space.

3.a.Probabilistic Road Mapping

The probabilistic roadmap (PRM) is among the first sampling-based motion planners [5]. This approach utilizes random sampling of the state space to build a roadmap of the free state space. This roadmap is analogous to a street map of a city. To illustrate the fundamentals of the PRM, a simple example of a 2D workspace and freely moving point robot will be used.

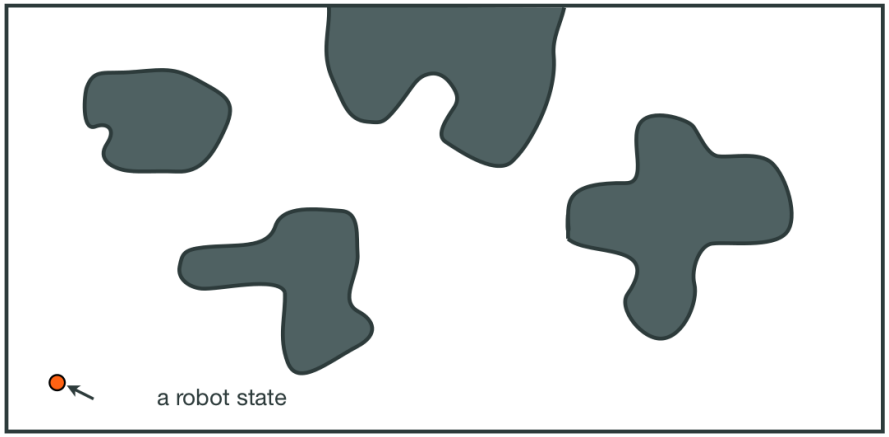


Figure 3.1 The 2D workspace and a single state for the point robot

Consider the workspace in Figure 3.1. This image shows a bounded workspace for the point robot in which the shaded regions are obstacles. One particular state for the robot is highlighted. The PRM works by uniformly sampling the free state space and making connections between the samples to form a roadmap of the free state space. The roadmap can be stored efficiently as a graph data structure where the random samples compose the vertices, as in Figure 3.2. It should be noted that the free state space is almost never explicitly known in sampling-based methods. Each sample that is generated is checked for collision, and only collision free samples are retained. Additionally, there are many different ways to sample the free state space, and changing the sampling strategy is beneficial in many planning instances.

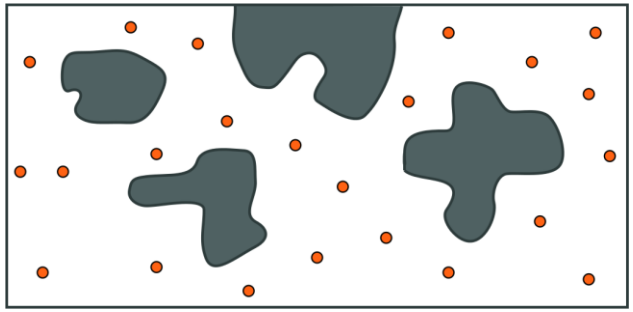


Figure 3.2: One possible set of uniform random samples of the free state space

Once the desired number of free samples have been found, the roadmap itself can be constructed by connecting the random samples to form edges. The canonical PRM attempts to connect each sample to the k samples nearest to it by using a local planner that is tasked with finding short collision free paths. The local planner finds this path by interpolating the motion of the robot between the two samples, checking for collisions at some prescribed resolution. If no configuration of the robot between the samples collides with an obstacle, then an edge is inserted to the roadmap. Figure 3.3 shows a complete probabilistic roadmap in the 2D workspace example. Once the roadmap is complete, it can be used to answer motion planning queries by connecting the start and goal states to the roadmap using the local planner, and performing a graph search to find the shortest path in the roadmap. This is seen in Figure 3.4.

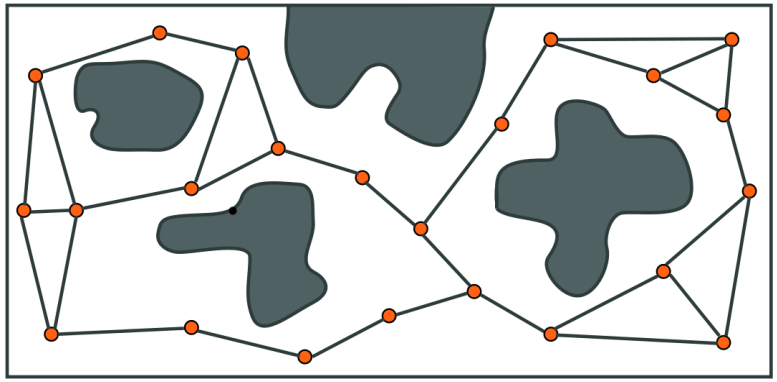


Figure 3.3

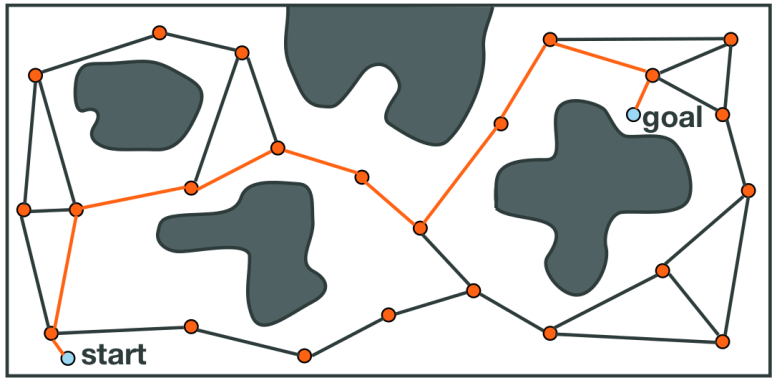


Figure 3.4

- Figure 2.3: Using a local planner, the PRM is formed by connecting samples that are close to one another using a straight path in the free state space.
- Figure 2.4: An example of a motion planning query on a PRM. The start and the goal are connected to the roadmap, and the shortest path in the graph is found.

Input:

n : number of nodes to put in the roadmap

k : number of closest neighbors to examine for each configuration

Output:

A roadmap $G = (V, E)$

```
1:  V ← ∅
2:  E ← ∅
3:  while |V| < n do
4:     repeat
5:        q ← a random configuration in Q
6:     until q is collision-free
7:     V ← V ∪ {q}
8:  end while
9:  for all q ∈ V do
10:    N_q ← the k closest neighbors of q chosen from V according to dist
11:    for all q' ∈ N_q do
12:       if (q, q') ∉ E and Δ(q, q') ≠ NIL then
13:          E ← E ∪ {(q, q')}
14:       end if
15:    end for
16: end for
```

>>Road Map constructing Algorithm [2]

Input:

q_{init}: the initial configuration

q_{goal}: the goal configuration

k: the number of closest neighbors to examine for each configuration

$G = (V, E)$: the roadmap computed by algorithm 6

Output:

A path from q_{init} to q_{goal} or failure

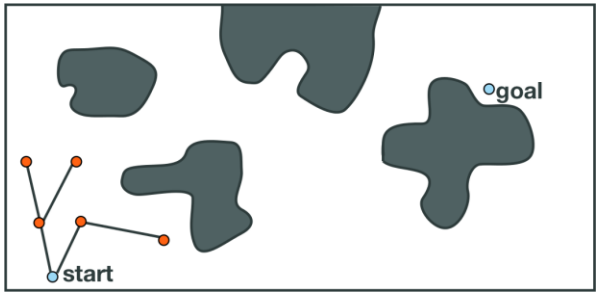
1: $N_{q_{\text{init}}} \leftarrow$ the k closest neighbors of q_{init} from V according to *dist*
2: $N_{q_{\text{goal}}} \leftarrow$ the k closest neighbors of q_{goal} from V according to *dist*
3: $V \leftarrow \{q_{\text{init}}\} \cup \{q_{\text{goal}}\} \cup V$
4: set q' to be the closest neighbor of q_{init} in $N_{q_{\text{init}}}$
5: **repeat**
6: **if** $\Delta(q_{\text{init}}, q') \neq$ `NIL` **then**
7: $E \leftarrow (q_{\text{init}}, q') \cup E$
8: **else**
9: set q' to be the next closest neighbor of q_{init} in $N_{q_{\text{init}}}$
10: **end if**
11: **until** a connection was succesful or the set $N_{q_{\text{init}}}$ is empty
12: set q' to be the closest neighbor of q_{goal} in $N_{q_{\text{goal}}}$
13: **repeat**
14: **if** $\Delta(q_{\text{goal}}, q') \neq$ `NIL` **then**
15: $E \leftarrow (q_{\text{goal}}, q') \cup E$
16: **else**
17: set q' to be the next closest neighbor of q_{goal} in $N_{q_{\text{goal}}}$
18: **end if**
19: **until** a connection was succesful or the set $N_{q_{\text{goal}}}$ is empty
20: $P \leftarrow$ shortest path$(q_{\text{init}}, q_{\text{goal}}, G)$
21: **if** P is not empty **then**
22: **return** P
23: **else**
24: **return** `failure`
25: **end if**

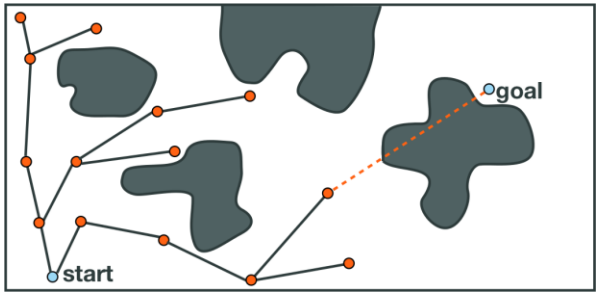
>> Solve query Algorithm [2]

There are other variations of PRM too but the main concepts underlying the strategies have been discussed here.

3.b.Tree based techniques

There exist many types of sampling-based planners that create tree structures of the free statespace. The trees generated by these methods are analogous to the probabilistic roadmap, except that the structure contains no cycles. There are a wide variety of tree based methods(e.g., RRT[6], EST[7],SBL[8], KPIECE[9]). To describe a genereal framework: These methods begin by rooting a tree at the starting configuration of the robot. With the first node of the tree intact, random sampling of the free space then occurs. The planner employs an expansion heuristic, which typically gives the method its name, from which the sample is connected to the tree along a collision free path. Figure 3.5 shows an example in the 2D workspace scenario where the first few valid samples are connected to the tree. Since it is highly improbable that the sampling process will ever sample the goal state exactly, the methods often bias the expansion of the tree toward the goal state. If it is possible to connect the goal to the existing tree, then the search is complete; a path through the free state space has been found from the start to the goal. Figure 3.6 shows a case where the goal cannot be connected to the tree, and Figure 3.7 shows the case where the goal is connected, terminating the search.





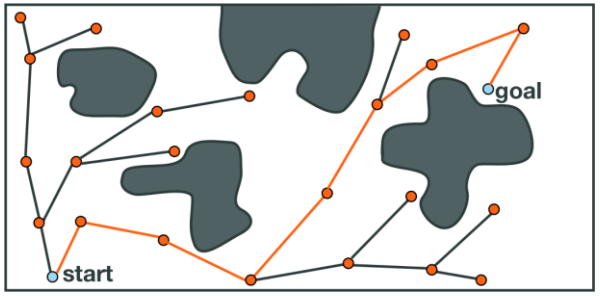


Figure 3.5 **Figure 3.6** **Figure 3.7**

It's important to differentiate between the roadmap-based planners and tree-based planners.

The tree-based techniques are most suitable for single-query planning. These trees do not normally cover the free space in the same manner that a roadmap would. However, when planning with differential constraints, it is not easy to encode control information into an undirected edge. Controls are usually directed commands, and require a specific pre-condition in order for a particular control to be valid. Tree-based methods, on the other hand, excel at planning with complex dynamics because of the directed, acyclic nature of the underlying data structure. Control information can be encoded for each edge of the tree, with the vertices of the tree satisfying the prerequisites for the valid controls. Finally, it should be noted that many sampling-based approaches require a smaller memory footprint than other motion planners. The compactness of these planners stems from the sampling process itself, as well as the fact that no explicit representation of the state space is needed in order to solve the problem. Storage and search of the underlying data structure (e.g., graph,tree) should be efficient to fully maximize the quality of these methods. However, as tree planners are used for increasingly difficult problems, memory requirements may increase in order to keep information that guides the expansion of the search.

3.c.What can we expect from Sampling based methods

Sampling-based planning is a very powerful tool for planning in high-dimensional spaces or for system with complex dynamics. There exists many kinds of sampling-based motion planners, with many commonalities, but the method in which one samples the state space is key to computing a solution.

Collision checking is a very important part of sampling-based planning. It is used not only in the local planner when attempting to find collision free paths between samples, but also during the sampling process itself. In a complex or high-dimensional system it may not be easy to explicitly represent the free state space, but in sampling-based methods it is not necessary to create this space. It is the job of the collision checker to accept a configuration of the robot and quickly determine whether or not this state is in collision. Nearest neighbor searching is another cornerstone of sampling-based methods. It is from the ability of determining whether two states of the robot are close that many of the common approaches are able to effectively find paths through a high-dimensional space. Distances, however, are not easy to compute in non-Euclidean spaces where many of the interesting problems reside. Kd-trees offer one way to perform this search, but the optimal connection strategy for samples remains elusive.

The core OMPL library cleanly implements all of these primitives of sampling-based motion planning. Further, let's see how the planners bundled with OMPL can be used to directly solve motion planning queries, and Chapter 4 details how these primitives map to concepts within the open motion planning library.

4.Planners in MoveIt!

MoveIt! is designed to work with many different types of planners, which is ideal for benchmarking improved planners against previous methods. Below is a list[10] of planners that have been used with MoveIt!, in descending order of popularity/support within MoveIt!:

4.a.Open Motion Planning Library (OMPL)

OMPL is an open-source motion planning library that primarily implements randomized motion planners. MoveIt! integrates directly with OMPL and uses the motion planners from that library as its primary/default set of planners. The planners in OMPL are abstract; i.e. OMPL has no concept of a robot. Instead, MoveIt! configures OMPL and provides the back-end for OMPL to work with problems in Robotics. Fully supported. [4]

4.b.Stochastic Trajectory Optimization for Motion Planning (STOMP)

STOMP (Stochastic Trajectory Optimization for Motion Planning) is an optimization-based motion planner based on the PI^2 (Policy Improvement with Path Integrals, Theodorou et al, 2010) algorithm. It can plan smooth trajectories for a robot arm, avoiding obstacles, and optimizing constraints. The algorithm does not require gradients, and can thus optimize arbitrary terms in the cost function like motor efforts. Partially supported. [11]

4.c.Covariant Hamiltonian Optimization for Motion Planning (CHOMP)

Covariant Hamiltonian optimization for motion planning (CHOMP) is a novel gradient-based trajectory optimization procedure that makes many everyday motion planning problems both simple and trainable (Ratliff et al., 2009c). While most high-dimensional motion planners separate trajectory generation into distinct planning and optimization stages, this algorithm capitalizes on covariant gradient and functional gradient approaches to the optimization stage to design a motion planning algorithm based entirely on trajectory optimization. Given an infeasible naive trajectory, CHOMP reacts to the surrounding environment to quickly pull the trajectory out of collision while simultaneously optimizing dynamical quantities such as joint velocities and accelerations. It rapidly converges to a smooth collision-free trajectory that can be executed efficiently on the robot. Integration into latest version of MoveIt! is work in progress. [12]

4.d.Search-Based Planning Library (SBPL)

A generic set of motion planners using search based planning that discretize the space. Integration into latest version of MoveIt! is work in progress. [13]

5.Conclusion & Remarks

OMPL includes two types of motion planners: ones that do not consider controls when planning and ones that do[14]. If a planning algorithm can be used to plan both types of motions, with and without controls (e.g., RRT [6]), two separate implementations are provided for that algorithm, one for each type of computed motion. This choice was made for efficiency reasons. With additional levels of abstraction in the implementation it would have been possible to avoid separate implementations [15]. The downside would have been that the implementation of planners would have had to follow a strict structure, which makes the implementation of new algorithms more difficult and possibly less efficient. For purely geometric planning (i.e., controls are not considered), the solution path is constructed from a finite set of segments, and each segment is computed by interpolation between a pair of sampled states (PathGeometric). Several geometric planning algorithms are implemented in OMPL, including KPIECE [9], bidirectional KPIECE, bidirectional lazy KPIECE, RRT [6], RRT-connect , lazy RRT, SBL [8], EST [7], and PRM. The “lazy” variants listed above defer state validity checking in the manner described in [16]. In addition, there are multi-threaded versions of RRT and SBL. When controls are considered, the solution path is constructed from a sequence of controls (PathControl). Controlbased planners are typically used when motion plans need to respect differential constraints as well. Several algorithms for planning with differential constraints are implemented in OMPL as well, including KPIECE, SyCLoP [17], EST, and RRT.

6.References

[1] Jean-Claude Latombe. *Robot Motion Planning.* Kluwer academic publishers, Norwell, USA 1991.

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

[3] Steven M LaValle, *Planning Algorithms.* Cambridge University Press, USA 2006

[4] https://ompl.kavrakilab.org . *OMPL- A Primer.* Kavraki Lab,Rice University

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

[6] Steven M. LaValle and James J. Kuffner. *Randomized kinodynamic planning*. International Journal of Robotics Research, 17(5):378–400, 2001.

[7] David Hsu, Jean-Claude Latombe, and Rajeev Motwani. *Path planning in expansive configuration spaces.* International Journal of Computational Geometry and Applications, 9(4/5):495– 512, 1999.

[8] Gildardo Sanchez and Jean-Claude Latombe. *A single-query bi-directional probabilistic roadmap planner with lazy collision checking*. In Robotics Research, volume 6 of Springer Tracts in Advanced Robotics, pages 403–417. 2003.

[9] Ioan ¸Sucan and Lydia E. Kavraki. *A sampling-based tree planner for systems with complex dynamics*. IEEE Transactions on Robotics, 28(1):116–131, 2012.

[10] http://moveit.ros.org/documentation/planners/ *Planners | MoveIt!*

[11] M Kalakrishnan, S Chitta, E Theodorou, P Pastor, S Schaal, *STOMP: Stochastic trajectory optimization for motion planning* Robotics and Automation (ICRA), 2011 IEEE International Conference on, 4569-4574

[12] Nathan Ratliff, Matthew Zucker, J. Andrew Bagnell, Siddhartha Srinivasa, *Covariant Hamiltonian Optimization for Motion Planning,* http://www.nathanratliff.com/thesis-research/chomp

[13] *Search based Planning with motion primitives,* Maxim Likhachev, CMU,2010 http://wiki.ros.org/Events/CoTeSys-ROS-School?action=AttachFile&do=get&target=robschooltutorial_oct10.pdf

[14] Ioan A. Sucan, Mark Moll and Lydia E. Kavraki, *The Open Motion Planning Library,* IEEE Robotics & Automation Magazine, December 2012.

[15] E. Plaku, K. E. Bekris, and L. E. Kavraki, *OOPS for motion planning: An online open-source programming system*, in Proc. 2007 IEEE Intl. Conf. on Robotics and Automation, Rome, Italy, 2007, pp. 3711–3716.

[16] R. Bohlin and L. E. Kavraki, *Path planning using lazy PRM*, in Proc. 2000 IEEE Intl. Conf. on Robotics and Automation, San Francisco, CA, 2000, pp. 521–528.

[17] E. Plaku, L. Kavraki, and M. Vardi, “Motion planning with dynamics by a synergistic combination of layers of planning,” IEEE Trans. On Robotics, vol. 26, no. 3, pp. 469–482, jun. 2010.