Cornell University

ECE 6950 Robots as Embodied Algorithms

Final Course Project 2023

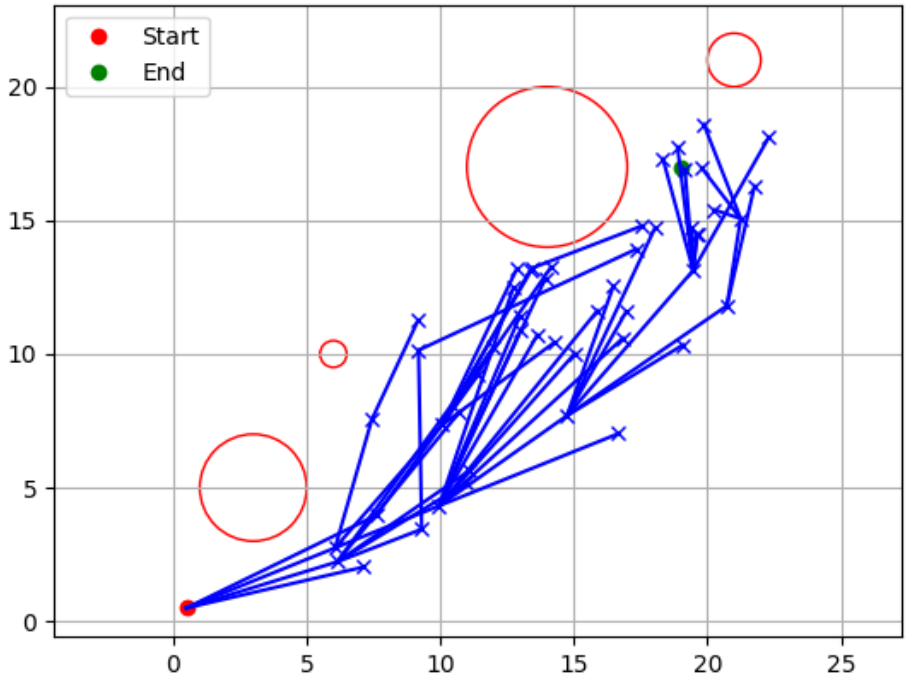

Figure 1: RRT* Constructed Tree in this Project for a Given Map Showing the Exploration Process with 50 Nodes Present.

Project Title:	**Developing a RRT Based Robotic Path Planner for Holonomic/ Non-Holonomic Robots**
Student:	**James Jun Hao Ong**
CID:	**jo445**
Course Supervisor:	**Professor Nils Napp**

1 Introduction

Motion planning forms a crucial part of robotic applications, ensuring robots are capable of generating an efficient path between source to destination nodes while avoiding obstacles present on trajectory [1]. Within robotic literature, many planning algorithms have been proposed comprising A*, probabilistic roadmaps, and RRT [2]. This project focused on developing a software simulator for an RRT based planner due to its relative efficiency at exploring high dimensional space. Meanwhile this also explores the effect of using real-life motion models such as Ackermann steering when conducting path planning. Overall, this results in the delivery of software for calculating and animating RRT paths is developed while the results from this simulator and environments are also reported.

2 Methodology

2.1 Core RRT Algorithm

The primary objective of an RRT algorithm is to provide an efficient means of exploring a given search space. It is designed to navigate complex environments that contain objects and obstacles. This algorithm is aimed at being able to provide a relatively efficient solution that is capable of being deployed for real-time usage. Its operating principle works by rapidly and randomly constructing a space-filling tree which expands to encompass unexplored search space regions. As this continues, the planner will expand to encompass regions of space closer to the goal node providing a solution that is probabilistically complete. A depiction of the algorithm and its functionality is captured and depicted by the following pseudo-code in algorithm 1.

Algorithm 1 RRT Core Algorithm

```
procedure RRT_runalgo(x, y, max_iter)
    initial_node ← robotNode(x, y)
    node_list ← []
    for i = 1 to max_iter do
        rand_sample ← get_random_sample()
        parent ← get_parent(rand_sample)
        new_node ← get_best_sample(parent)
        node_list.append(new_node)
        if new_node ∩ dest_node then
            return new_node
    return min_dist(node_list)
```

In this example, the robotNode() object refers to a data-structure like node with a series of x,y coordinates and parent nodes in the expanding random tree. Meanwhile, the node_list array maintains a collection of all previously explored nodes as needed for determining the optimal path. Here, returning the new_node as an object will provide enough information to obtain a singular RRT path taken from the source to destination while avoiding certain unnecessary information about the whole tree architecture. The operation of the functions and underlying algorithms are better explained using a series of table and diagrams and are as such presented in these two tables (using a collection of random data).

Datapoint	Coordinates	Distance to End
Sample 1	(13,9)	10.0
Sample 2	(17,14)	3.61
End Node	(19,17)	0.0

Table 1: Collection of 2 sample points and their corresponding euclidean distance to goal node.

In table 1, it can be seen that there would be multiple samples randomly generated from a given parent node. In the base code implementation, these parameters are generated based on a series of distance and angle measurements obtained from a random sampler. In this instance, the RRT algorithm selects sample 2 as it has the minimal distance to the destination node at (19,17). Meanwhile, an identical process occurs when selecting and determining the parent node based on the random sample. When performing this step, the algorithm will determine the parent node from the node_list collection which has the minimal euclidean distance to the random sample.

2.2 Ackermann Steering

Another component of this project focused on developing a simulator to capture and represent the kinematic constraints pertaining to driving. With this principal in mind, the project aims to use an Ackerman steering model representing a series of nonholonomic constraints [3]. This model provides the car with 2 degrees of freedom enabling any series of x,y coordinates to be reached in the plane, however, the orientation angle θ of the car cannot necessarily be set. The kinematic equations required to obtain new x,y,θ parameters are given by the equation 1 below.

$$\begin{bmatrix} \delta_x \\ \delta_y \\ \delta_\theta \\ \delta_\phi \end{bmatrix} = \begin{bmatrix} u_1 \cos(\theta) \\ u_1 \sin(\theta) \\ \frac{u_1}{L} \tan(\theta) \\ u_2 \end{bmatrix} \quad (1)$$

In this equation, the $u_1$parameters refers to the wheel speed/ velocity of the robot. Meanwhile, parameter u_2 refers to the steering angle of the vehicle present which can be used to determine the straightness of the paths taken/ tightness of turns. Here, if the $u_2$parameter is set to zero and the initial value of ϕ was 0, the robot would essentially move in a straight line trajectory between the source to destination node and vice versa. A diagram of the robot and representation of ackermann steering is provided below in figure 2.

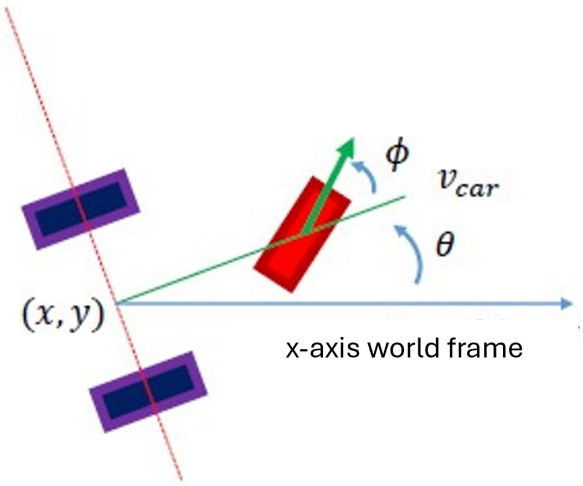

Figure 2: Steering representation for a non-holonomic robot with steering angle ϕ and values (x,y,θ) representing the present position of robot.

2.3 RRT* Algorithm

Following the development of the Ackermann steering model this project focused on developing an RRT* based approach seeking to prioritize path search efficiency. The RRT* algorithm seeks to progressively optimize the paths taken via an iterative path optimization procedure. When developing this algorithm for every newly sampled node, RRT* will attempt to rewire the tree by checking the nodes neighboring the recently sampled point and determining whether the distance to the neighbor could be reduced. Here, the pseudocode described by algorithm 2 effectively depicts the operating principle of RRT* which builds on top of the RRT algorithm after obtaining the random_sample.

Algorithm 2 RRT Core Algorithm

```
    procedure RRT_runoptimizer(origin, rad)
2:      for node ∈ node_list do
            new_dist ← origin.dist + dist(entry)
4:          if node.dist <= new_dist <= rad &&
    no_collide(origin, node) then
                node.dist_travelled ← new_dist
6:              node.parent ← origin
                node_list.append(new_node)
8:      return
```

In algorithm 2, the origin effectively refers to the source node of the model. Meanwhile, the radius is used to define and determine the neighbourhood that will be searched for the ground truth and correct answer.

2.4 Obstacle Avoidance

During the development phase of the rover, it was necessary to ensure the rover could plan a path that and route that is capable of obstacle avoidance. This needed to ensure that no part of the rover would collide and intersect with the obstacles. To approach this, we project a pair of parallel lines directly from the corners of the rover that is moving along path which takes factors including the width of the rover into consideration.

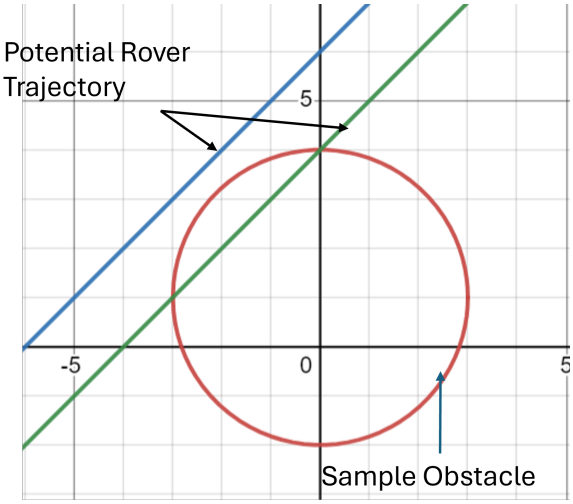

Figure 3: Rover path represented as parallel lines with obstacle as a circle.

From figure 3, it can be seen that there would be an invalid path between the rover and the sampled point. This is as part of the rover would intersect and collide with the circle as shown by the projection vectors. As such, the rover would need to randomly sample a new point on the map.

3 Software Development

3.1 Tools and Environment

A key objective of this project focused on developing the software necessary to deploy and run the RRT algorithms. As part of the operating environment, python was used to run the core code of the RRT algorithm and determining the optimal paths. This is due to the inherent simplicity of the language and offering a garbage collector which is useful given the object-oriented programming nature of the code and presence of graph-like data-structures [4]. Meanwhile, python offers several established and well-developed tools related to assist with the geometry of the problem offering comprehensive collision detection using the shapely library amongst many other features.

Meanwhile, for the visualization and deployment of code, a Pygame interface was used to develop the environment and obtain a visualization of the final paths. Here, figure 4 shows an image of the map and obtained path after running a version of RRT* of which is algorithmically shown in section

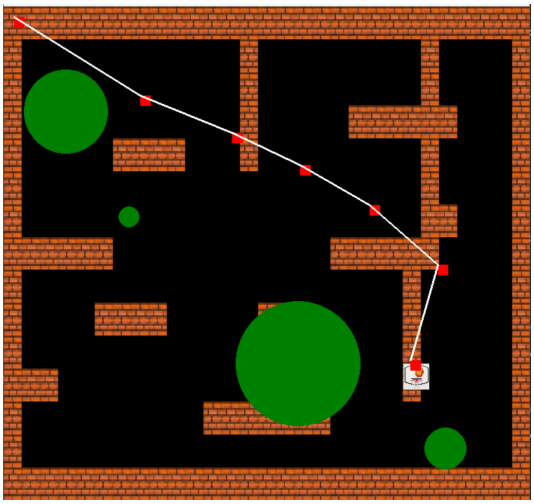

Figure 4: Here, the path taken by the robot to the destination node is shown by the series of white lines with the rover represented by red squares. It can be seen that the rover is able to construct a path capable of avoiding obstacles while effectively navigating to the end node.

4 Results and Analysis

To obtain a series of results and data for the development of this model, experiments were run for a total of 500 iterations. This attempted to measure and benchmark key performance metrics related to RRT, RRT* and Ackermann Steering Models. In the development of these models, it is necessary to ensure that there are robust collision avoidance procedures. This in part happens as there might be a series of obstacles which the rover may intersect with on its course between the source to obstacle. To effectively program a navigation suite for the rover we represent the rover's dimensions as a series of parallel lines which would be present at the corners of the rover.

4.1 RRT Model Base Performance

The results of percentage of occurrences where the goal is reached for RRT algorithm are shown by figure 6. This demonstrates that the percentage guarantee of success tends to asymptotically approach 100% as the number of iterations increases. This conveys how RRT algorithms generally tend to provide a probabilistic guarantee of obtaining a solution.

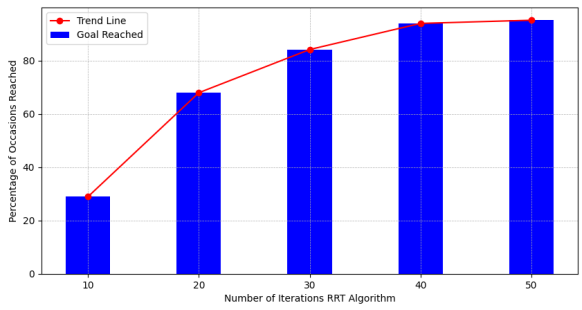

Figure 6: RRT algorithm number of new nodes generated against percentage of times that the destination is reached.

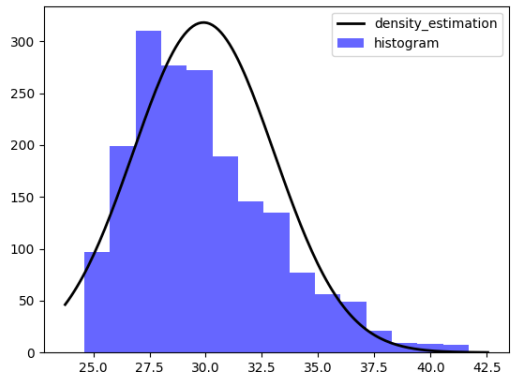

Figure 7: Diagram of the number of paths taken and path length out of a total of (1853 where the goal was reached) and normal distribution of path lengths with benchmark minimal euclidean distance length of 23.8.

From the histogram in figure 7, it can be seen that the path lengths tend to follow an approximately normal distribution with a mean of around 30.0 when developed. Here, it is a property that if the RRT converges on the goal node that this may not be the optimal solution with the obtained solution only offering probabilistic guarantees on optimality.

4.2 RRT* Performance

While RRT doesn't necessarily provide guarantees of optimality, it can be refined to developing an RRT* based algorithm. This is due to the RRT* algorithm doing a continual rewiring of nodes as mentioned and described in section 2.3. Additionally, RRT* will also exhibit the property of continually improving and refining paths as the number of iterations increases. The figure from 8 shows a comparison between an RRT* algorithm that was run for a maximum of 250 iterations vs the regular RRT in terms of path lengths.

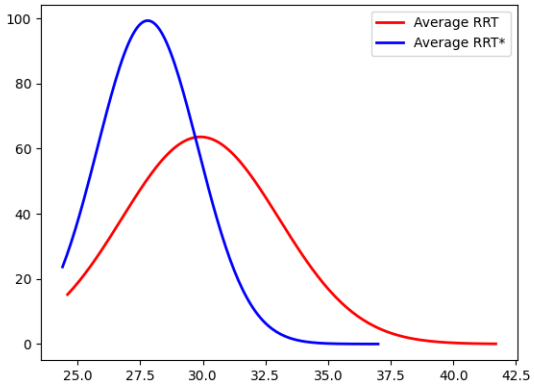

Figure 8: Comparison between RRT and RRT* with x-axis representing the average path length and y axis representing number of paths.

The results obtained in figure 8 supports the notion that RRT* algorithms tend to outperform the RRT distributions when ran for high numbers of iterations. There is a clear trend that the average path length for the RRT* algorithm tends to be significantly shorter than that of the RRT algorithm. The mean path length for the RRT* distribution was approximately 27.8, whereas it is determined to be 30.5 for the equivalent RRT algorithm suggesting the performance advantages offered by RRT* in comparison to RRT. While the trend isn't fully conspicuous based on a cursory observation of the distribution, it should be noted that as a benchmark, the minimal path length is approximately 23.4 (based on euclidean distance, but unachievable on a real-map due to obstacles present). If there were potentially more samples that were present, the trend could potentially be much clearer.

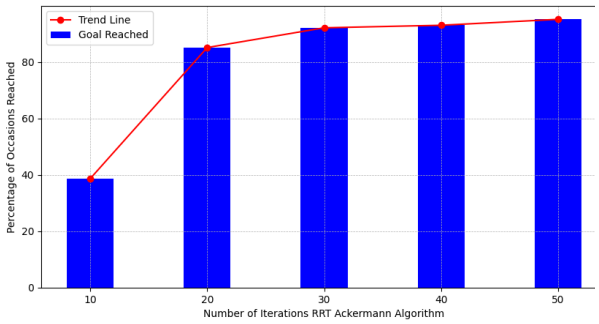

Figure 9: Statistics Indicating the Performance of the RRT Model on Ackermann Steering

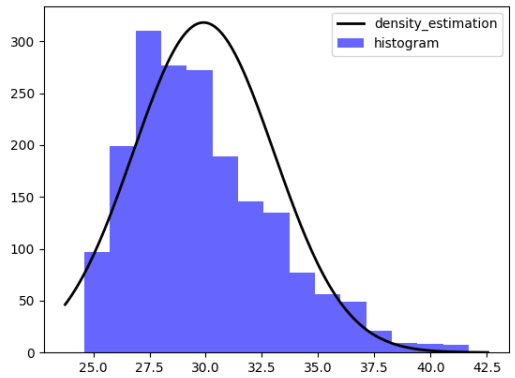

Figure 10: Graph to Show the Distribution of Ackermann Steering Performance using RRT Sampling.

5 Preliminary Results Ackermann Steering

A model was developed using the kinematic constraints defined in section 2.2 to simulate Ackermann Steering. The results from the number of iterations from RRT Ackermann suggest that this distribution does tend to resemble the trend shown by RRT in figure 4. This illustrates that as the number of samples increases the likelihood of reaching the goal node is also increased. However, one relatively surprising trend was the similarities between figure 10 and 7. While these results are useful in showing trends, it should be the case that non-holonomic Ackermann steering should result in potentially shorter average path lengths. Hence, to fully capture the path length values, it is possible that several modifications need to be made to the collision detection system. With this fixed and developed it will likely result in path lengths that are accurate and reflective of real values.

5.1 Conclusion

The scope of this project delivers the preliminary software tools needed for simulating and mapping the trajectories taken by a rover and conducting obstacle avoidance. It can be seen from the results obtained in this project that this software verifies several trends. This namely includes and pertains to that the rover is presently capable of obstacle avoidance while also now being capable of finding shorter paths using the RRT* based approach. Additionally, the project has been well-developed and designed from a software frontier. It is capable of running a full scale visualization process displaying a live-render mode of path planning while also providing the base code to enable users to tweak and modify RRT parameters as necessary. In future development of this project, features including improvements to the Ackermann steering model will be made and deployed.

References

[1] S. Azadi, R. Kazemi, and H. R. Nedamani, "Chapter 10 - trajectory planning of tractor semitrailers," in *Vehicle Dynamics and Control* (S. Azadi, R. Kazemi, and H. R. Nedamani, eds.), pp. 429–478, Elsevier, 2021.

[2] K. Karur, N. Sharma, C. Dharmatti, and J. E. Siegel, "A survey of path planning algorithms for mobile robots," *Vehicles*, vol. 3, no. 3, pp. 448–468, 2021.

[3] N. Correll, B. Hayes, and C. Heckman, "Introduction to autonomous robots." Accessed: 2023-12-13, Chapter 6, Section 2.

[4] Python Software Foundation, "Garbage collector - python developer's guide," 2023.

[5] F. Afrasiabi and N. Haspel, "Efficient exploration of protein conformational pathways using rrt* and mc," pp. 1–6, 09 2020.

6 Appendix

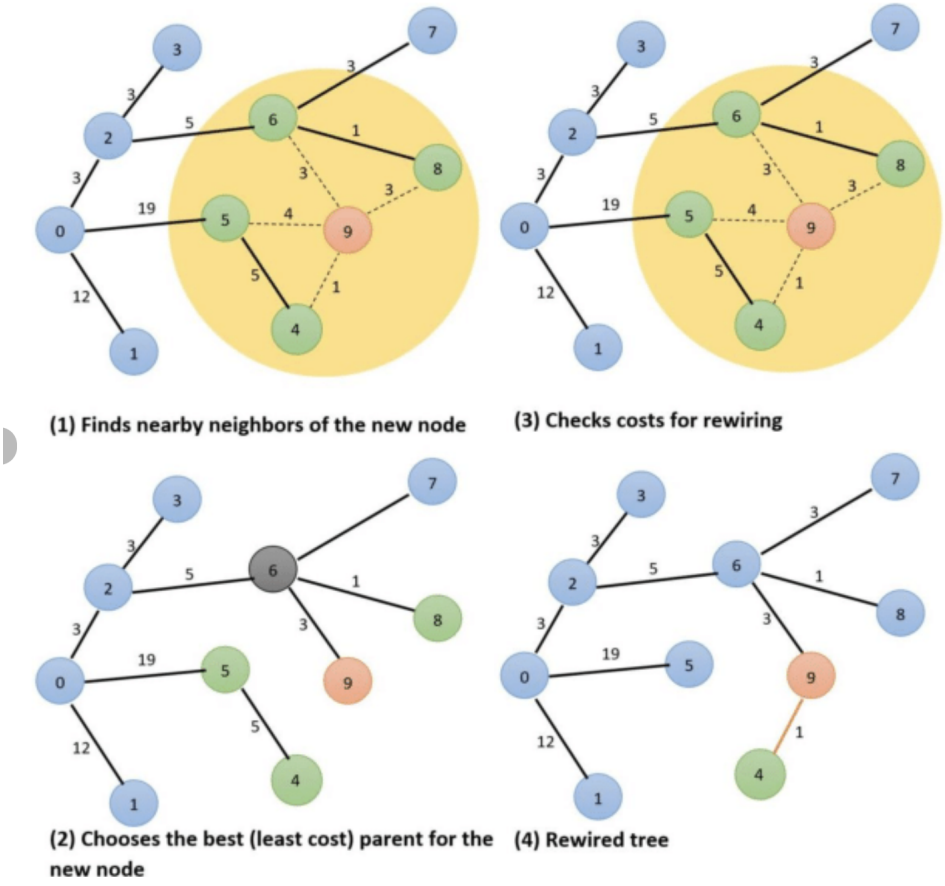

Figure 11: Here, the node 9 is added in red with a shortest path length of 8. From the diagram in the bottom right, we see that there is a shorter path through node 9 to destination node [5].