Comparison and Implementation of PRM and Lazy PRM on mobile robots

Gunjan Khut, gkkhut@terpmail.umd.edu, UID - 115813083

Utsav Patel, upatel22@terpmail.umd.edu, UID - 115816756

***Abstract* — Asymptotically optimal sampling-based motion planners, like RRT, perform vast amounts of collision checking and are therefore rather slow to converge in complex problems where collision checking is comparatively expensive. This paper describes a new approach to probabilistic roadmap planners (PRMs). The overall theme of the algorithm called Lazy PRM, whose aim is to minimize the number of collision checks performed during the planning and hence minimize the running time of the planner. The algorithm builds a roadmap in the configuration space, whose nodes are the user-defined start and goal configurations and a number of randomly generated nodes. Neighboring nodes are connected by edges representing paths between the nodes. In contrast with PRMs, our planner initially assumes that all nodes and edges in the roadmap are collision-free and searches the roadmap at hand for the shortest path between the initial and the goal node. The nodes and edges along the path are then checked for collision. If a collision with the obstacles occurs, the corresponding nodes and edges are removed from the roadmap. Our planner either finds a new shortest path, or first updates the roadmap with new nodes and edges, and then searches for the shortest path. The above process is repeated until a collision-free path is returned. Lazy PRM is tailored to efficiently answer single planning queries but can also be used for multiple queries. Experiments show that the new methods converge toward the optimum substantially faster than existing planners on rigid body path planning problems.**

***Keywords*— PRM, lazy-PRM, Pathfinding algorithm, mobile robots, sampling-based motion planners.**

I. INTRODUCTION

The basic optimal motion planning problem asks to find a cost-minimizing path subject to obstacle avoidance constraints and has been a highly active research topic since the development of sampling-based planners with proven asymptotic optimality properties (e.g., RRT [1]).

The sampling-based planners work well in high-dimensional spaces for the feasible motion planning problem because their running time is not explicitly dependent on dimensionality. It may work well for the optimal motion planning problem in many dimensions. But the practical experience has demonstrated slow convergence to the optimum, especially as dimension increases.

Convergence is typically slow because, once a path is found, sampling-based planners need to sample a huge number of configurations before finding one that shortens the current best path. Each additional sample adds to the cumulative cost of collision checking because testing is done on both the configuration and edges leading to nearby configurations.

This is wasteful since most work is being performed on elements with no opportunity to be part of an optimal path. Specifically, as the path approaches optimality, the region of free space that might contain a shorter one approaches a set of measure zero.

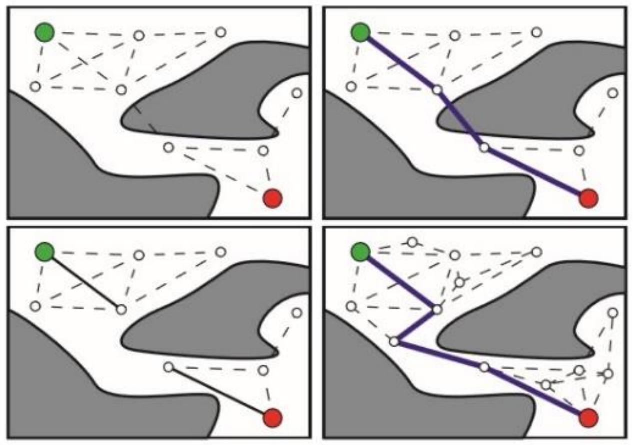


Fig. 1. Illustrating the lazy planner. Edges in the roadmap are not checked for collision (dotted lines) until a candidate path to the goal is found. Feasible edges are marked (solid lines) and infeasible ones are deleted. The process repeats until a feasible path to the goal is found.

Inspired by this observation, this paper proposes a lazy strategy [2] to drastically reduce the fraction of edges that undergo collision checking. This benefit is substantial because edge collision checking accounts for most of collision checking time. This presents the planners, Lazy-PRM, which build a roadmap of feasible configurations (also known as milestones), but with the difference that edges are not immediately checked for collision. If a better candidate path to the goal is found, then all edges along that path are checked for collision. Edges that fail the check are removed from the roadmap (Fig. 1). This strategy eliminates most of the cost of collision checking, at the cost of greater bookkeeping of shortest paths along the roadmap. We use a dynamic shortest paths algorithm [3] and other strategies to reduce the overhead of recomputing shortest paths.

We use a dynamic shortest path algorithm and other strategies to reduce the overhead of recomputing shortest paths. Lazy-PRM converges up to several orders of magnitude faster than PRM, RRT, and other asymptotically-optimal planners.

II. METHOD

In this paper, we are going to compare the run time of two sampling-based motion planning algorithms which are PRM and Lazy PRM. First, we are going to implement the PRM method and after implementing the PRM we will be implementing the Lazy PRM. The implementation of both methods will be done on Turtlebot2, and the C space will be the RRL lab of the University of Maryland. The details of the C space and the turtle bot are given in the following sections.

A. Turtlebot 2 – It is a mobile robot with an open source software. Which is equipped with a YUJIN Kobuki base, a 2200 mAh battery pack, a Kinect sensor, an Asus 1215N laptop with a dual-core processor, fast charger, and a hardware mounting kit attaching everything together and adding future sensors [4]. The Turtlebot 2 uses differential drive mechanism to drive around.

Fig. 2. Turtlebot 2 [4]

The Turtlebot 2 has following dimensions [5]

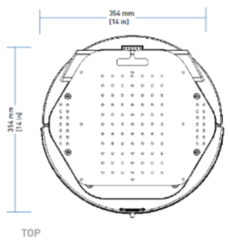


Fig. 3. Top View

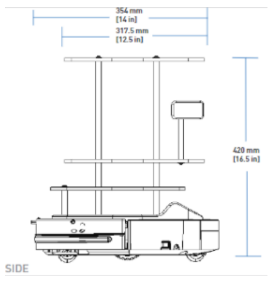


Fig. 4. Side View

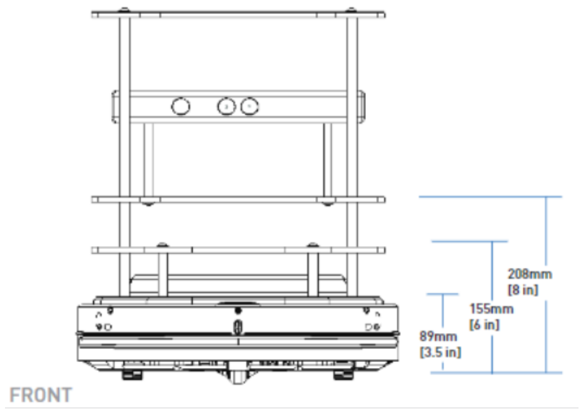


Fig. 5. Front View

The main dimensions L X W X H = 354 x 354 x 420 mm

- Configuration space - Here, we are planning to implement the PRM and Lazy PRM to find a path in RRL lab at the University of Maryland, College Park. The top view of the lab is given in the Fig. 6.

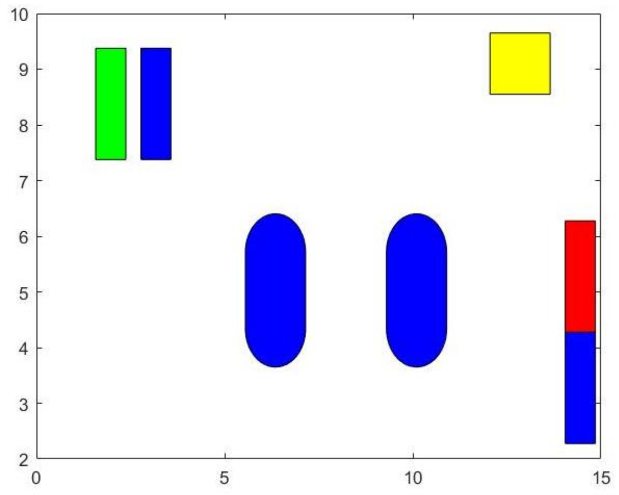


Fig. 6. Configuration Space of RRL Lab

The path planning algorithms assume the Turtlebot 2 as a point robot, so we cannot directly implement the algorithms on the configuration space given in Fig. 6. To consider the Turtlebot 2 as a point robot we need to inflate the obstacles in the configuration space by the radius of the Turtlebot 2. So, the first step to implement PRM and Lazy PRM will be the inflation of the obstacles in the configuration space. The configurations space with the inflated obstacles is shown in Fig. 6.

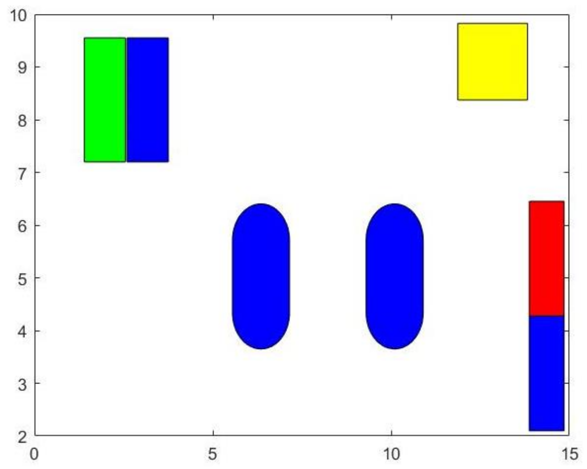
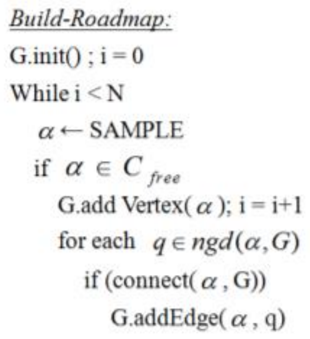


Fig. 7. Inflated configuration space

B. Implementation of the PRM [6] [7] [8] –



```
Build-Roadmap:
G.init() ; i = 0
While i < N
    $\alpha \leftarrow$ SAMPLE
    if $\alpha \in C_{free}$
        G.add Vertex( $\alpha$ ); i = i+1
        for each $q \in ngd(\alpha, G)$
            if (connect( $\alpha$ , G))
                G.addEdge( $\alpha$ , q)
```

Fig. 8. Algorithm for PRM

The algorithm is shown in Fig. 8, it starts with an empty graph: G (Vertex, Edge). First, we select the start and the goal point. Assume that we are going to generate N random points. While *i* is less than N, a sample configuration is generated and assigned to α. Then we run the collision detector to find out whether α is in the collision. If α is collision-free, add α to the graph G. After that, we need to find out all the edges that α can connect to.

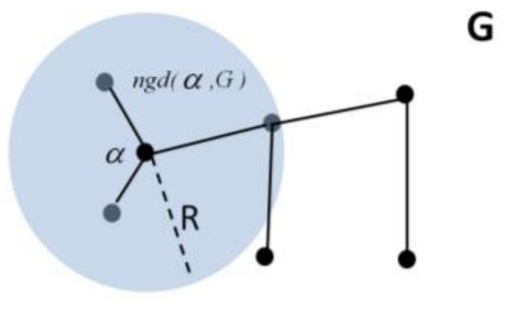


Fig. 9. Definition of ngd (α, G)

Define and((α, G) as the set of candidate neighbors which is chosen from all vertexes inside G. This set is made of nodes within a certain distance of R. Fig. 8 illustrates the definition of ngd((α, G). For each point in the set ngd((α, G), which is one of the points in the blue area in Figure 9, we try to connect it to α by using a straight line if it is not already graph-connected to α. The distance of neighbor R is the number that tells you the scope of how far you think you can connect to configurations by straight lines. There is a tradeoff when choosing R. If R is small, neighbor points should be close to α, which is easier to connect but takes a lot of time before filling up the whole search space. If R is large, α is able to connect far away points with straight lines, but it will be easier to fail. The algorithm says the α should be small. The idea is that if you start building the graph and every point is very close to each other, eventually it will be very easy to chain all the points together to go from any point to any goal. If there is no obstacle in the configuration space within the straight line, we add the edge to the graph G. Otherwise, we just remove the edge. Then we keep going until we go through all the N sample points. After generating the map, the Start point and the goal points are connected to the map and the shortest path that connects the start and goal point is found.

C. Implementation of Lazy PRM [8] [9] [10] –

The basic idea of the Lazy-PRM algorithm is to delay collision check, until Inevitable.

Firstly, assume the C-space is clearly open, and then create a dense PRM without ANY collision checking. When given start point and goal point, the algorithm will find the path from the start point through the graph to the goal. This process will be very fast. And then, take the path just found and check collisions. Different from normal PRM, now the program only needs to check the collision just along the path, instead of to check collisions of the whole graph. If any collision is found (the path is broken), then remove the edge and re-plan the path. The algorithm will re-route around the broken edge and repeat the whole process again until it gets a path to the goal.

Lazy-PRM is one way to bridge the gap between high sparsity and doing less collision check. The worst-case performance of lazy PRM will be the same as normal PRM. This happens when basically every edge is broken so that the program will do the collision check for almost all the edges. But the average performance of lazy PRM will be much better than normal PRM.

D. Implementation in different C-Spaces.

We implemented PRM and Lazy PRM in four maps all with different complexities to observe the change in the runtime. We selected the same start and end points for all the four maps and used both the algorithms to generate the path. We also carried this process multiple times as we wanted to get a surety result and then could evaluate the outcomes of the algorithms. We have included the maps where we implemented PRM and Lazy PRM below:

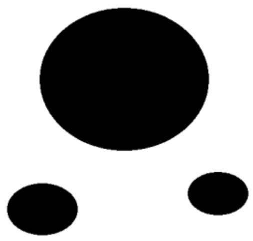


Fig. 10. Map 1

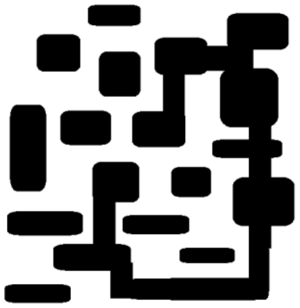


Fig. 11. Map 2

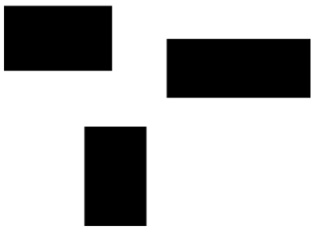


Fig. 12. Map 3

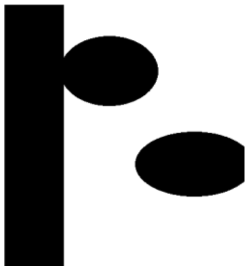


Fig. 13. Map 4

III. PREVIOUS WORK

Milestones (a roadmap of configurations) are built by sampling-based motion planners, which are sampled from the free space connected by straight line paths [11], [12]. Paths on the roadmap connecting the start and goal are the feasible solution paths in the free space. This was originally applied to the feasible motion planning problem, in which the cost of the path was ignored, researchers had applied these methods to the optimal planning problem, in which a minimum cost path was desired (e.g., the shortest path). In the feasible regime, sampling-based planners enjoy theoretical guarantees of probabilistically completeness, in that the probability of finding a feasible path, if one exists, approaches 1 as more milestones are sampled [11], [12]. Moreover, they have state-of-the-art empirical performance in finding feasible paths in high-dimensional spaces. not like methods based on grid search or cell decomposition, that run-in time exponential in dimension, the expected time to find a feasible path isn't directly dependent on dimensionality, but rather on the visibility characteristics of the free space [13]. As a result, there has been enormous interest in applying sampling-based methods to the optimal motion planning problem.

Karaman and Frazzoli recently presented the RRT* tree-growing algorithm and PRM* graph growing algorithm, that are proven to be asymptotically optimal [14]. Their key contribution uses a result from the theory of percolation in random graphs to determine connection neighborhoods that guarantee that the shortest path within the roadmap approaches the optimal path in free space. However, theory predicts O(1/d) samples are required to achieve an -suboptimal path in d dimensions, that is no better than grid search, and experiments confirm that convergence rate is rather slow in high dimensional spaces. As a result, recent work aims to improve convergence rate. Early convergence can be accelerated by checking fewer edges to arrive at suboptimal solutions faster [15], with recent approaches provably converging toward near-optimality (e.g., LBT-RRT* [16], SPARS [17]). other approaches reduce time per-step once the roadmap grows dense, either by using sampling strategies that place samples wherever they are more likely to yield an improvement in the optimal path [18], or by using C-obstacle distance queries to incrementally cover free-space with balls that are known to be collision-free, that allows collision checks to be skipped for samples close to existing milestones [19].

Lazy edge collision checking has been used in the past to speed up sampling-based motion planners, most notably Lazy-PRM [20] and SBL [21]. It's effective because edge checking is often one or two orders of magnitude more expensive than configuration checking. Existing edge checkers either 1) discretize finely and call many configuration checks, or 2) construct a representation of the swept volume as the robot moves along a configuration space path. both are computationally expensive. Lazy approaches add edges to the roadmap without checking and remove them after failing a collision test. However, not like previous approaches that stop after a first solution is found, we continue searching for progressively shorter paths. The speed gains are even more impressive than infeasible planning because the fraction of irrelevant edges grows toward 1 as the current best path approaches the optimal one. Another motion planner, FMT*, bears some likeness to this approach. It first builds a roadmap of feasible samples and computes a cost-to-come heuristic; it then expands a tree forward along the roadmap using the heuristic [22]. The differences are that 1) FMT* is not any-time, and 2) it checks edges along a tree whereas our approach checks edges along a path. This results in many fewer collision checks. The lazy strategy is applied to several underlying planners, leading to the Lazy-PRM* algorithm presented here. It's also compatible with various heuristics, like bidirectional planning and ellipsoidal pruning [23]. Experiments recommend that the effect of these heuristics is small compared to the overall effect of laziness. laziness.

IV. RESULTS

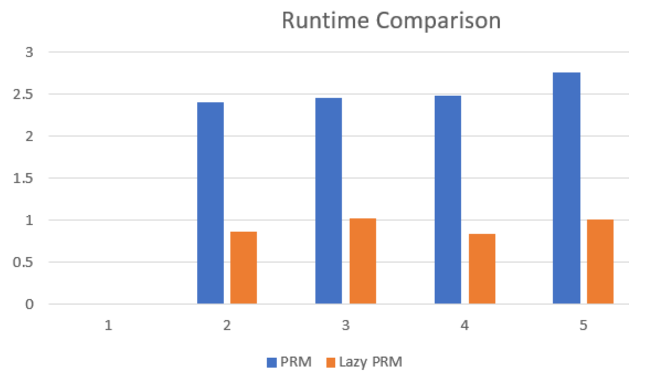


Fig. 14. Runtime comparison of PRM and Lazy PRM

The fig. 14 shows the comparison of runtime for PRM and Lazy PRM. The runtime is calculated on 4 different maps, all the maps have different obstacle configuration, so the complexity of the maps are different. For each map, we calculated the runtime for 3 times and then the average runtime is calculated, the values are shown in the graph. We can observe that in each case runtime for Lazy PRM is less than PRM. The percentage decrease in the runtime is also shown in fig. 15.

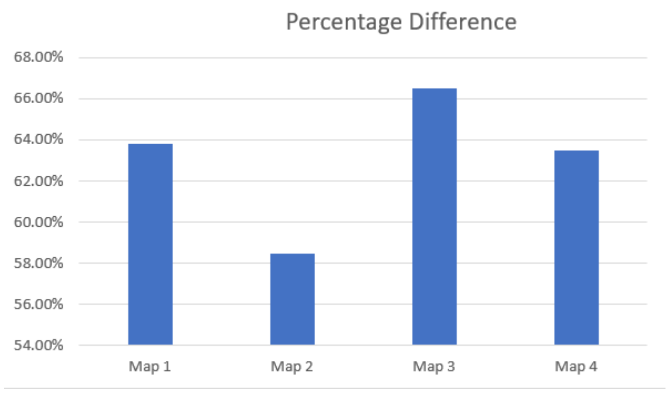


Fig. 15. Percentage difference

While implementing the algorithms on three different maps we found one interesting results. We can see that in case of map 2 the difference of the runtime for both algorithms is less. And in map 1 (fig. 10) the runtime difference of both maps is very high. The reason behind this anomaly can be easily explained. In map 2 (fig. 11) the complexity is very high in comparison to all other maps, so in both algorithms, the difference in a number of collision checks will not be very high and because of the less difference in the number of collision checks the runtime does not differ much for both algorithms.

After comparing the algorithms on 4 different maps, we ran the lazy PRM algorithm on the RRL lab of the University of Maryland, college park. we successfully generated the trajectory for the Turtlebot. The trajectory is shown in fig. 16. We simulated the Turtlebot in V-rep and the fig. 17 shows the screenshot of the simulation of Turtlebot.

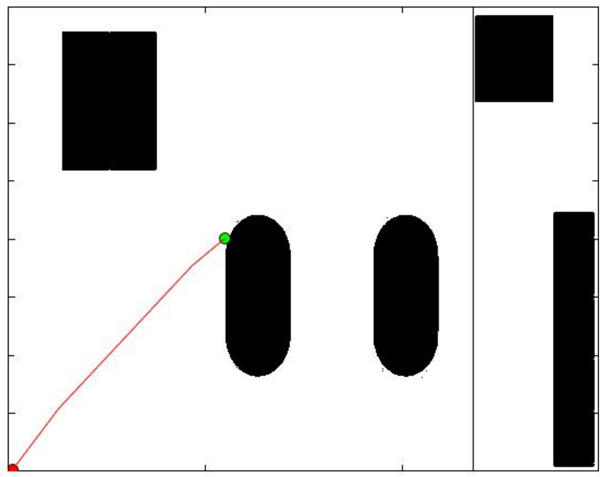


Fig. 16. RRL lab

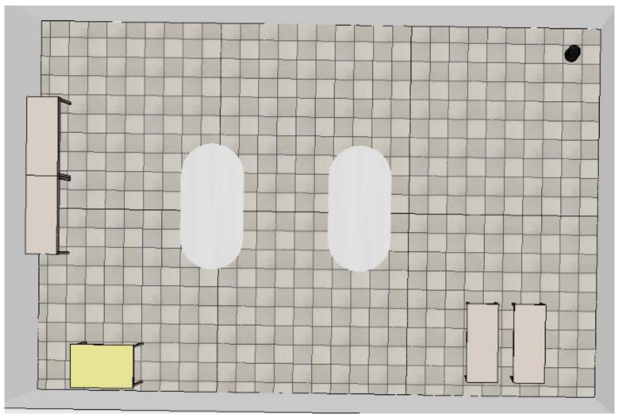


Fig. 17. V-rep implementation in RRL lab

V. SIMULATION DETAILS

In this paper, we implemented the PRM and Lazy PRM on turtlebot2. We simulated the algorithms in V-rep as in fig. 17. We wrote the code for both the algorithms in MATLAB, and used Remote API to establish the connection between MATLAB and V-rep. As the turtlebot2 has the differential drive, we need to consider differential constraints to move the turtlebot2 in the actual environment. The MATLAB code also generates the text file, which saves the velocity values for the robot. Now, to implement the algorithms on the real turtlebot2, we wrote a node in ROS, which reads the velocity values from the text file and publish those velocity values to move the turtlebot2 in actual environment.

VI. CONCLUSION

We can conclude from the results that, the runtime for the Lazy PRM is considerably less than PRM. The Lazy PRM algorithm considerably reduces the number of collision checks and as the collision check process being the most time-consuming process, the overall runtime reduces in the case of Lazy PRM. We have also shown in the results section that in the case of complex maps the difference in runtime is not much high, and we can conclude from the result that in case of highly complex maps, the Lazy PRM could not reduce the number of collision checks by a considerable amount. And the result is obvious as in case of highly complicated maps when the Lazy PRM finds the feasible path, the probability of the path colliding with the obstacles increases; and consequently, the algorithm needs to go over the map again to find the next feasible path.

VII. FUTURE WORK

In this paper, we mainly focused on the comparison of runtime for the PRM and Lazy PRM algorithm. We also concentrated mainly on mobile robots. In future, we can extend the comparison to the high dimensional C-spaces for manipulators, also we can extend the runtime comparison process for different paths with different lengths.

VIII. REFERENCES

[1] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. Int. J. Rob. Res., 30(7):846–894, 2011.

[2] R. Bohlin and L. E. Kavraki. Path planning using lazy prm. In IEEE Int. Conf. Rob. Aut. pages 521–528, San Francisco, CA, 2000.

[3] D. Frigioni, A. Marchetti-Spaccamela, and U. Nanni. Fully dynamic algorithms for maintaining shortest paths trees. J. Algorithms, 34(2):251 – 281, 2000.

[4] https://en.wikipedia.org/wiki/TurtleBot

[5] http://www.turtlebot.com/turtlebot2/

[6] http://www.cs.columbia.edu/~allen/F15/NOTES/Probabilisticpath.pdf

[7] https://www.cs.princeton.edu/courses/archive/fall11/cos495/COS495-Lab8-MotionPlanning.pdf

[8] https://personalrobotics.ri.cmu.edu/files/courses/16662/notes/rrt/16662_Lecture12.pdf

[9] Lazy collision checking in asymptotically-optimal motion planning by Kris Hauser

[10] Path Planning Using Lazy PRM by Robert Bohlin and Lydia E. Kavraki

[11] L. E. Kavraki, P. Svetska, J.-C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Trans. Robot. and Autom., 12(4):566–580, 1996.

[12] S.M. LaValle and J.J. Kuffner, Jr. Randomized Kino dynamic planning. Int. J. Rob. Res., 20(5):379 – 400, 2001.

[13] D. Hsu, J.-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In IEEE Int. Conf. Rob. Aut., pages 2219–2226, 1997.

[14] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. Int. J. Rob. Res., 30(7):846–894, 2011.

[15] R. Alter1ovitz, S. Patil, and A. Derbakova. Rapidly-exploring roadmaps: Weighing exploration vs. refinement in optimal motion planning. In IEEE Int. Conf. Rob. Aut., pages 3706–3712, 2011.

[16] O. Salzman and D. Halperin. Asymptotically near-optimal rrt for fast, high-quality, motion planning. In IEEE Int. Conf. Rob. Aut., Hong Kong, 2014.

[17] A. Dobson and K. E. Bekris. Sparse roadmap spanners for asymptotically near-optimal motion planning. Int. J. Rob. Res., 33(1):18–47, 2014.

[18] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot. Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In IEEE/RSJ Int. Conf. Intel. Rob. Sys., Chicago, USA, 2014.

[19] J. Bialkowski, M. Otte, S. Karaman, and E. Frazzoli. Efficient collision checking in sampling-based motion planning. In Alg. Found. Robotics X, volume 69. Springer, Berlin / Heidelberg, 2013.

[20] R. Bohlin and L. E. Kavraki. Path planning using lazy prm. In IEEE Int. Conf. Rob. Aut., pages 521–528, San Francisco, CA, 2000.

[21] G. Sanchez and J.-C. Latombe. On delaying collision checking in PRM planning: Application to multi-robot coordination. Int. J. of Rob. Res., 21(1):5–26, 2002.

[22] L. Janson and M. Pavone. Fast marching trees: a fast-marching sampling-based method for optimal motion planning in many dimensions. In Intl. Symp. Robotics Research, 2013.

[23] B. Akgun and M. Stilman. Sampling heuristics for optimal motion planning in high dimensions. In IEEE/RSJ Int. Conf. Intel. Rob. Sys., 2011.