ENPM661 Project 5: Lazy PRM Path Planning Algorithm

Rey Roque-Perez[#1], Dustin Hartnett[*2]

University of Maryland - Robotics Engineering

[1]reyroque@umd.edu

[2]dhartnet@umd.edu

Abstract

This paper describes this group's approach to implement the Lazy PRM sampling based path planning algorithm. The project started as a regular Lazy PRM implementation based on the original R. Bohlin and L. E. Kavraki algorithm proposal and ended up implementing parts of M. Ramirez, D. Selvaratnam, and C. Manzie revisited algorithm proposal. Lazy PRM is a novel way of implementing the PRM algorithm by not checking for collisions while sampling and connecting nodes to create the roadmap. By avoiding collision checks until the moment of path generation, the amount of collision checks and computations can be minimized. This method proves to be very effective at reducing collision checks. By applying some of the revisited approach's method, the calculations performed while connecting the roadmap can also be reduced. The graph generated by the Lazy PRM algorithm was traversed using A and the output was connected and used to run a turtlebot simulation in Gazebo.*

I. Introduction

This paper addresses an assignment for a University of Maryland robotics engineering course on path planning. The task was to implement a cutting edge, sampling-based path planning algorithm to navigate a robot through a defined map with obstacles. The paper will discuss how the lazy probabilistic roadmap (PRM) sampling-based path planning technique was implemented to navigate a turtlebot robot through a map simulated in a Gazebo environment. The motivation behind selecting the Lazy PRM method for this problem lies in our desire to implement a PRM path planning variation. Traditional PRM methods of path planning are very effective, but incredibly computation intensive. The Lazy PRM sampling-based path planning algorithm offers the effectiveness of a traditional PRM method, but returns a path in a less computationally intensive way by limiting the number of collision checks required.

II. Background and related work

The lazy PRM path planning algorithm was first published in IEEE in April 2000 by R. Bohlin and L. Kavraki. The paper introduces the lazy PRM method as a way to limit the number of collision checks done when computing a path. The paper proposes a method where collision checks for nodes only occur after the PRM has been created, and a path planning technique, such as A*, produces a path from the start node to the goal node. This ultimately reduces the number of collision checks required, and hence offers a less computationally intensive path planning algorithm. Since 2000, the lazy PRM path planning algorithm has continued to appear in academic publications, although researchers publishing on IEEE and at institutions such as Rensselaer Polytechnic Institute, Stanford University, Campus Cuernavaca, , KTH Royal Institute of Technology, and University of Melbourne have introduced further variations to this algorithm. Some of these variations to the lazy PRM method include use for protein folding analysis, reactive path planning, asymptotically-optimal motion planning, and bi-directional motion planning ([1], [2], [3], [4], [5], [6]).

Traditional PRM path planning algorithms continue to be popular due to their robust and effective path planning method, however, the lazy PRM algorithm

and variations of lazy PRM have proved to be an efficient PRM variation for applications requiring a less computationally-intensive but still effective sampling-based path planning algorithm.

III. Method

I.Description

The lazy PRM algorithm we implemented in this project is a three step algorithm: PRM building, path searching, and collision checking (Fig. 1, 2). The goal of this method is to efficiently find the shortest path to a goal given the PRM created.

The PRM building step of this algorithm is intended to create a roadmap through a given environment. In our case, this step accounts for the size of the map, the desired resolution of nodes, and the start and goal node. The roadmap is created by uniformly sampling the environment, defining nodes, and connecting neighboring nodes, including the start and goal nodes, to create a roadmap.

The path searching and collision checking steps of this algorithm operate together. The path searching step involves using A* to search the PRM for a path to the goal, and the collision checking step involves checking the A* output path for collision, and subsequently removing nodes that are in collision and triggering the A* path searching step again if collision exists. The following sections will describe each of these steps in further detail.

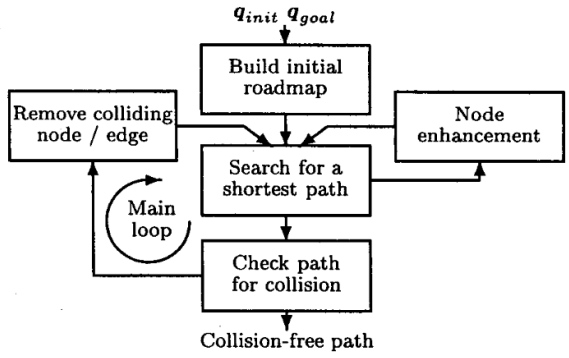

Fig. 1 Block diagram illustrating method of Lazy PRM [1]

Lazy PRM Algorithm

Input: $node_{start}$, $node_{goal}$, *C*, *Obstacle*
Output: *Solution Path*

1: $q_{uniform} \leftarrow$ *SampleMap(C)*
2: PRM ← ConnectNodes($q_{uniform}$, $node_{start}$, $node_{goal}$)
3: **while** *Collision* is *True*
4: | *path* ← *A*(PRM)*
5: | **foreach** *node* ϵ *path*
6: | | *Collision* is *False*
7: | | **if** *node* in *Obstacle*
8: | | | *Collision* is *True*
9: | | | *Delete node from PRM*
10: | | | ***EXIT*** **foreach**
11: | | **end**
12: | **end**
13: end
14: **return** *SolutionPath*
15: *TurtlebotMovement* ← *SendPath(SolutionPath)*

Fig. 2 Pseudocode for Lazy PRM

II. Building the Probabilistic Roadmap

The PRM is populated by comparing all sampled nodes. These nodes are then compared using a simple Euclidean distance equation to see which ones fall within an acceptable distance from any one node. All nodes that fall in this threshold will then be added to a possible neighbor list to connect them. All possible neighbors for any one node will be compared against a list of possible movements (Fig. 3). This list is created based on kinematics constraints applied to previous nodes. A node contains positional data (x,y) and orientation data (Θ), which are associated with the location and orientation of the robot at that node within the PRM. To determine the possible new nodes that can be reached and added to the PRM from the current node being explored, kinematic constraints of the differential drive robot are taken into account.

This algorithm has three different wheel speed options available (0, 1 ,2) to the robot for movement. For our implementation of this algorithm, we chose wheel speed options of 40 and 80 rotations per minute (RPM). The action set, being a combination of left and right wheel speeds (l ,r), describes the eight possible ways the robot can maneuver from a node.

$$(\omega_l , \omega_r) = [(0 , \omega_1), (\omega_1 , 0), (\omega_1 , \omega_1), (0 , \omega_2), (\omega_2 , 0), (\omega_2 , \omega_2), (\omega_1 , \omega_2), (\omega_2 , \omega_1)]$$

Using the robot's wheel radius (r) of 3.3 centimeters and wheel spacing, or robot diameter, (L) of 28.7 centimeters, the velocity of the robot in the x- and y-directions (v_x , v_y), as well as angular velocity (ω_θ) can be determined,

$$time = [0: dt: 1]\ seconds$$
$$x_{new} = x_{old} + v_x * dt$$
$$y_{new} = y_{old} + v_y * dt$$
$$\Theta_{new} = \Theta_{old} + v_\theta * dt$$

Performing these calculations for each possible action set from the current node being examined will result in eight new reachable nodes to be added to the workspace.

ConnectNodes

Input: $q_{uniform}$, $node_{start}$, $node_{goal}$
Output: *Connected PRM Graph*

```
1: foreach node ∈ q_uniform
2: |   foreach node' ∈ q_uniform
3: |   |   distance ← GetDistance(node, node')
4: |   |   if threshold_min < distance < threshold_max
5: |   |   |   reachable ← moves(node)
6: |   |   |   if node' in reachable
7: |   |   |   |   Add node' to PRM(node)
8: end
9: return PRM
```

Fig 3: Node Connection Pseudocode

$$v_x = \frac{r}{2} * (\omega_r + \omega_l) * cos(\Theta)$$
$$v_y = \frac{r}{2} * (\omega_r + \omega_l) * sin(\Theta)$$
$$\omega_\theta = \frac{r}{L} * (\omega_r + \omega_l)$$

In this problem, each action set is defined as motion for one second, with a time step for calculation purposes of 0.1 seconds (dt). From this, the eight possible nodes that are reachable in the PRM from the current node being examined can be calculated.

These eight possible nodes are then compared to the list of possible neighbors. All possible neighbors that are reachable according to the move functions will be added to the connected graph for the PRM. It is important to note this process will run once per sampled node and assumes no collisions.

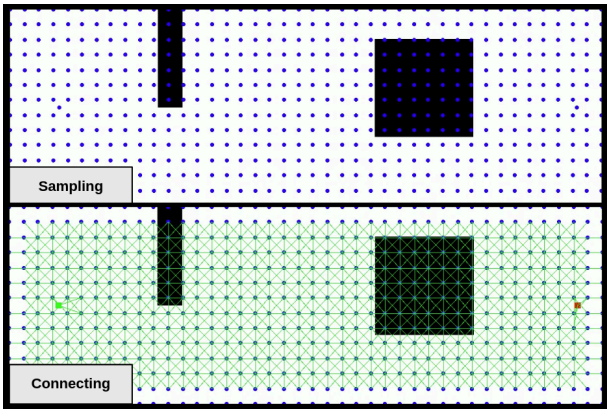

Fig. 4 Example of sampling and connecting nodes in PRM

III. Path Searching

Once the PRM is formed, the next step is to search for the shortest available path to the goal from the starting node. We use an A* algorithm to find the shortest path. The metric for determining the shortest path is path distance, so the path with the least cost is ultimately the shortest path available.

If no path is found, the lazy PRM algorithm will return failure. In future work, a node enhancement step could be added to increase the node density

around regions where the A* algorithm failed. This would allow for more searches to be performed, ideally resulting in a discovered path to the goal node [1].

IV. Collison Checks

Once the A* algorithm has found the best path to the goal node, we must check to see if there is any collision with obstacles. Obstacle collision means a node in the path resides within a space identified as an obstacle, or within the defined buffer region between the robot and obstacles. The goal of this method is to check nodes for collision, but ensure we aren't checking unnecessary nodes that are not part of the path found by the A* algorithm.

To check a path for nodes that are in collision with obstacles, we begin searching from the beginning of the path towards the end, node by node. For each node, we compare its x- and y-coordinates alongside the known regions of obstacles and buffers. If a node is within an obstacle or the buffer zone, the collision search stops, the node is removed from the PRM, and the A* algorithm is run again (Fig. 5).

Eventually, all nodes that are in collision with an obstacle or the buffer zone that lay along the desired path towards the goal will be removed, and the A* algorithm will output a collision-free path if there is one.

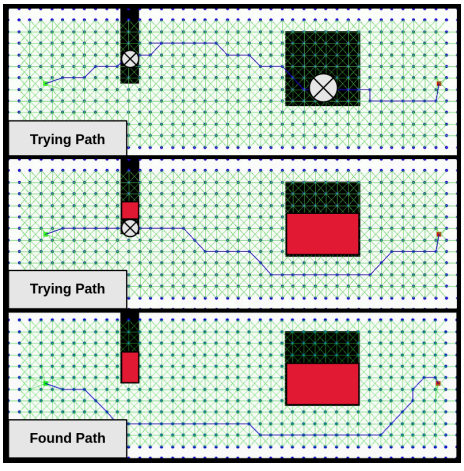

Fig. 5 Example of running A* algorithm and checking nodes for obstruction until an unobstructed path is found

IV. Simulation

The lazy PRM algorithm used in this problem case is intended to be used to navigate a differential drive Turtlebot robot through a simulated Gazebo environment. The gazebo environment is a 600 x 200 centimeter map containing two obstacles (Fig. 6). The obstacles were specifically designed to test two different capabilities of the lazy PRM algorithm. The first obstacle is a simple obstacle, only requiring the robot to detect the obstacle space and navigate around it before continuing. The second obstacle is significantly larger and forces the robot into a channel between the obstacle and one of the map's boundary walls, testing the lazy PRM algorithm's ability to detect and maneuver around an extended obstacle space in a confined channel. In this simulation setup, the starting location of the Turtlebot is fixed for every iteration, but the goal location can be defined by a user.

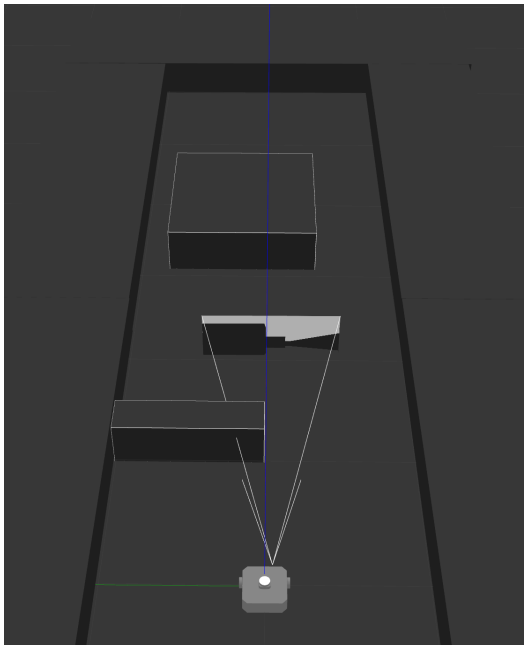

Fig. 6 Gazebo simulation environment showing obstacles

When the lazy PRM algorithm outputs a path from a start to goal point, a separate python script intakes the output path and converts each node in the path's associated action set of left and right wheel velocities (ω_l, ω_r), with a wheel radius (r) and wheel spacing, or robot diameter, (L), to linear and angular velocity commands for the Turtlebot robot,

$$v_{linear} = \frac{r}{2} * (\omega_r + \omega_l)$$
$$\omega_{angular} = \frac{r}{L} * (\omega_r - \omega_l)$$

These commands are then sent to the Turtlebot's velocity controllers for one second each while the robot operates in the Gazebo environment. Correct navigation in the Gazebo environment is double checked by the user by comparing the robot's observed motion with the graphic output produced by the lazy PRM algorithm.

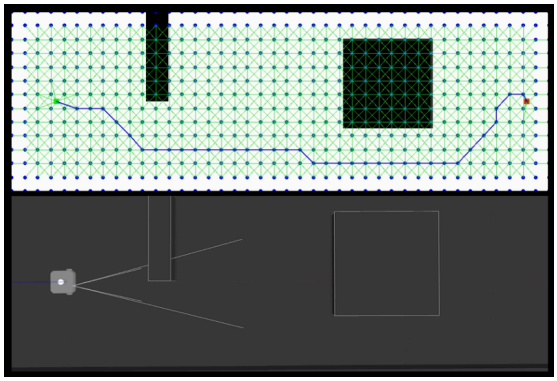

Fig. 7 Example of Lazy PRM algorithm output that will guide the Turtlebot robot through the Gazebo environment

V. Results

The lazy PRM algorithm we implemented performed very well. When checking for a path across the entire map, from end to end, and around both obstacles, we found the lazy PRM algorithm was able to sample the map in less than 0.01 seconds and connect neighboring nodes to make a PRM in 15 seconds. Iteratively running the A* algorithm and collision checking algorithms output a path to the goal node in 0.65 seconds (Fig. 8). User-defined goal nodes closer to the fixed start node required less time for the algorithm to output a solution path. These computation times are significantly faster than the A* algorithm implemented in Project 3, which took 85 seconds to run and solve for a path, although the map was slightly different.

```
LazyPRM.py
Sampled points in:  0.008001327514648438 s
Connected nodes in:  15.524304151535034 s
Path Found in:  0.6491560935974121 s
```

Fig. 8 Times for the Lazy PRM path planning algorithm to find a path to reach a goal node on the opposite side of the map

When the output path is ingested by the Gazebo script, the Turtlebot robot is expected to navigate through the Gazebo environment. Unfortunately, not every path found succeeded in guiding the robot to its goal node. For goal nodes near the start node, the robot performed as expected. When the goal node resided far from the start node, errors in the Gazebo environment seemed to lead the robot off its course. The lazy PRM algorithm and the Gazebo script that commands the robot operate in "real time", meaning real time samples are used to calculate robot motion and path planning. Unfortunately, the Gazebo environment we used operates with "simulation time", which differed from real time. Over the course of a long path, the robot accrued errors that eventually built up enough to drive it off course.

V. Discussion

The Lazy PRM algorithm implemented ended up being different than the one found in the referenced paper: "Lazy probabilistic roadmaps revisited". The referenced paper proposes keeping the same basic Lazy PRM algorithm but adjusting the node connection phase. A Batch-Cut (BC) algorithm was suggested to solve the node generation problem rather than calculating kinematic constraints for each sampled node. This implementation was meant to reduce the computational load of calculating kinematic constraints for each possible movement. The implementation of the batch-cut method did not produce consistent results in a reasonably fast runtime (runtime over 10 minutes). The group decided to reduce the amount of complex kinematic calculations by setting a threshold for neighbor nodes based on the euclidean distance between nodes. This way only nodes that fell within this threshold were

connected using the full kinematic constraints thus reducing the runtime.

An issue faced when implementing the lazy PRM algorithm was fitting the kinematic constraints of the robot to the PRM that was made. The Turtlebot is defined in this case as having eight possible actions at each node, producing eight more nodes. The actions consist of combinations of left and right wheel speeds allowed to operate for one second at a time. This produces varying straight and curved paths resulting in a series of new locations and orientations the robot can get to in the map. Because the PRM is uniformly sampled, though, resolution constraints limit the algorithm's ability to match each action with a unique, sampled node on the PRM. Increasing the resolution of the uniformly sampled area increases the chances each action set results in a new position in the map that correlates to a sampled node in the PRM. Decreasing the resolution of the uniformly sampled area increases the chance that multiple actions end up rounded to the same sampled node on the PRM, which over time will result in increasing error as the robot's calculated action sets don't align with the nodes the lazy PRM and A* algorithms used for path planning.

Another issue faced early on, that was eventually solved, was navigating through the channel created by the second obstacle. The lazy PRM sampling-based path planning algorithm variation we implemented performs A* when trying to search the PRM for a path to the goal. Because of this, the preferred path by this algorithm is always the most direct route between any starting node and the goal, which is often a straight line. When a node in the A*-derived path is found to be in an obstacle or in collision, it is removed or marked as unavailable and then A* is re-run. In the created channel, this became an issue, resulting in snake-like movement. The desired path often resulted in the robot entering the channel moving parallel to the length of the channel, then trying to turn left and move towards the goal before readjusting due to the nearby obstacle and turning back towards the side of the map, away from the obstacle. Without a large buffer defined to limit the acceptable distance between the robot and the obstacles, the action set, or kinematic constraints, assigned to the robot allowed for sharp, slow turns that created reachable nodes that were not in collision with the second obstacle. This allowed the robot to turn towards those nodes. However, the next node that would be in a straight line between the robot's current position and the goal node, resided in the obstacle. The robot was then forced to turn back away from the obstacle, creating a snake-like movement pattern. This resulted in accrued errors in Gazebo as it is not a perfect simulation. The solution to this problem was increasing the buffer value, limiting how close the robot can get to the obstacles. This limited the robot's ability to turn towards the obstacle, forcing it into a nearly straight path through the channel. Ideally, a path smoothing algorithm would nullify these effects without the need for a constricting buffer, however, path smoothing techniques were not applied in this instance.

VI. Conclusions

The team successfully implemented the Lazy PRM algorithm for a non-holonomic differential drive system robot in Gazebo. Based on the results, the algorithm seems to best fit situations where the same map will be traversed multiple times. In this circumstance the PRM only has to be calculated once and the time taken to find the actual path will be negligible compared to just running A*. In addition, due to the nature of removing colliding nodes while generating the paths, each subsequent path generation will get faster until all obstacles are eliminated from the PRM graph. The resulting path might not be optimal, but this can be alleviated using smoothing techniques. In conclusion, while the path generated by Lazy PRM might not be optimal, it makes up for it by being significantly faster to run than an algorithm like A* (around 531% faster in this case).

Acknowledgment

The authors would like to thank the University of Maryland, Dr. Reza Monfaredi, Shantanu Parab, and Saksham Verma for their assistance in understanding the material throughout our course and for supporting our efforts.

References

[1] R. Bohlin and L. E. Kavraki, "*Path planning using lazy PRM," Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation, vol 1, pp. 521-528, 2000.*

[2] M. Ramirez, D. Selvaratnam, and C. Manzie, *"Lazy probabilistic roadmaps revisited," arXiv, Sep. 2022*.

[3] Hauser, Kris. (2015). Lazy collision checking in asymptotically-optimal motion planning. Proceedings - IEEE International Conference on Robotics and Automation. 2015. 2951-2957. 10.1109/ICRA.2015.7139603.

[4] Y. A. Girdhar, "EFFICIENT SAMPLING OF PROTEIN FOLDING FUNNELS USING HMMSTR AND PATHWAY GENERATION USING PROBABILISTIC ROADMAPS," M. Computer Science thesis, Rensselar Polytechnic Institute, Troy, New York, Apr. 2005.

[5] Sanchez-Ante, Gildardo & Latombe, Jean-Claude. (2001). A Single-Query Bi-Directional Probabilistic Roadmap Planner with Lazy Collision Checking. International Journal of Robotic Research - IJRR. 403-417.

[6] K. Hauser, "Lazy collision checking in asymptotically-optimal motion planning," 2015 IEEE International Conference on Robotics and Automation (ICRA), Seattle, WA, USA, 2015, pp. 2951-2957, doi: 10.1109/ICRA.2015.7139603