An Exploration into PRM Maintenance in Dynamic 2D Environment

Jingran Shen

***Abstract*—PRM, which stands for Probabilistic Road Map, serves as a major family of sampling-based algorithms in the Motion Planning field. The feature of building static road map offline enables it to support the multi-query scenario, but also makes it suffer from dynamic environment changes. When new obstacles enter the environment, blocking existing nodes and edges on the pre-constructed road map, a dynamic version of PRM should be designed to update the road map and reconnect the separated sub-graphs of it. This report designs a dynamic PRM, namely *Repair-PRM*, which aims to 1) keep using the former feasible path before the new obstacle is added as much as possible, 2) selects two alternative end nodes on the former feasible path segments according to the evaluation of node freedom, and 3) utilizes RRT* to reconnect the separated sub-graphs of the road map by finding a short feasible path between two selected alternative nodes. Results show that Repair-PRM with node freedom evaluation and RRT* has a higher probability to output feasible paths with smallest costs upon new obstacle additions.**

***Index Terms*—Motion Planning, Sampling-based Method, Probabilistic Road Map, Dynamic Environment**

I. Introduction

SAMPLING-based methods have been widely adopted in the Motion Planning research field, mainly due to better time efficiency as well as increasing flexibility. At present, PRM (Probabilistic Road Map) [1] as well as RRT (Rapidly-exploring Random Tree) [2] serve as the two major sampling-based algorithms. Essentially, PRM focuses on building the road map, which is a static graph in the set-up phase. A specified number of points are randomly sampled in the free space (points that are collision-free to obstacles) and edges are formed among the sample points according to certain criteria. In the serving phase, the algorithm 1) receives a query, which is a point pair specifying the starting point and the goal point, 2) connects the two end points to the nearest sample points on the road map, and 3) outputs a shortest path between the two sample points utilizing algorithms like Dijkstra and A-Star. An example on PRM operation is shown in Figure 1. The starting point and the goal point are represented as red triangle and red diamond, respectively. The black bounding box with dots specifies the map region, while other black objects represent the obstacles in the environment. The cyan points and yellow solid lines represent the constructed static road map. Finally, a feasible path for the query is rendered in purple color. In particular, naive PRM constructs the road map offline for one single time. With the ability to support multi-query naturally, however, it is vulnerable to dynamic environment changes. For example, when a huge rock suddenly falls from the mountain and blocks the road, PRM will not be able to synchronize with such changes. In most real-world scenarios, the environment robots operate in is not static. As a result, it is necessary to consider dynamic environment changes and maintain the road map of the PRM algorithm at runtime.

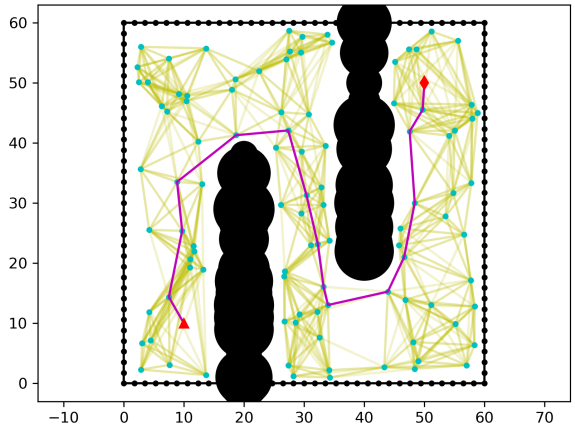

Fig. 1: PRM (Probabilistic Road Map): an example.

A. Problem Clarification

The report focuses on the maintenance of PRM algorithm in dynamic 2D environment. Basically, approaches to update the road map online are explored, in order to repair the algorithm when it malfunctions due to additions of new obstacles. In detail, an initial map with static obstacles is first input to the PRM solver, who then samples several points to build a road map. Afterwards, a query is passed to the solver for a feasible path. Next, a new static obstacle is added to the map to block the formerly feasible path. There are cases when such blocking event leads to the road map containing multiple disjoint sub-graphs. In this case, the broken road map becomes no longer capable of providing any feasible path. The method in this report aims to handle this specific case so that new nodes (i.e., points) and edges are added to repair the road map. Figure 2 shows a problem case where the new obstacle blocks the nodes and edges in the original road map, marked in red color.

This report is written for a course project (*EE5058 - Introduction to Information Technology, Spring 2023*) in SUSTech, which features on enhancing existing sampling-based motion planning methods, specifically PRM and RRT.

Jingran Shen (Peter S) is currently a Master student (SID: *12132354*) with the Department of Computer Science and Engineering, Southern University of Science and Technology (SUSTech), Shenzhen, China

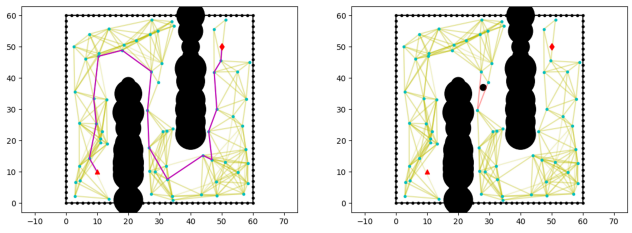

Fig. 2: New obstacle blocks the former feasible path.

B. Assumptions

For simplicity, the report assumes the following:

1) The robot operates on a 2D (two-dimensional) rectangle map, and is not expected to exit the map range. The edge of the map is surrounded by manually constructed point obstacles.
2) The robot as well as the obstacles in the environment are regarded as round shape.
3) The obstacles in the environment remain static once added or deleted.
4) The same query is

The remaining of the report is constructed as follows: 1) Section II explains the attempted method - *Repair-PRM* in detail; 2) Section III explains about the experiment details and analyze the results; 3) Section IV summarizes the report and provisions future work.

II. Proposed Method

This section explains the attempted *Repair-PRM* algorithm in detail.

A. Former Feasible Path

The algorithm begins with a former feasible path that can either be cached in a database, or calculated real-time with a shortest path algorithm. The former feasible path indicates that the PRM algorithm used to work on the query with the initial road map, but fails now, which aligns with the specified problem. With the former feasible path, the algorithm can enable lazy-updating where it attempts to adopt as much of the former path as possible. Such design tends to reduce the number of new points sampled and increase the execution efficiency. To discover which portion of the path is feasible, the blocking states of the road map nodes and edges should be recorded. Specifically, when a new obstacle o enters the environment, it only affects the sample nodes around it, and the set of blocked nodes can be denoted as $V_o = \{v|\text{dist}(v, o) \leq r_v + r_o\}$. Meanwhile, all edges related to the blocked nodes are naturally marked as blocked. For other edges that are potentially pointing to blocked nodes, the other end of the edge, which is a "clear" (not blocked) node, cannot reside out of $r_v + r_o + L$, where L specifies the maximum allowed edge length. Using the two limitations, the algorithm can efficiently search for and mark out all blocked nodes and edges by the new obstacle. Later on, as a result, it can utilize such state information to detect the clear segments of the former feasible path.

B. Reconnection with RRT-Star

Basically, the first and last segment of the former feasible path are fetched and kept as a portion of the final path. Then, an alternative starting node and ending node are selected from the first and last path segment, respectively. In Figure 3, the blue triangle and diamond represent the newly selected starting and ending point. Then, a reconnection operation is performed using the *RRT** algorithm [3]. Essentially, RRT* samples for each time a new node connecting to their nearest neighbors, and then uses a steering function to retrieve the final new node. Then, it uses the final new node to find the node on the road map having the minimum aggregated cost from the starting point and forms an edge with that node. Finally, it checks whether nearby road map nodes can be rewired to it. Such algorithm design attempts to ensure that the generated path is as short as possible with the sampled nodes, which serves the same purpose as the shortest path algorithm (e.g., Dijkstra) applied in PRM. Therefore, it indicates possibilities to combine RRT$*$ into the dynamic Repair-PRM design so that the new path provided by the algorithm is kept short at best. Figure 4 shows the output solution of Repair-PRM. The generated path by RRT* is added to the road map so that future queries can reuse it to reach the other sub-graph that was formerly disjoint. Algorithm 1 describes the workflow of Repair-PRM.

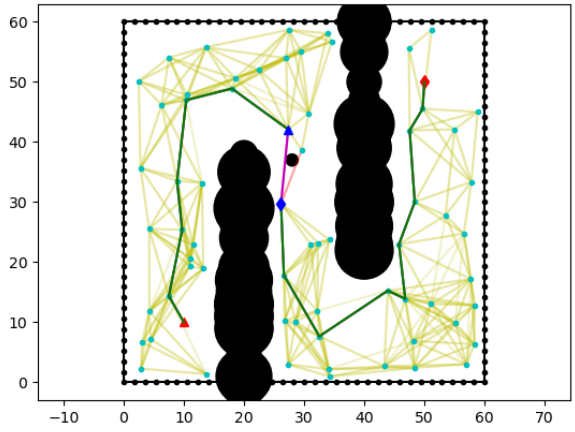

Fig. 3: Node Selection: alternative starting & ending point

C. Node Freedom Evaluation

By default, the alternative starting node is selected as the last node on the first segment path and the alternative ending node is selected as the first node on the last segment path. This follows the intuition to keep as much of the former feasible path as possible. Nevertheless, such selection scheme does not necessarily ensure lower searching cost of the RRT* algorithm. To address this issue, a node metric called the "freedom" of the node is introduced. Essentially, it measures how many directions the robot can move outwards from the node without collision. Nodes with higher freedom values will have higher

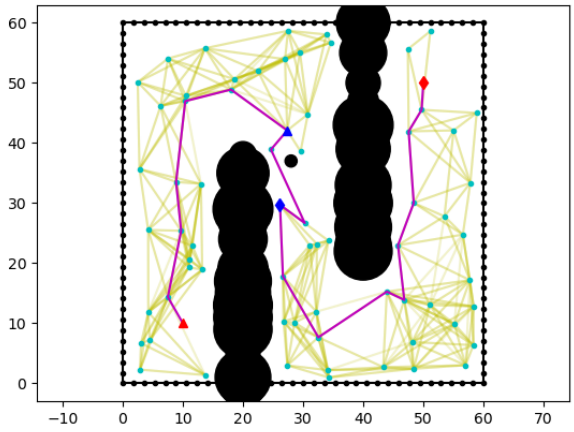

Fig. 4: Repair-PRM: tailor former feasible path, select alternative end points, and reconnect with RRT*.

Algorithm 1 Repair-PRM.

REPAIRPRM(x_s, y_s, x_g, y_g, G)
  $p_0, c_0 \leftarrow$ GETCACHEPATH(x_s, y_s, x_g, y_g)
  if $p_0 = \emptyset$ **then**
    return RRT*(x_s, y_s, x_g, y_g)
  end if
  $p_0^{(1)}, p_0^{(N)} \leftarrow$ first, last segment of p_0
  $x'_s, y'_s \leftarrow$ select node from $p_0^{(1)}$
  $x'_g, y'_g \leftarrow$ select node from $p_0^{(N)}$
  $p', c' \leftarrow$ RRT*(x'_s, y'_s, x'_g, y'_g)
  if $p' = \emptyset$ **then**
    return $\emptyset$
  end if
  $p \leftarrow p_0^{(1)} + p' + p_0^{(N)}$
  $c \leftarrow c_0^{(1)} + c' + c_0^{(N)}$
  return p, c

chances to be able to reach out to the destination. Algorithm 2 shows the detail of the node freedom evaluation.

III. Results and Analysis

This section describes how the experiment is performed and analyzes the results.

A. Experimental Setup

The algorithm is implemented in Python [1]. The initial logic is based on *PythonRobotics* [2] [4]. For more flexible dynamic environment handling, the road map and obstacles have been refactored to node graphs. The experiment is conducted on *Intel(R) Core(TM) i7-11800H* CPU processor and repeated for 20 times.

The test data generation is divided into two steps. In the first step, the map size is randomly sampled ($l \in [20, 100]$) and the number of obstacles is decided based on the robot radius r and the maximum allowed obstacle radius ($0 \leq r_o \leq 2r$), so that in the worst case, the obstacles occupy 20% of the entire map. Then, queries are randomly generated and tested with naive PRM for $n = 3$ times. Feasible queries that pass all tests are stored along with the map.

In the second step, new obstacles are sampled for each query. If PRM that worked before the new obstacle is added fails afterwards, use naive RRT to check whether the query can still retrieve a feasible path. Store the new obstacles if the aforementioned scenario is satisfied. Using such data generation scheme, 100 test cases are generated, each containing one map configuration dictionary, one query configuration dictionary, one pre-generated PRM road map (for "bPRM"), as well as one new obstacle. For each test case, use the road map and the map configuration dictionary to construct the PRM solver, then test whether naive PRM can find a feasible path. If it can, add the new obstacle to the map and re-test all relative algorithms.

The algorithms to be tested are as follows:

1) **bPRM**: PRM before the new obstacle is added. bPRM should be able to output feasible paths according to the design of test cases.
2) **PRM**: naive PRM after the new obstacle is added. The rest of the algorithms are all tested after the new obstacle addition.
3) **RRT**: naive RRT to as a reference to the Repair-PRM.
4) **R-PRM**: Repair-PRM which reconnects the former feasible path using RRT.
5) **RF-PRM**: Repair-PRM-with-Freedom which select alternative end nodes based on node freedom evaluation.
6) **R-PRM***: Repair-PRM-Star which uses RRT* for R-PRM.
7) **RF-PRM***: Repair-PRM-with-Freedom-Star which uses RRT* for RF-PRM.

Algorithm 2 Node Freedom Evaluation.

FREEDOM(x, y, r, λ)
  $d \leftarrow \lambda r$
  $n \leftarrow 0,\ N \leftarrow 0$
  $\theta \leftarrow 0$
  while $\theta \leq 360$ **do**
    $\phi \leftarrow$ RADIAN(θ)
    $x' \leftarrow x + d\cos\phi$
    $y' \leftarrow y + d\sin\phi$
    $n', N' \leftarrow$ COLLISIONCHECK(x, y, x', y')
    $n \leftarrow n + n'$
    $N \leftarrow N + N'$
    $\theta \leftarrow \theta + 36$
  end while
  $v \leftarrow \frac{n}{N}$
  return v

B. Result

Table I shows the evaluation results. It can be inferred that the naive PRM performs poorly upon the addition of

[1]Implementation: https://github.com/WingsUpete/dynamic-prm
[2]PythonRobotics: PRM implementation

the new obstacle, which is as expected due to the test case generation logic. Meanwhile, the proposed Repair-PRM family members all manage to repair the disconnectivity caused by the new obstacle, with the highest feasible path retrieval rate as 93%. Comparing to the traditional single-step RRT, the output paths of R-PRM and RF-PRM have higher costs, while R-PRM* and RF-PRM* both provide paths with lower costs. This proves the shortest-path keeping heuristic of choosing RRT* over RRT for reconnection. Surprisingly, for planning time, R-PRM* even runs faster than RRT. This is probably due to the generated lower-cost path reducing the number of sampling steps. Finally, it can be noticed that the RF-PRM family members both outperform the R-PRM family members on the feasible path retrieval rate, confirming that the node freedom evaluation helps selecting the best node pair for path planning.

Figure 5 demonstrates the results of one of the test cases. In the test case, naive PRM fails to handle the query after the addition of the new obstacle (blocking the red edges). However, R-PRM* manages to re-route the former feasible path from bPRM and achieves a feasible solution. Compared to RRT which detours largely on its solution, R-PRM* is able to keep the sampled path short at best.

Algorithm	Feasible%	Path Cost	Planning Time	#New Nodes
bPRM	62	-	-	-
PRM	2	-	-	-
RRT	90	27.4757	0.0192	-
R-PRM	90	28.3335	**0.0169**	4.64
RF-PRM	92	27.6495	0.0208	4.70
R-PRM*	92	26.6384	0.0189	4.40
RF-PRM*	**93**	**26.5858**	0.0242	4.51

TABLE I: Evaluation delta results compared to bPRM.

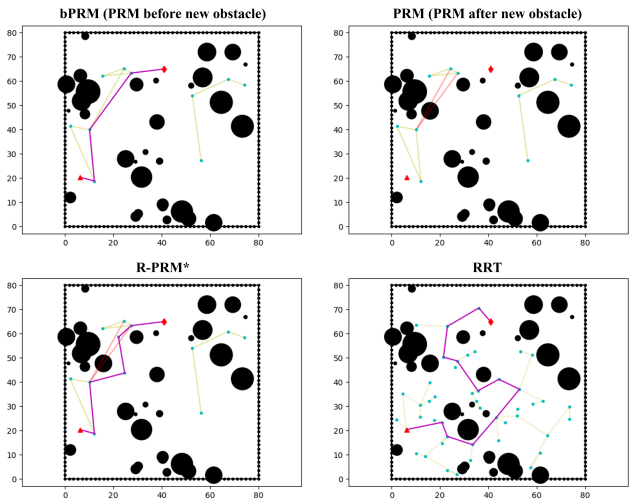

Fig. 5: Results on one of the test cases.

IV. Conclusion

In this report, a modified PRM algorithm, namely Repair-PRM is attempted to maintain the road map in dynamic 2D environment. When new obstacles enter the map and block existing nodes and edges along the former feasible path of a query, Repair-PRM tries to retain the segments of the former feasible path that are still collision-free, select two alternative end nodes on the path segments, and utilizes RRT* to reconnect the path. Experiment results show that Repair-PRM with RRT* helps keep the new generated path as short as possible, and Repair-PRM with node freedom evaluation helps increase the possibility of finding new feasible paths. Nevertheless, the generated dataset commonly includes road maps of small sizes, due to the constraints of automatic random generations. In the future, large-scale maps with moving obstacles should also be further studied in order to increase the robustness of the Repair-PRM algorithm.

References

[1] L. 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*, vol. 12, no. 4, pp. 566–580, 1996.

[2] S. M. LaValle, "Rapidly-exploring random trees : a new tool for path planning," *The annual research report*, 1998.

[3] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," 2011.

[4] A. Sakai, D. Ingram, J. Dinius, K. Chawla, A. Raffin, and A. Paques, "Pythonrobotics: a python code collection of robotics algorithms," 2018.