CSci 8994
Project - Drive
Report

Yashasvi Sriram Patkuri - patku001@umn.edu
Stephen J. Guy - sjguy@umn.edu

Robot: A machine that can sense, plan and act on the world.

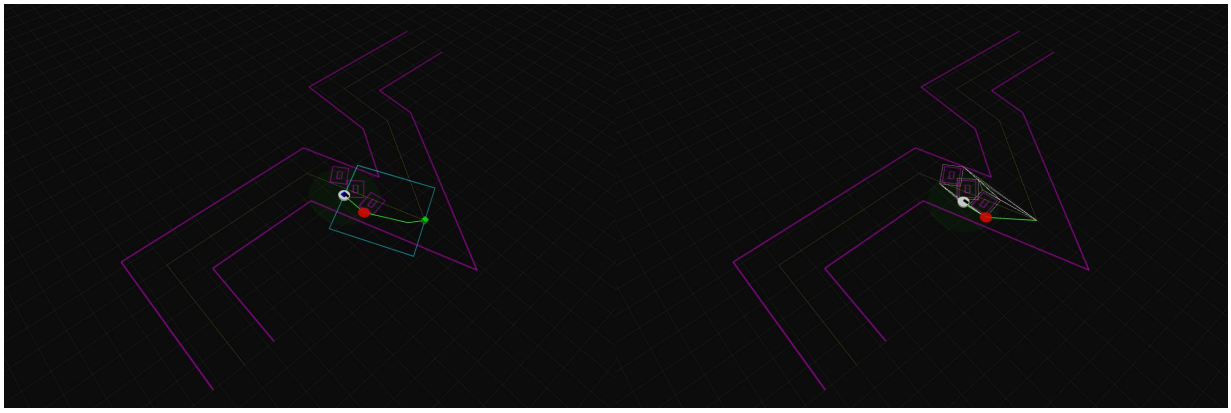

Figure 1: Agent navigating a path. Left: Planning using RRT*, Right: Planning using Visibility graph.

1 Abstract

Following a path while avoiding obstacles is one of the basic tasks of autonomous driving. In this work, we develop two real-time methods for solving this task for static initially unknown obstacles, using RRT* and Visibility graphs respectively, using a simple trick called padding obstacles. Both of these methods theoretically guarantee safety from collision and are asymptotically optimal. Both methods can be tuned to work real-time (at 30 Hz) with up to 10 sensed obstacles at a time. We test both methods on several scenes and compare their behavior and computation times. Finally we outline the pros, cons and bottlenecks of both methods and ways to make the output feasible for bicycle kinematic model.

2 Introduction

Goal Given a pre-planned path consisting of a list of points and a differential drive or bicycle kinematic model agent equipped with an obstacle sensor, the goal is to output a path that is,

1. Close to the given path.
2. Avoids obstacles.
3. Feasible with the kinematic model of the agent.

Purpose This project was designed to earn knowledge and experience in basic planning techniques in an outdoor autonomous vehicle context, so that they can be composed together and/or improved upon later to build more sophisticated and robust pipelines. A typical car on a road is kept in mind while designing the implementation and experiments.

Assumptions The robot motion is all in 2D. We assume no uncertainty in the sensors or control. We assume that we get rectangular bounding boxes of obstacles from the sensors. We assume static obstacles only. We assumed that a new control is applied instantaneously to the robot. The simulation is done in ROS/RViz.

Related work There is a lot of interesting work in the problem of path planning in known and unknown environments, with static and dynamic obstacles. Kavraki, Svestka, Latombe, and Overmars 1994 introduces a simple paradigm changing method called probabilistic road map for global path planning that avoids local mininma by discretizing continuous space as a graph. This combined with A* graph search makes a simple path planning algorithm. Koenig and Likhachev 2002 introduces a simple algorithm called D*Lite for re-planning in dynamic environments which can be used in case of unknown environments too. Hartman and Benes 2006 introduces a simple repulsive force based method for animating bird like agents in a realistic fashion without collisions. Karaman and Frazzoli 2011 introduces RRT*, an asymptotically optimal version of RRT. Planar visibility graphs are extensively studied for motion planning purposes. El Khaili 2014 introduces visibility graphs for planning in presence of moving obstacles. RRT* and Visibility graph are used extensively in this project.

Our contribution is to use these methods along with a simple trick to output a path similar to given path which avoids collision with obstacles. And in the process making it feasible for both differential drive and bicycle kinematic models.

3 Approach

We incrementally present our approach in this section.

Setup The given path is constant throughout the motion. We use a chain of line segments to represent the given path. This typically represents the path given my GPS planners.

For a given agent we often operate with its circumcircle. For the remaining of the text radius of an agent refers to the radius of circumcircle of the agent. We use a circle with orientation arrow to represent the agent.

We assume that the obstacles are static, but their position are unknown to us at the beginning. These obstacles can model parked cars, blockades, pits, damaged road etc... The obstacle sensors on the agent give a list of nearby obstacles at some rate. If an obstacle is in some arbitrary shape, the sensors give us a best fit rectangular bounding box around it. Therefore we use oriented rectangles to represent the obstacles. This way there is no loss of generality.

Configuration space To plan using graphical methods in presence of obstacles, one can just remove the edges of the graph that intersect with the obstacles. A subtlety here is that we need take the extent of the agent into account. For example if we represent path from the center of the robot, the path in the first sub-figure of Figure 2 will make robot collide whereas the path in second one does not.

Let the set of possible points in space where center of the robot can exist be called configuration space. Let the set of points where the center the robot cannot exist for it to be not colliding with any obstacle be called a configuration space obstacle. For a circular agent and a line obstacle the configuration space obstacle will be intersection of two circle

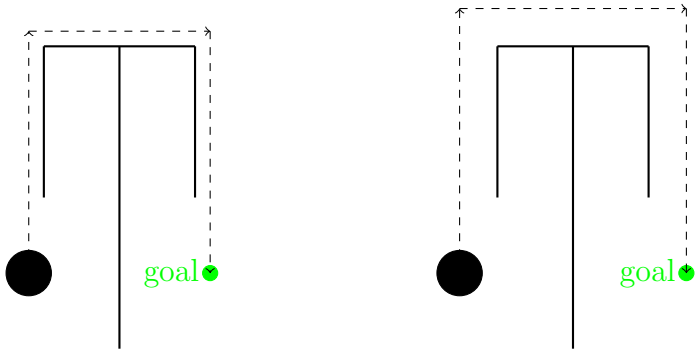

Figure 2: Two possible paths for the center of robot. Left: Collides with obstacle, Right: Does not collide with obstacle

and a rectangle as illustrated in first sub figure of Figure 3. The exact configuration space obstacle will have rounded corners. To make intersection calculations simpler, we extend the configuration space obstacle to have a rectangular shape that circumscribes the original one. This does not matter in most cases and more importantly does not compromise on collision avoidance. Thus for every line segment we have a rectangle around it at a distance of agent's radius that describes the configuration space obstacle of the lines segment and agent duo.

Figure 3: For a circular agent and line obstacle, Left: Exact configuration space obstacle, Right: Extended configuration space obstacle

Intersecting line segments To check an intersection b/w a line segment and a rectangle we check intersection b/w line segment and four line segments of rectangle. Since all obstacles in this text are assumed to be rectangles, line segment line segment intersection check comes up a lot. For detecting intersection b/w line segment AB and CD let,

$$A \equiv \begin{bmatrix} b_y - a_y & -(b_x - a_x) \\ d_y - c_y & -(d_x - c_x) \end{bmatrix}$$

$$b \equiv \begin{bmatrix} a_x b_y - a_y b_x \\ c_x d_y - c_y d_x \end{bmatrix}$$

$$x = A^{-1} b$$

x represents the point of intersection of two lines formed by joining points A and B, and C and D respectively. To further ensure that point of intersection of these lines falls in b/w both line segments we check if

$$\|A - B\|_2 \equiv \|A - x\|_2 + \|x - B\|_2$$

$$\|C - D\|_2 \equiv \|C - x\|_2 + \|x - D\|_2$$

If lines are parallel or co-incident $det(A) == 0$ and A^{-1} is undefined. In such cases we define the line segments as not intersecting, unless the line segments are co-incident and touch each other.

A subtle case is when a line segment is completely contained inside a rectangle. In such case the above check fails to detect a collision. But for our context such an additional check is unnecessary since all line segments that start outside the rectangle connecting to the inner line segment will be detected and removed, effectively removing the inner line segment from any path generated.

Padding obstacles For the actual planning we use a simple trick to achieve the first two of our goals i.e. staying close to actual path and avoiding obstacles. For each line segment in the given path we draw two parallel imaginary line segments at some distance on either side. Then we join/clip the imaginary line segments on both sides to make a padding for the given route. These sets of line segments form two paths that go along the given path on both sides. This is illustrated in figure 4.

Figure 4: Construction of padding obstacles. Left: First step; drawing parallel lines. Right: Second step; clipping/joining them at intersections.

These sets of line segments are called padding obstacles. We include these with other static obstacles that sensors provide us while planning. By controlling the padding length we can tune how close to the given path the output path should be. At limit as the padding approaches zero, the output path and given path become the same.

One problem with this method is that for intersecting or looped paths the padding obstacles block the given path. This is because while joining/clipping we do not handle such cases. This is illustrated in figure 5. Near 180° turns in the path also cause weird padding obstacles. These problems can be mitigated with more sophisticated padding obstacle creation or simply by considering local padding obstacles.

Planning Figure 6 is used to illustrate the planning algorithms. The white circle represents the agent's circumcircle, the black line segment on it represents its orientation. The yellow path represents given path. The faint green circle around the agent represents sensing radius. The corresponding pink paths represent padding obstacles in configuration space. The pink rectangles represent static obstacles; inner ones are physical obstacles and outer ones are configuration space obstacles (at a distance of agent radius from inner ones). The green path represents the current output path. The red circle represents the next target position of agent after trajectory optimization.

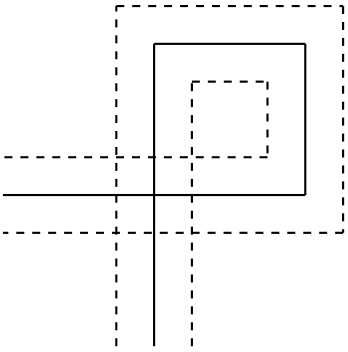

Figure 5: Padding obstacles blocking given path when it is self-intersecting.

RRT* Given the current pose of agent and current set of configuration space obstacles, both from physical obstacles and padding obstacles, we choose the sampling region for RRT* as the rectangle with center between current position and next point on given path, length as distance b/w current position and next point on path and a fixed width. This is illustrated in left sub figure of 6. The blue and green squares represent current position and next point on given path. The cyan rectangle represents the sampling region for RRT* for this instant. The white tree is the current RRT*. After sampling a fixed number of nodes and growing the tree, we output the path to the node closest to the next point on given path. This is repeated at every iteration of the loop.

Depending on the size of sampling region and the number of nodes sampled, the path may or may not be close to optimal path. But since this step is repeated at every instant and the sampling region decreases as the agent approaches the next point on given path, the path becomes more and more optimal.

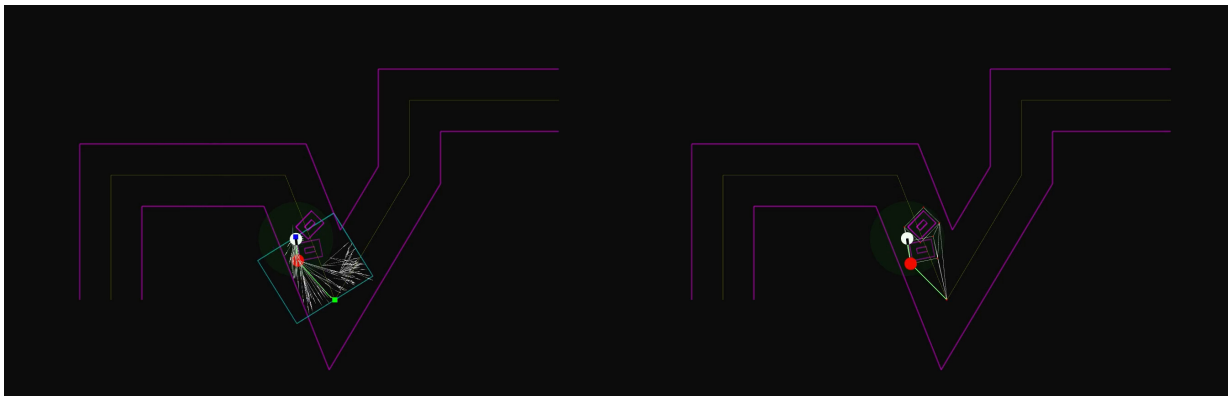

Figure 6: Agent navigating a path. Left: Planning using RRT*, Right: Planning using Visibility graph.

Visibility graph Given the current pose of agent and current set of configuration space obstacles, both from physical obstacles and padding obstacles, we create a graph with the following vertices

1. Current position.
2. Next point on the given path.
3. The endpoints of configuration space obstacles from physical obstacles.

We join any pair of vertices with an edge if it does not intersect with any configuration space obstacle (both from physical obstacles and padding obstacles). Such a graph is called visibility graph. This is illustrated in right sub figure of figure 6. The red dots are vertices and white line segments are edges of the visibility graph. In such a graph we do a A* search from the current position vertex and output the path to the vertex nearest to the next point on the given path. This is repeated at every iteration of the loop.

An subtlety during implementation is that while creating vertices from endpoints of configuration space obstacles, they need to be created a bit further away from the obstacle. Otherwise we get spurious intersections with obstacles even for valid edges.

Differential drive agent A differential drive robot has two controls linear velocity and angular velocity which can be independently controlled. To follow the path generated by any method, we follow a simple strategy of orient and translate. Invariantly, after reaching a milestone on the path the robot orients itself in the direction of next milestone without translating and then translates without rotating. After reaching a point on given path, the agent signals the local goal to be updated to the next point on the path.

Trajectory optimization We also perform an impromptu trajectory optimization over the generated path. At every instant instead of going towards the next milestone the agent goes to the furthest milestone in the path for which the line segment joining the current position and the milestone does not intersect with any configuration space obstacle.

4 Experiments

We used ROS/RViz for simulation. We used scenarios illustrated in figures 7, 8 and 9 for our experiments. We used parameters tabulated in Table 1.

Figure 7: 1: Straight path, 2: Arbitrary path.

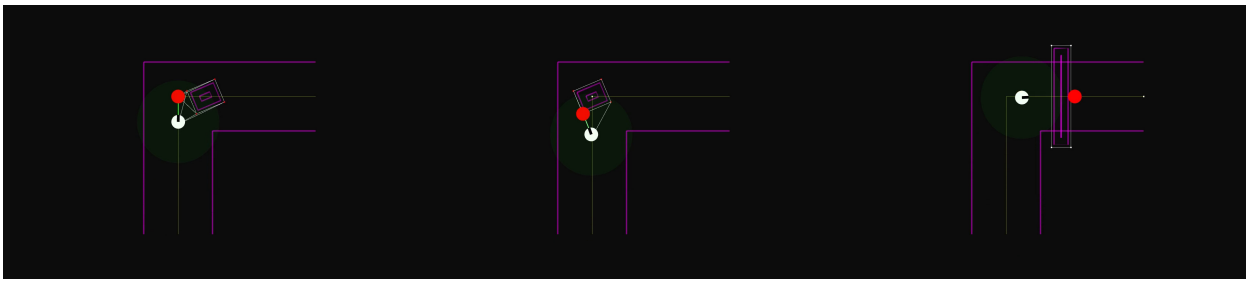

Figure 8: 1: Perpendicular turn with an obstacle, 2: Obstacle on a point of given path, 3: Complete blockade.

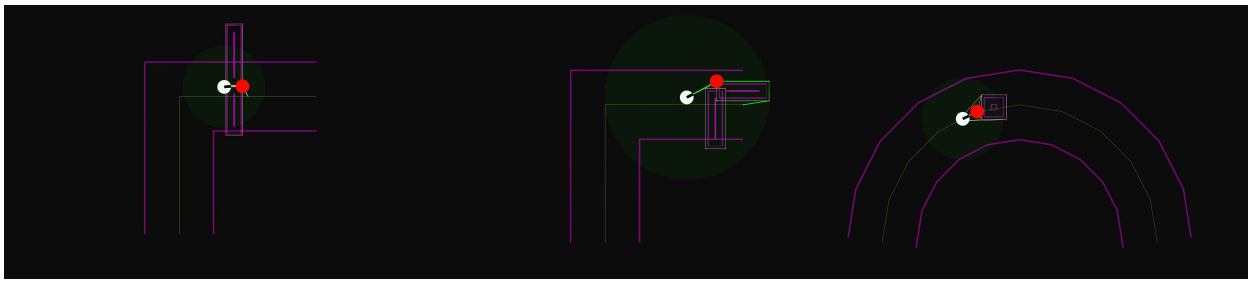

Figure 9: 1: Blockade with small opening, 2: Blocking last point on given path, 3: Arc.

Parameter	Value
Agent radius	0.25m
Route padding	1.5m
Sampling region width	1.8m
Sensing range	1.5m
Agent linear speed	0.03 m/s
Agent angular speed	0.03 rad/s
Planning frequency per loop	1
Simulation frequency per loop	5-10
Loop frequency	30Hz
Agent position slack	0.05m
Agent orientation slack	0.05m
Growth limit for RRT*	2m
Neighbour limit for RRT*	2m
Number of nodes for RRT*	250

Table 1: Parameters used for the experiments.

5 Results

Straight path This scene should ideally produce the given path as the output path. Visibility graph does this since the graph just contains two vertices and there is just one edge which is identical to given path. RRT* can also produce path very identical to given path. But it's behaviour is heavily dependent on tuning parameters. For small growth limits, the output path tends to wobble a lot. The same happens in many other cases. Usually large

growth limits mitigate the problem well. Increased density of points on given path tends to mitigate this as well, since it keeps the sampling region small.

Perpendicular turn Results and analysis in this scene is similar to the straight path scene. Differential drive agent takes a 90° turn at the junction.

Obstacles on course In this scene, there is one obstacle which will intersect with given path and one which will not collide with agent if it just follows the given path. Both methods successfully go around the former obstacle, with visibility graph path having a bit sharper turns. Both methods do not deviate near the latter obstacle.

Obstacle on a point of given path In this scene, there is one obstacle which is on a point on given path. Both methods are able to successfully navigate the scene. RRT* does a bit better since it does not rely on vertices of obstacle, visibility graph produces a path where agent goes in wrong direction and turns around. Since both methods go to nearest node they can find to the point on given path, both of them try to go as near as possible to the obstacle. This is unnecessary and potentially unsafe. This can be mitigated by planning a couple of points ahead on the given path.

Obstacle completely blocking path In this scene, there is one obstacle which completely blocks the final point on path. RRT* goes to the nearest point (it can find on it's sampled tree) to the final point on given path. Visibility graph stops when the obstacle is sensed, since the edges that connect from the agent center to the vertices intersect with padding obstacles. In a sense, visibility graph does better in this scene by preemptively stopping.

Blockade with small opening In this scene, there is two obstacles such that there is a very small passage b/w then to the final point on given path. Since RRT* relies on sampling, it struggles to generate new nodes that grow the tree through the small opening. Visibility graph is not affected by this since it only relies on the vertices and produces path seamlessly. The advantage of visibility graph is clearly seen in this scene.

Obstacles shielding last point on given path In this scene, there is two obstacles such that they shield the last point on given path from the direction of approach of agent. Since RRT* samples from the agent center to the next point on path, it struggles to find a path at the end. Visibility graph is not affected by this since it only relies on the vertices and produces a good path. By improving sampling region of RRT* this problem can be mitigated.

Arc In this scene, the given path roughly approximates a semicircle. This is similar to round about junctions common on highways. The path is littered with obstacles, modelling other cars. In this scene both RRT* and Visibility graph do comparatively well.

Arbitrary path In this scene, the given path is arbitrary and the obstacles are littered recreating some of the above cases. Both RRT* and Visibility graph do comparatively well.

Planning frequency For this algorithm to function in a real-world setting it is beneficial if it is as fast as possible. The main time taking steps in both algorithms are intersection checks. The figure 10 illustrates the effect of increasing number of sensed obstacles on the planning times and hence planning frequency of both algorithms. The planning frequency of visibility graph method decreases quite steeply with number of obstacles and actually goes below 1 for 50 obstacles. Whereas the planning frequency for RRT* method (with 250 nodes) decreases relatively slowly and remains above 10 even at 50 obstacles.

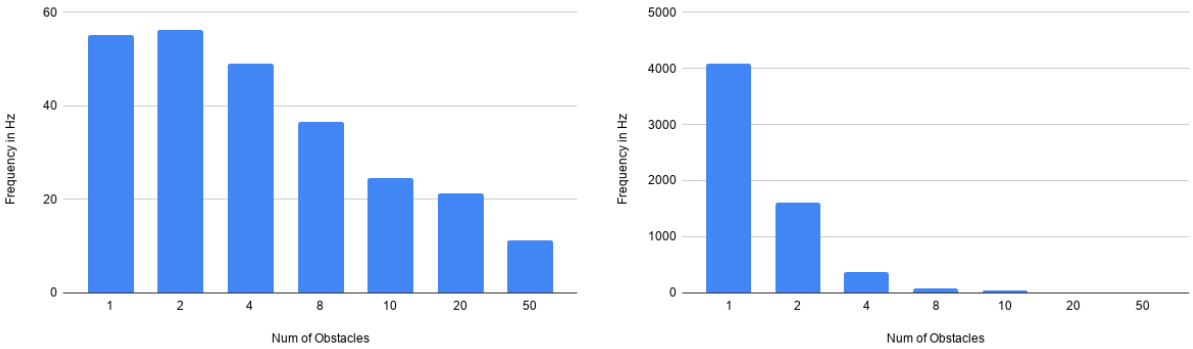

Figure 10: Planning frequency vs number of sensed obstacles. Left: RRT*, Right: Visibility graph.

Figure 11 shows the planning frequency for RRT* with number of sampled nodes. The frequency decreases as expected. Only at around 250 nodes the planning frequency is above 30 Hz.

6 Conclusion

We started out with a goal to output a path that is

1. Close to the given path.
2. Avoids obstacles.
3. Feasible with the kinematic model of the agent.

We have achieved the first two using our approach. As explained in the previous section both methods have their pros and cons. RRT* produces natural paths and does well even in cases of many obstacles, but it suffers from sampling problem and has to be tuned well to prevent wobbly paths. Visibility graph can navigate through small regions but does poorly in cases of lots of obstacles and generally produces paths with sharp turns. Both algorithms currently make the agent go to the nearest node to the local goal which sometimes produces unnecessary and potentially unsafe behaviour. In general with some improvements RRT* is more robust than visibility graph.

We have demonstrated feasibility of the path for differential drive agent in our experiments.

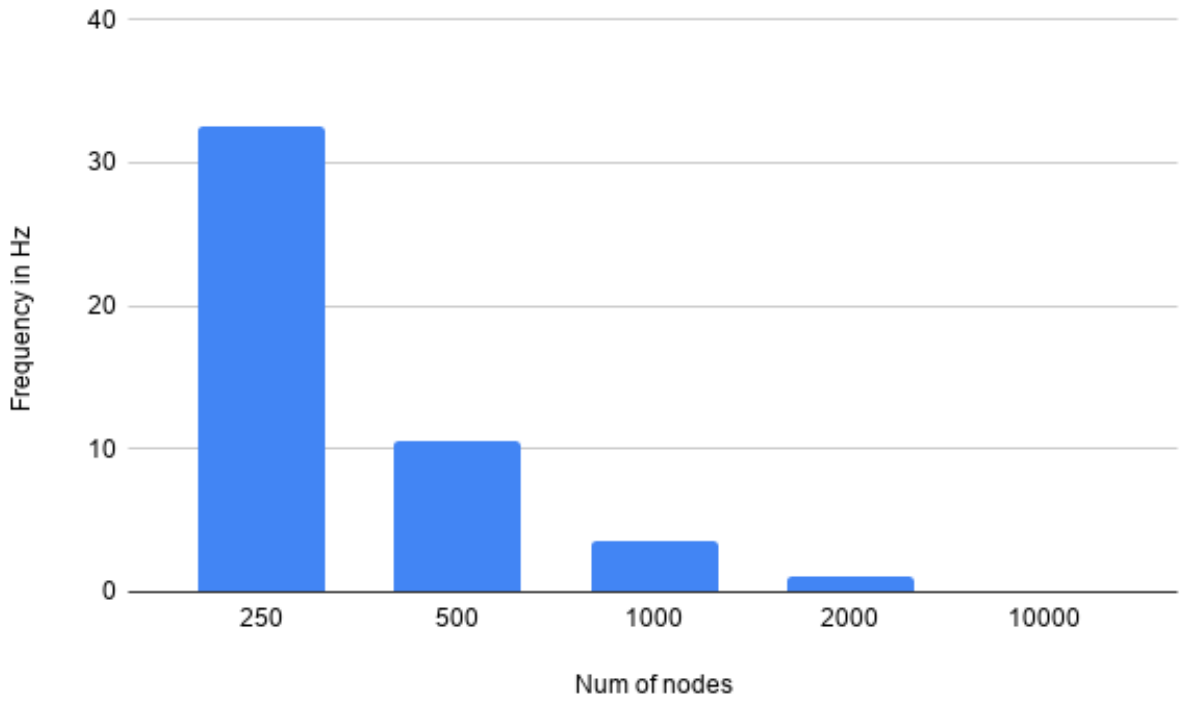

Figure 11: Planning frequency vs number of nodes for RRT*.

Bicycle agent is more constrained than differential drive agent in that it cannot turn in place. In order for the paths to be feasible for bicycle agent, they should be smooth. This can be done by increasing the density of points on given path and rounding at corners. One more way would be to plan for a few points ahead on the given path, as this would produce smoother paths at junctions. This is a good direction for future work.

The main challenges faced were

1. Implementing and debugging intersection checking algorithms.
2. Visualizing in RViz.
3. Tuning the parameters of RRT*.
4. Debugging intersection logic for visibility graph (slight padding for vertices).

The major bottleneck for RRT* is the number of nodes it samples and for visibility graph is the number of sensed obstacles. Intersections checks make up a large portion of planning time for both algorithms. Finding nearest neighbours is also a common operation in RRT*. Any spatial data structure (like K-D tree) that can speed these operations can decrease the planning times considerably. This is a good direction for further refinements.

More aspects like comfort can be modelled and optimized along with safety. Dynamic obstacles can be handled. Currently all the implementation is in a single ROS node. It can be divided into separate nodes with individual responsibilities. Current implementation does not keep track of previously sensed obstacles. Obstacles can be tracked to reduce computation time. These are also good directions for future work.

References

[1] L. Kavraki, P. Svestka, J. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," Stanford, CA, USA, Tech. Rep., 1994.

[2] S. Koenig and M. Likhachev, "D*lite," in *Eighteenth National Conference on Artificial Intelligence*, Edmonton, Alberta, Canada: American Association for Artificial Intelligence, 2002, pp. 476–483, ISBN: 0262511290.

[3] C. Hartman and B. Benes, "Autonomous boids: Research articles," *Comput. Animat. Virtual Worlds*, vol. 17, no. 3–4, pp. 199–206, Jul. 2006, ISSN: 1546-4261.

[4] S. Karaman and E. Frazzoli, *Sampling-based algorithms for optimal motion planning*, 2011. arXiv: 1105.1186 [cs.RO].

[5] M. El Khaili, "Visibility graph for path planning in the presence of moving obstacles," *Engineering Science and Technology an International Journal*, vol. 4, pp. 118–123, Sep. 2014.