Low dispersion probabilistic roadmap for dynamic environments

ENPM661 - Final Project

Gokul Hari
M.Eng. Robotics
University of Maryland
hgokul@umd.edu

Aswath Muthuselvam
M.Eng. Robotics
University of Maryland
aswathm@umd.edu

Shailesh Pranav
M.Eng. Robotics
University of Maryland
d@umd.edu

Abstract

Path Planning in real time systems have often adopted sampling based planners over action based planners due to the massive difference in the time complexity between two classes of planners. These planners with random sampling techniques, often assumes a known-static obstacle configuration. This is ineffective for real world scenarios involving obstacle spaces of unknown environments. To deal with such dynamic obstacle environments, we worked on a fast auto-adjusting probabilistic roadmap algorithm. The generated samples are stored in 2D array-like data structure, used as a lookup table to check if there are obstacles encountered. If a sample lies within the obstacle, our algorithm adjusts the collision points so that we have samples free from the obstacle space. The roadmap adjusts itself to have samples from the obstacle-free space whenever a change in the environment is observed. This report contains our experimentation and simulation of our implementation of this self-adjusting roadmap algorithm. Our codebase is available in this github repository- ***https://github.com/shaileshpranav/LD-PRM-for-dynamic-environments***

1. Introduction

Sampling based planners were developed to be faster and they do not require the complete detailed map of the environment. This comes at the cost of optimality as they involve checking if a randomly selected configuration of the robot is in obstacle space or not. Local planners using an average resolution interpolation between two configurations can be utilised to decide if configurations can be connected without collision or not.

Probabilistic roadmaps come under the class of multi query algorithms. It discovers collision free space by selecting random configurations of robot's state, making them the verrtices of a graph and connects the edges if the path between them is collision free. Since we want to address planning with dynamic changing environment, we employ a grid based sample classification process to classify whether the state is in obstacle space or free space, and therefore we do not need to reconstruct the entire planning graph.

In a basic sampling based planner, N configurations/states are sampled in the configuration space C, and retain those states in C_{free} (obstacle free configuration space) which are commonly referred to as 'milestones'. Next, a roadmap edge is constructed by connecting any two milestones A and B if the line segment or edge AB is completely in C_{free}. This is repeated until the added edges result in a path that connects the start node S and Goal node G. If the added edges do not result in a path, it either because the planner did not sample enough milestones or no path exists.

These planners are well suited for high-dimensional configuration spaces. Unlike combinatorial algorithms, their running time is not exponentially dependent (explicitly) on the dimension of configuration space C. They are also (generally) substantially easier to implement. They are probabilistically complete, meaning the probability that they will produce a solution as time reaches infinity.

The implementation of the paper was solely done by simulation and no sensor errors or characteristic physical error were taken into consideration. In this work, we prioritize more on the fast execution and accuracy than the optimality. Our work is easily extendible to higher dimensional spaces and real world implementation due to the advantages on latency and runtime complexity that our work has over traditional planners.

1.1. Problem statement

Sampling based planners assume that the environment is known and pre-defined. However, in real world, the operational environment of a mobile robot is rarely known, and the planning with respect to the state of a dynamic obstacle is challenging. We address this problem using a fast probabilistic roadmap algorithm, that uses a low dispersion sampling strategy. We need to approach this problem in a

multi-query fashion which helps us to selectively re-route the plan when an obstacle intervenes the preplanned path.

2. Background

In this section we will discuss techniques that are fundamental for the derivation of this work.

2.1. Probabilistic Road maps (PRM)

Since we are addressing planning in dynamic environments, a multi-query based sampling method is a reasonable choice to incorporate fast re-planning when the obstacle space modifies over time. Probabilistic Roadmaps are well known multi-query sampling based method for planning. This planner can be seen as it comprises of two components main. An obstacle-check component that determines if the selected pair of samples lie on the free space. The local-planner component detemines if the two samples have a valid path that lies in the free space. We try to make modifications to these two major components of the PRM method and design our self-adjusting road-map planner.

2.2. PRM Extensions

Conventional PRM and its extensions are not able to deal with unknown and changing environments because they require to learn the configuration space before the navigation starts. Few extensions of PRM have been proposed to deal with planning in unknown dynamic evironments. Li et al [1] introduced an incremental learning approach as a PRM based planner has been proposed that provides a framework to managing either roadmap under changing settings. When the changes in the environment are noticed, the road-map nodes in the obstacle space and colliding paths are discarded. This leads to the emergence of separated roadmaps, or forests. However, this planner results in increasing the computational costs. Khaksar et al [2] introduced a real-time PRM based on a a novel sampling strategy that ignores the obstacles in the configuration space while generating a roadmap and encodes the mapping from workspace cells to nodes and arcs in the graph. In his subsequent work [3], when the environment changes, this mapping is used to make appropriate modifications to the graph and paths can be planned over the modified roadmap. However, since this planner ignores every obstacle in the environment, when dealing with crowded environments or dense maps, the processing time will increase substantially in crowded environments.

3. Methodology

3.1. Sampling Strategy - Low-Dispersion PRM

Sampling queries in multi-query based planners can have interesting effects based on the sampling strategy. Probabilistic Roadmaps utilize a graph data structure with nodes

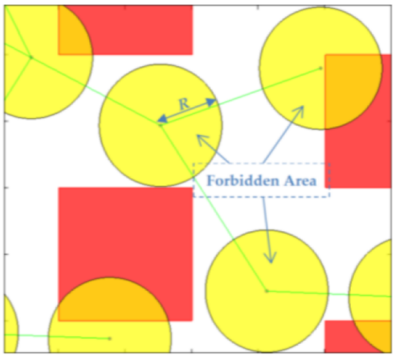

Figure 1. Low Dispersion samples.

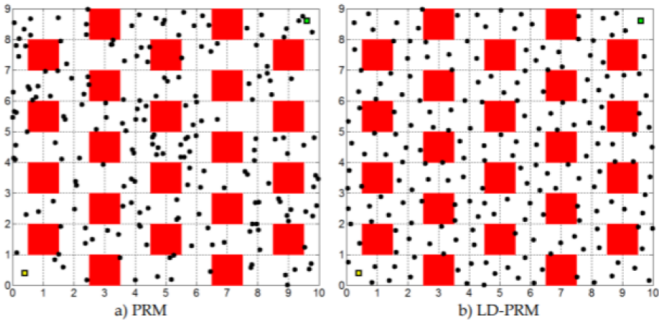

Figure 2. Regular PRM vs Low Dispersion PRM.

as states in configuration space. The edges between the nodes exist if there is a collision free path between two states. A typical PRM algorithm has two phases: a learning phase and a querying phase.

- In the learning phase, the roadmap is created. Collision-free configurations, called 'samples', are generated randomly which form the nodes of the roadmap (a subset of all the points in the configuration space).
- In the query phase, a number of iterations are conducted based on the number of sampled points and an attempt is made to connect these points based on a distance criteria to find its nearest neighbours, thus adding edges to the roadmap.

Graph search strategies are utilised to solve a query by finding the path, given the start and goal states are added to the roadmap. The efficiency of the algorithm depends on how well the roadmap can capture the connectivity of the configuration space. In this work, n random number of sample configurations are generated and a connectivity of atmost x nodes in the graph is established to define the entire graph structure.

In Low-dispersion PRM algorithm, learning phase involves a huge modification compared to the traditional

PRMs. The learning phase selects samples that need to fulfill an additional criterion in order to be included in the roadmap. This criterion forbids the samples to be close to each other more than a predefined radius. This is instrumental in reducing the number of queries to be taken into consideration in th probabilistic road map. The key idea behind LD-PRM is that if we want to fit non-overlapping circles/ n-dimensional spheres inside the configuration space, the aggregate volume of these balls should be smaller than the volume of the free space.

Based on the Lebesgue measure (volume) of the collision-free configuration space, this radius can be defined as follows :

$$R_S(n) = [L(Q_{free})(n - \lambda)/(\pi n^2)]^{1/2} \quad (1)$$

where, Q_{free} is free configuration space. n is number of samples in the road-map. λ is a positive scaling factor.

We utilise this low-dispersion sampling strategy to optimize the number of query samples taken for the roadmap, to achieve more coarser roadmap. The visualization of radius chosen due to lebesque measure for the LD-PRM is shown in this figure 1. The result of the low-dispersion sampling schema is shown in this figure 2

The algorithm is presented in 1.

Algorithm 1 Low Dispersion PRM

G(V,E) $\leftarrow \phi$
R $\leftarrow$ radius of forbidden space
S $\leftarrow$ { sample(1), sample(2).. sample(N) }
for i **in** 0 *to* N **do**
  $sample_c = S[i]$
  $dist_{map} = \phi$
  if $sample_c$ **not in** G(V, E) **then**
    V $\leftarrow sample_c$
  end if
  for j **in** 0 *to* N **do**
    d = $\|sample_c - S[i]\|_2^2$
    if $S[j]$ *is not* $sample_c$ **and** $d \leq R$ **then**
      $dist_{map} \leftarrow \{d, S[j]\}$
    end if
  end for
  $sort(dist_{map} : d)$
  for $(d, node)$ **in** $dist_{map}$ **do**
    E $\leftarrow$ node
    if |E| $>$ x **then** break
    end if
  end for
end for
return G

We start with graph generation which is used by the global A* planner to find the optimal path. The following parameters are taken into consideration for planning.

- Number of nodes (V) to be generated 'x'.
- Minimum and maximum distance between sample nodes (V) in the graph.
- Maximum number of connections (E) that one can have.
- Nodes (V) and Edges (E) should not lie or intersect in the obstacle space.

3.2. Dynamic re-Planning.

Once the A* planner finds the optimal path in the generated road-map, the robot navigates through the path by continuously monitoring/polling the environment checking for obstacles in the configuration space. The procedure behind Dynamic Planning is as follows.

- The planner checks whether the obstacle intersects the rest of the optimal path that the robot needs to follow.
- If there is an obstruction in path due to new obstacle, perform the following roadmap adjusting protocol.
  - i) find the nodes and the edges in the remaining path that are in or intersecting with the obstacle space.
  - ii) The nodes in the path and the rest of the nodes in the sample space that lie in the obstacle's region are shifted to come outside in free space. This is followed by the re-adjusting the edges/connection of the existing graph to accommodate the newly shifted points as well as remove any edges that intersect the obstacle space. This results in the adjusted road-map.
- The robot is navigated to follow the re-configured path from the point where it observed the new obstacle.
- If the new obstacle does not obstruct the rest of the path, the currently planned path is followed.

By mapping each node in road map to a occupancy grid's cell (working as a 2D lookup table), we can infer if node is occupied by obstacle or retrieve the node during run-time. This occupancy grid helps optimizing computational complexity.

3.3. Key features

- Includes a safety feature in the planner during navigation, acting as a dynamic obstacle avoidance planning scheme.
- The occupied nodes are refactored to free space and are not removed from the set of samples. Thus, the size of graph is maintained. This can help optimizing the set data-structure with faster programs.

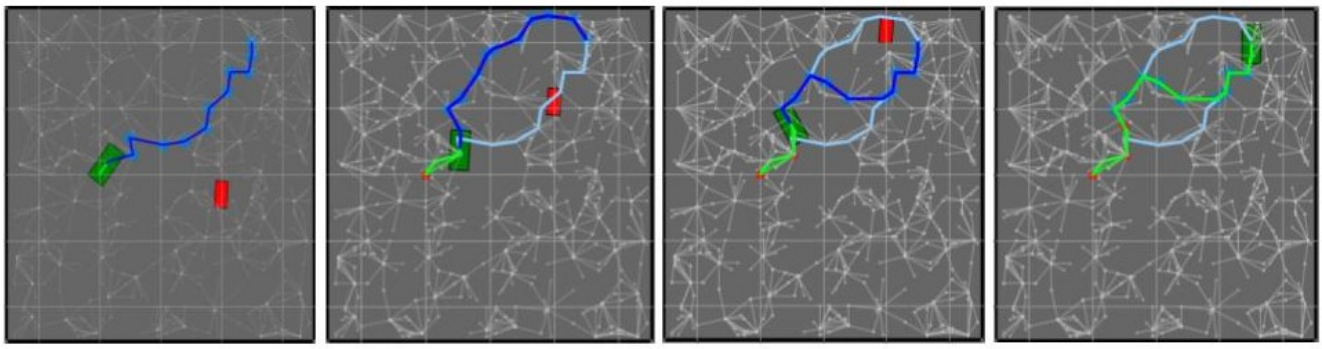

Figure 3. a) Robot path decided by LD-PRM, b) c) re-planned due to interefering moving obstacle (in red) d) robot finally reaches the goal

	test-case-1		test-case-2		test-case-3	
Configuration	time to plan	time to re-adjust	time to plan	time to re-adjust	time to plan	time to re-adjust
no obstacles	1.16	0	2.01	0	0.92	0
static	2.1	0	2.65	0	1.78	0
1 moving obstacle	9.4	514	8.8	429	6.3	326
4 moving obstacles	16.1	923	14.2	890	18.6	789

Table 1. Test Run results on latency (in ms)

n samples	Avg time to plan	Avg time to re-adjust
100	5.2	467
200	9.3	690
500	16.1	923
1000	22.4	1301

Table 2. Latency (in ms) vs Number of Samples for Roadmap

- This algorithm stores samples in a occupancy grid structure based on their position in the corresponding configuration space which makes it computationally feasible to perform collision check.
- Only the occupied grid cells by the dynamic obstacles and their correspoding sample states will be checked. Consequently, the roadmap responds to the changes in the environment.

4. Experiments and Results

We implemented the low-dispersion sampling strategy and the dynamic re-planner in python. We executed our experiments in a Intel i7 10750H CPU with 6 cores. For this work we performed the following experimentation to evaluate the performance of the model. We considered 3 different configuration spaces with 10 test cases of start-goal for each C-space . We report the inference time to obtain a planned path and the time to re adjust in seconds. These results are shown in Table 1. **The simulation results can be seen in this video**

We also observed that as we increase the density of the sampling (by modifying the low -dispersion sampler) result in longer time taken to re-plan and which affected the overall performance of Dynamic re-planner. There was an observed increase in optimality of results but at a cost of latency. These results are shown in Table 2

5. Conclusion

In this project we implemented, and experimentally evaluated a probabilistic road map variant for planning in dynamic environments. Our self-adjusting road-map, utilised a low dispersion sampling strategy for obtaining multiple queries at lower latency. In future work, we want to implement this codebase in C++ and deploy in a robot and create a turtlebot package repository to address real-time planning problems.

References

[1] Tsai-Yen Li and Yang-Chuan Shie. An incremental learning approach to motion planning with roadmap management. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, volume 4, pages 3411–3416 vol.4, 2002. 2

[2] Weria Khaksar, Tang Sai Hong, Mansoor Khaksar, and Omid Motlagh. A low dispersion probabilistic roadmaps (ld-prm) algorithm for fast and efficient

sampling-based motion planning. *International Journal of Advanced Robotic Systems*, 10(11):397, 2013. 2

[3] Weria Khaksar, Md Zia Uddin, and Jim Torresen. Self-adjusting roadmaps: A fast sampling-based path planning algorithm for navigation in unknown environments. In *2018 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 1094–1101. IEEE, 2018. 2