Lab 6 Report: Path Planning

Team 3

Jinger Chong
Nina Gerszberg
Ethan Hammons
Juan Rached
David von Wrangel

6.141 Robotics: Science and Systems

April 16, 2022

1 Introduction

Juan Rached

With our localization and our parking controller implemented, Alfredo the robot was almost ready for the final challenge. Now that our racecar can determine its position in space as it follows a trajectory, we implement several path planning procedures, including both search-based and sampling-based methods. In this lab, we discuss Probabilistic Roadmaps, Bidirectional Rapidly-exploring Random Trees, and A*. Additionally, we explore potential applications of Motion Planning in a Graph of Convex Sets.

2 Search-based Planning

2.1 A*

Nina Gerszberg

A* works similar to Dijkstra's algorithm with the added element of a heuristic. A* explores the graph, prioritizing more likely nodes based on a given heuristic. For our purposes, the heuristic is Euclidean distance since we are trying to find the shortest path. A* goes though the graph and finds the shortest path to each node. It accomplishes this by going through all of the nodes in order of the heuristic, noting the cost of getting to each node, until it finds the end node its looking for or runs out of nodes to look through. It then returns a list of nodes

representing the shortest path between the start point and end point.

We tested our A* algorithm on a graph made directly from the OccupancyGrid. We converted the Occupancy Grid into a 2D numpy array where each index of the graph represents a node in the graph for A* to search through. When A* queries a given node's neighbors, it receives all of that node's immediate neighbors as well as its neighbors along the diagonal. To test this, a user can open RVIZ and pick a start point and an end point and our code will draw the shortest path it finds between them.

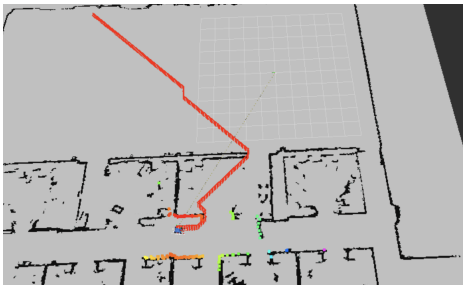

Figure 1. Sample of using A* to create the shortest path between two points using OccupancyGrid Graph

In addition, we also tested our A* algorithm on an OccupancyGrid generated from the Stata basement map. A* was able to return a path that traverses 8 provided milestones as shown in Fig. 2.

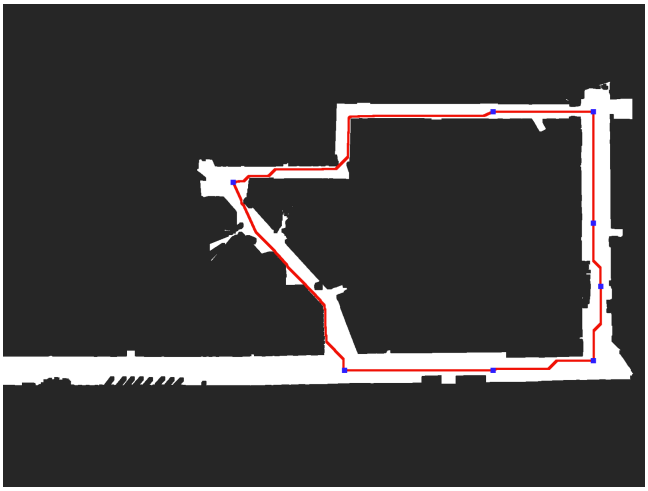

Figure 2. The path returned by A* in the Stata basement map.

3 Sampling-based Planning

3.1 Bidirectional Rapidly-exploring Random Trees (BiRRT)

David von Wrangel

BiRRT is extending two trees simultaneously; thus, less space is needed to be covered to find a path. The advantage of RRT-like algorithms is that they can consider the change in environments. The BiRRT connect algorithm between two points is as follows:

```
start_tree = Tree(start_position)
end_tree = Tree(goal_position)

while not timed_out:
  swap_trees(start_tree, end_tree) with 50% chance
  if chance < 5%:
    q_sample = random node of end_tree
  else:
    q_sample = random sample

  target_node = extend(start_tree, q_sample)
  if extended:
    last_node = extend(end_tree, target_node)
    if reached:
        return path
```

Algorithm 1: BiRRT connect

The extend function in BiRRT is a linear interpolation to the target node where all collision-free paths will be added to the nearest neighbor of the tree. Once the second tree reaches the target node, the node's parents will be traversed to extract the path. Further post-processing, like shortcutting, can be applied to the path. For a sequence of nodes, we not only call BiRRT to connect between consecutive pairs but also reuse the end tree of the last connects as the start tree to speed up.
It is important to note that we are sampling in two-dimensional Cartesian space. For a non-holonomic system, like our car, we could use a kinodynamic RRT approach to get feasible trajectories. BiRRT will be slightly more complex since the goal tree has to make the car drive backward.

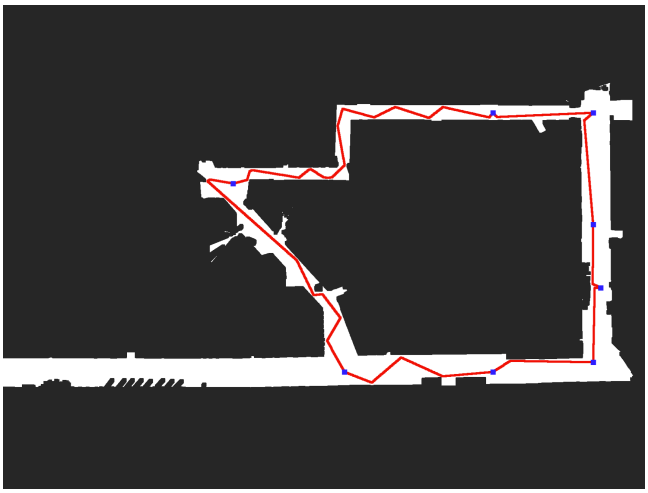

Figure 3. The path returned by BiRRT in the Stata basement map.

We used 8 milestones to call BiRRT connect. In 12.11 seconds, it returned a path in the Stata basement. While BiRRT explores the map very fast, we find that the basement does not suddenly change in geometry. Thus, using other methods like probabilistic roadmaps could be more efficient since they store a lot of the explored connections.

3.2 Probabilistic Roadmaps (PRM)

Juan Rached

To construct our Probabilistic Roadmap we sample 1500 random points from the given occupancy grid. We treat each point as a node in a graph structure. For each node we check state validity, whether that coordinate in the grid is occupied or not, and find its 5 closest neighbors. We append the 5 closest neighbors to the original node as edges. We verify the coordinate of each edge is indeed available in the occupancy grid. Any node or edge that is occupied is not added to the structure. We also check for edge validity, that is, we interpolate a straight line connecting the node to each of its edges. If any connection is obstructed then the edge is not added to the structure. A high-level overview of the algorithm is displayed below.

```
Let: O = Ocuppancy Grid, R = Empty Set
loop:
  n = random coordinate from O
  if n is not occupied:
    append n to R
for n in R:
  e = 5 closest neighbors of n
  for elem in e:
    if elem is occupied:
        drop elem from e
    if line connecting elem and n is obstructed:
        drop elem from e
  append e to n
```

Algorithm 2: Probabilistic Roadmaps

After this algorithm runs we are left with our roadmap R, a graph structure, with nodes n, and every node has a set of edges e.

3.3 Path Querying

Juan Rached

To query a path we create a function. It takes in a sequence of coordinates in the occupancy grid we want the racecar to traverse through before it reaches its final pose. Since the set of coordinates in our roadmap is a subset of the set of coordinates in the occupancy grid, not all points in our sequence will be in our roadmap. Hence for every point in the sequence we find the node in our graph that has the smallest distance to it and use that node to represent it in our roadmap. Lastly, we use A* search between every two nodes in the roadmap to find the optimal path between them. Fig. 4 shows an example of a path found through PRM sampling and A* search.

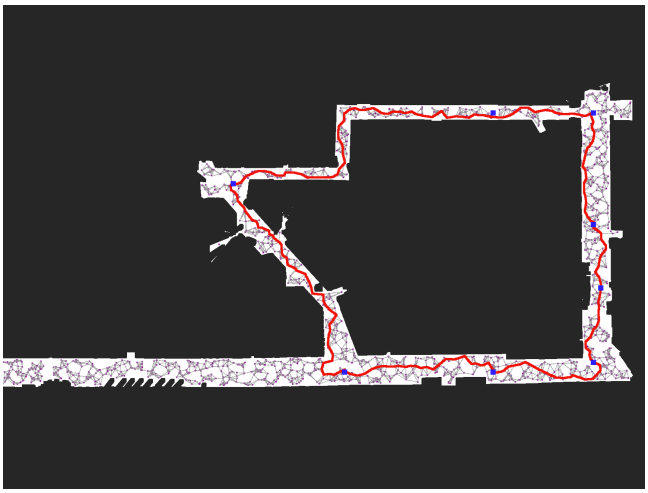

Figure 4. The path returned by PRM in the Stata basement map.

3.4 Path Shortcutting

Jinger Chong

To further improve the paths found through BiRRT and PRM, path shortcutting was implemented. This algorithm iterates through all the pairs of nodes in the path with at least one node between them, and checks whether directly connecting them results in a collision-free edge. If so, the path is then shortened by discarding all nodes between them and the algorithm is called again on the new path.

Unsurprisingly, this process significantly impacts runtime, especially if there are many nodes in the path or many obstacles in the map. To minimize recursive calls, pairs of nodes furthest from each other were hence checked first so that the number of nodes can be reduced quickly. Additionally, a memoization set was maintained to keep track of the pairs of nodes already known to produce an invalid edge. This way, subsequent calls do not have to repeatedly check its validity.

Figure 5 presents the results of applying path shortcutting on the paths shown in Fig. 3 and Fig. 4.

Figure 5. The shortened paths returned by BiRRT and PRM respectively in the Stata basement map.

4 Pure Pursuit

Ethan Hammons

The pure pursuit controller is a geometric path tracking algorithm that assumes the simple bicycle model to compute a steering angle under constant speed.

4.0.1 The Pure Pursuit Algorithm

Given a desired trajectory, we draw a look-ahead circle of radius L_{fw} offset by a constant l_{fw} from its rear axis. The intersection of the circle will be used to compute μ which will yield the steering angle δ.

$$\delta = atan2(\frac{L \sin \mu}{\frac{L_{fw}}{2} + l_{fw} * cos(\mu)}) \tag{1}$$

Here, L is the wheelbase distance. We chose the offset l_{fw} to be at the location of the LiDAR and fixed L_{fw} at 2.

```
def get_steering_angle(trajectory):
  center = point(lfw,0)
  circle = make_circle(center, radius = Lfw)
  intersections = find_intersections(circle, trajectory)
  if no intersections:
    return 0.0
  x_ref, y_ref = select(intersections)
  mu = atan2(y_ref, x_ref)
  delta = atan2(L*sin(mu), Lfw/2 + lfw*cos(mu))
  return delta
```

Algorithm 3: Pure Pursuit control

The pure pursuit takes advantage of finding an intersection between the look-ahead circle and a given trajectory. This given trajectory is outputted from one of our path planning algorithms, and the intersection position is then used as an input to control the car steering. This pure pursuit model utilizes the exact same algorithm as the wall follower, with only some minor tweaks to ensure data types line up and that reference frames are correct.

A circle can have multiple intersections with the trajectory. Depending on the task, one intersection might be less interesting than the other. For instance, an intersection behind the car is less valuable for following the wall. So we sort the solutions by the farthest distance in front of the vehicle.

5 Experimental Evaluation

Jinger Chong

To test and compare the various methods, an occupancy grid was generated from an image of the Stata basement map. The same sequence of 8 milestones

were then inputted in each planning algorithm. In addition to creating visualizations of the resulting nodes, edges, and paths, runtimes and distances were also recorded. These are shown in Table 1.

TABLE I. Average benchmarks for A*, BiRRT, and PRM.

Benchmark	**A***	**BiRRT**	**PRM**
initialization runtime (s)	573.714	N/A	125.712
path-finding runtime (s)	12.845	12.112	0.421
shortcutting runtime (s)	N/A	20.738	11.592
original path distance (px)	3100.20	3416.64	3652.49
shortened path distance (px)	N/A	3072.58	3080.72

BiRRT and PRM have their strengths and weaknesses. For instance, PRM's path-finding is faster than BiRRT, but its initialization takes significantly longer. This means that PRM is ideal for a static environment such as the Stata basement, while BiRRT is better for a dynamic environment such as a busy crosswalk. Evidently, however, A* is worse than BiRRT and PRM since both initialization and path-finding are slow.

Furthermore, it can be observed that shortcutting can decrease the total path length by $10 - 15\%$. While this comes at a trade-off in runtime, the reduced distance can be crucial in some applications. For instance, depending on the size of the map and the speed of the robot, the added runtime may actually be shorter than the added time taken to traverse the longer path.

The robot was then asked to follow the resulting path in simulation using pure pursuit. Similar to the previous lab, actual and predicted odometries were obtained by recording the `/odom` and `/pf/pose/odom` topics respectively. Fig. 6 and Fig. 7 show the plots of the various trajectories.

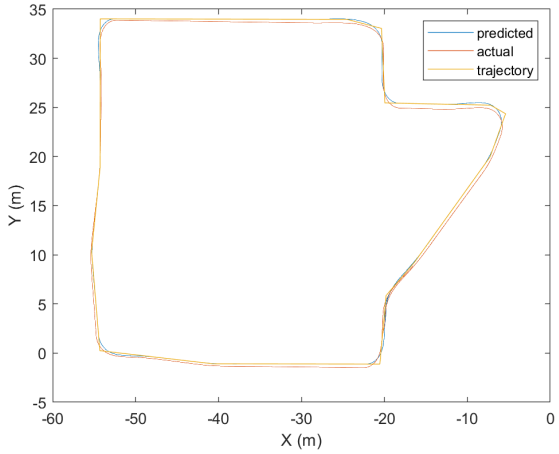

Figure 6. The trajectories in BiRRT with shortcutting simulation.

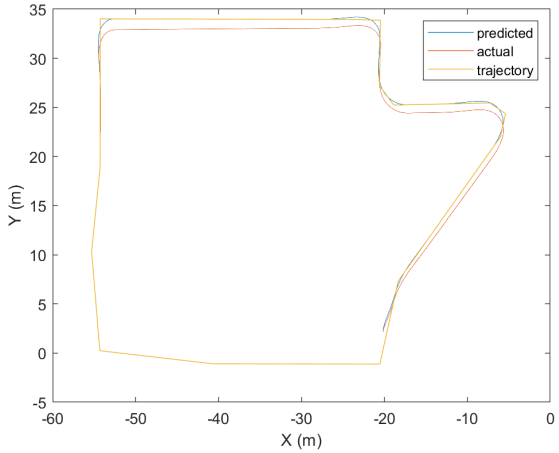

Figure 7. The trajectories in PRM with shortcutting simulation.

To evaluate these performances, average localization and trajectory errors were also computed and given in Table 2. Localization error is defined as the distance from each actual odometry position to the closest predicted odometry position, while trajectory error is defined as the shortest distance from each actual odometry position to its corresponding edge in the path. The full graphs of these errors with respect to time are provided in the appendix.

TABLE II. Average errors for BiRRT and PRM with shortcutting.

Errors	**BiRRT**	**PRM**
localization error (m)	0.2451	0.6139
trajectory error (m)	0.2068	0.5314

Since the average localization and trajectory errors for each algorithm are very close, the errors are likely caused by localization. This is further justified by the similarities in the error graphs. This means that pure pursuit thinks it is successfully following the path, but the predicted position of the robot is inaccurate.

6 Conclusion

Ethan Hammons

After testing three different path planning algorithms, it is clear to us that PRM is the best option based on our tests. Of all three algorithms it gives us the fastest run time along with very accurate paths. Because of this we will likely use this algorithm for path planning if we need it for the final challenge. Another option we looked at was shortcutting in our PRM algorithm. It has drawbacks such as a significantly longer runtime, but in an environment that is static the path may only need to be computed once. If we don't use shortcutting though, we get paths that are sometimes infeasible for the car. So, currently our plan is to keep using it, and maybe optimize it for faster runtimes. Now that we have our plan for path planning, along with all of the other foundational programs for the robot, our team is ready to begin work for the final challenge.

7 Lessons Learned

Jinger Chong

In terms of logistics, we were definitely more organized for this lab than any previous lab. We delegated tasks as soon as the lab came out by splitting into subteams for search-based planning, sampling-based planning, and pure pursuit. I chose to work on PRM because I already have experience with search algorithms and I wanted to learn new methods. I was able to contribute a lot in the technical aspect, which made me feel productive and fulfilled. David and I even created our own visualization and testing by loading the image of the Stata basement map as a numpy array and drawing the resulting nodes and edges of the graphs. It was very exciting to not only see our progress but also have solid performance benchmarks to prove it.

Ethan Hammons

In this lab I enjoyed getting to look at and better understand path planning algorithms. I had learned them in classes a while back but never got to look at real implementations of them, which was nice. I also feel like I reinforced my understanding of ROS and the systems of publishers and subscribers. In terms of communication, I think our team did a lot better this time because we were integrating much earlier on. After I made my pure pursuit, David and I spent many hours making sure it worked in simulation so that integration would go as smooth as possible. I know now how important preparing before integration is.

Nina Gerszberg

This lab has been one of my favorites. I implemented the search algorithm for path planning using A*. It's tied in well with what I have learned in my other course 6 classes which I really like. I think from an interpersonal perspective, we worked much better as a team for this lab which was wonderful. Starting early and being very thorough in our communications with one another seem to have been key towards success. It seems our lessons learned from previous labs have stuck which is amazing news. Overall, this lab has left me optimistic going in to the competition.

Juan Rached

For several labs before this one my teammates and I considered starting to work earlier down the week. Although for every lab after that initial discussion we started off a little earlier, this was the first time we started as soon as the lab was released. This turned out to be hugely beneficial as I happened to get sick later that week, making it more difficult for me to meet with the team and work on the assignment. By the time I was sick however, I had done enough work that we could make the deadline. I also learned how to implement a probabilistic roadmap so that was a lot of fun too.

David von Wrangel

Motion Planning is something I was thoroughly exposed to in my UROP, so it was easy for me to gauge how long this lab would take. I set the basic architecture early on by writing lots of skeleton code for the PRM team to fill out s.t. integration would be easier later. It turned out that it required only 2 lines to change in the integration code to go from BiRRT to PRM. I also learned that non-holonomicness is annoying and not as trivial to handle in configuration space as expected. Learning about the differential flatness of the bicycle model and playing a bit with graph search in convex sets was a lot of fun.

8 Appendix

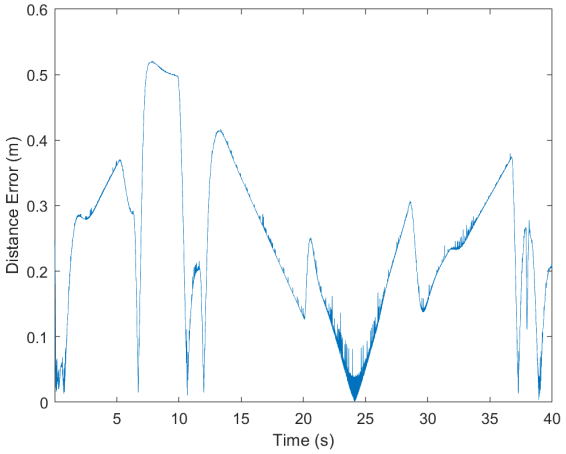

Figure 8. The localization error in BiRRT with shortcutting simulation.

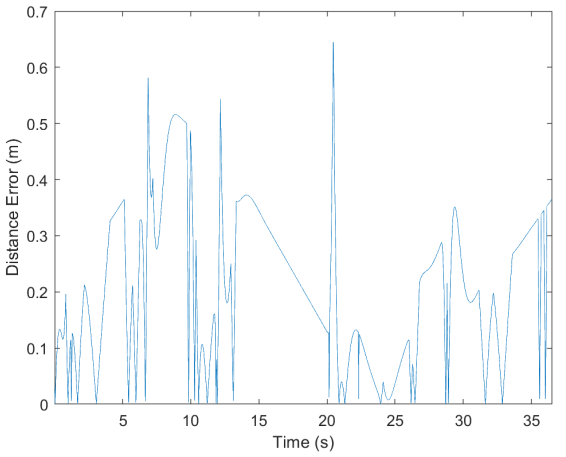

Figure 9. The trajectory error in BiRRT with shortcutting simulation.

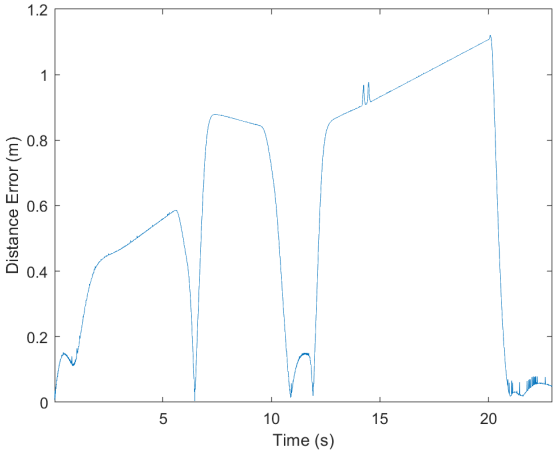

Figure 10. The localization error in PRM with shortcutting simulation.

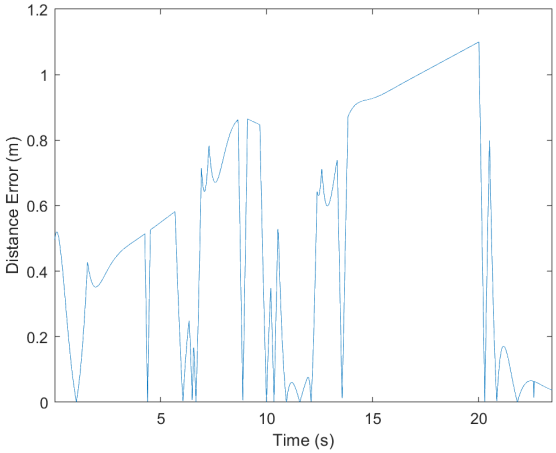

Figure 11. The trajectory error in PRM with shortcutting simulation.