RBE550(Spring 2023) - WildFire

Rohin Siddhartha Palaniappan Venkateswaran

April 3, 2023

The aim of this assignment is to demonstrate path planning with both combinatorial - A* and sampling-based methods - PRM. Similar to the previous assignment, this assignment also involves planning paths in a constrained environment that pose specific kinematic constraints for a truck. To tackle this challenge, state lattices were used to navigate the space while taking into account the unique kinematics of the vehicle. Here, the focus is on an Ackermann drive robot.

Ackermann

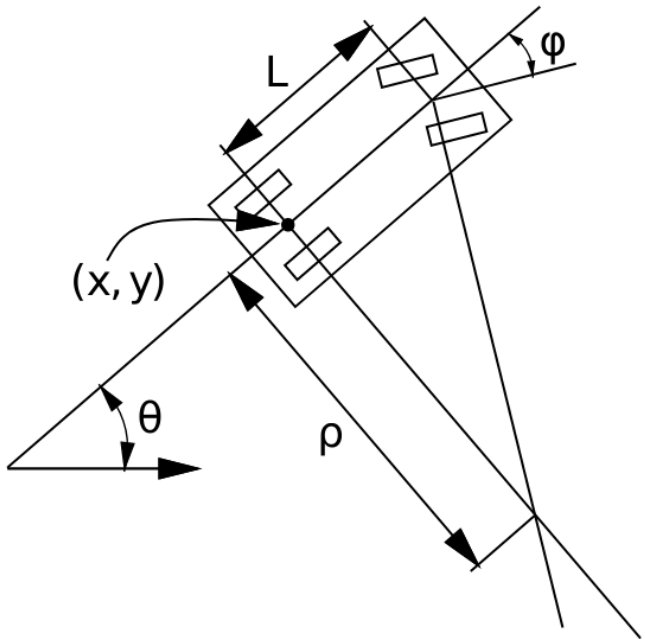


Figure 1: Ackermann Drive Robot

The kinematics consists of velocity v, steering angle ψ. In this assignment, wheelbase $L = 2.8$ meters. Limits of steering angle are also imposed $|\psi| = 60°$.

$$\dot{\theta} = \frac{v}{L}\tan(\psi) \tag{1}$$

$$\Delta\theta = \dot{\theta}\Delta t \tag{2}$$

$$\theta = \theta_0 + \Delta\theta \tag{3}$$

$$\dot{x} = v\cos(\theta) \tag{4}$$

$$\dot{y} = v\sin(\theta) \tag{5}$$

$$\Delta x = \dot{x}\Delta t \tag{6}$$

$$\Delta y = \dot{y}\Delta t \tag{7}$$

$$x = x_0 + \Delta x \tag{8}$$

$$y = y_0 + \Delta y \tag{9}$$

Planner Approach - A*

The standard A* search algorithm has been implemented for the purposes of this assignment. Kinematic time stepping for each step is used to create a state lattice, hence converting the continuous space to discrete space.

A priority queue has been implemented for storing the neighbouring as well as selected states. The cost function is the euclidean distance between states from start to this point, and a theta penalty has also been implemented to ensure an efficient path generation. With regards to the heuristic, the euclidean distance to the goal is penalized by simply multiplying it by a constant.

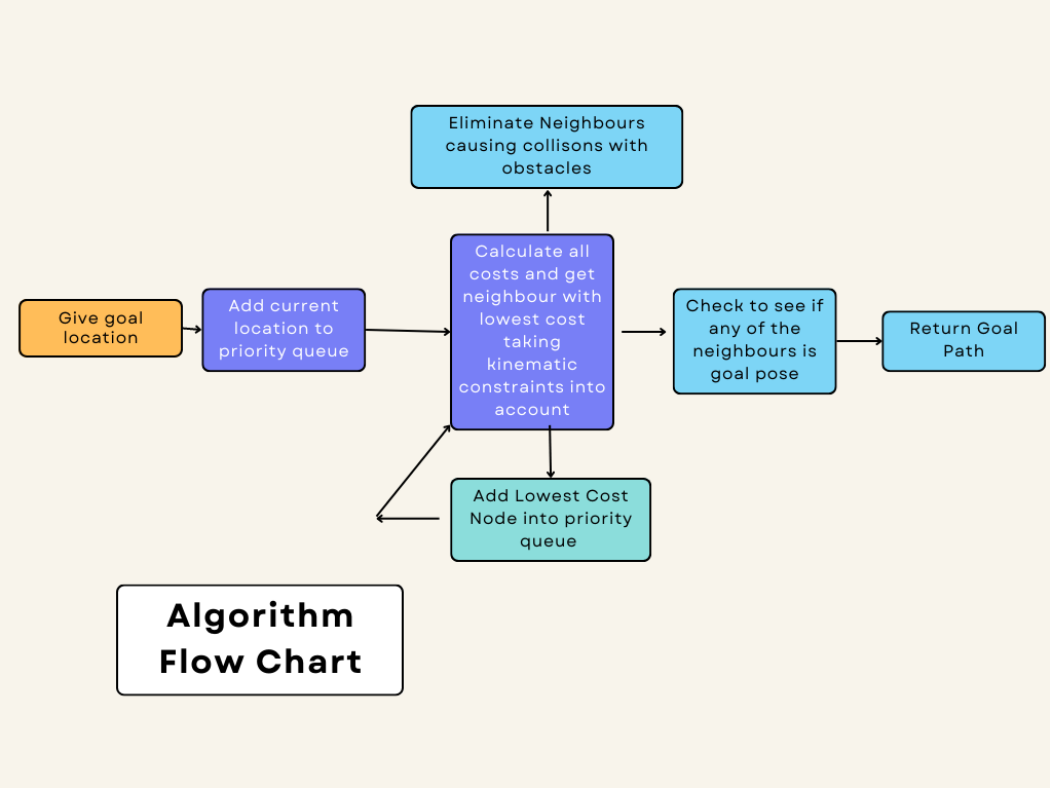


Figure 2: Path planning flow chart

Planner Approach - PRM

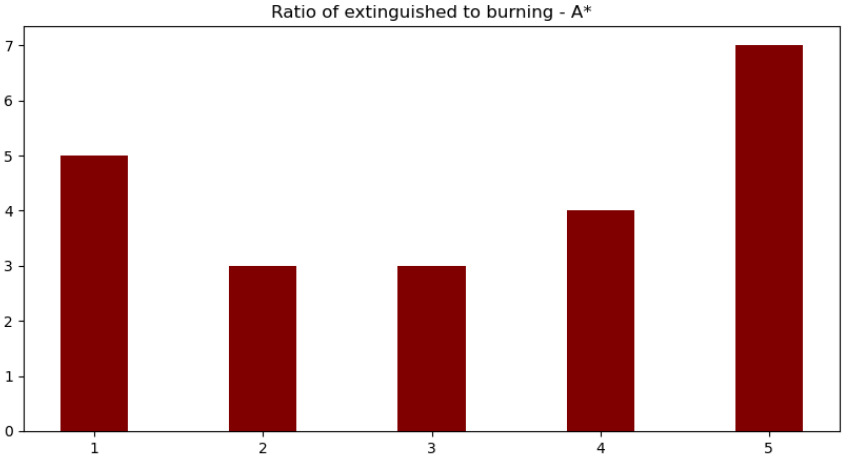
A probabilistic road map (PRM) of the environment was generated for experimentation. Random x and y points within the 0 to 250 meter range was first generated, along with a θ. Then, this configuration was checked for a collision.If not, we add it to the list of points. This experiment was repeated for $10,000$ points.

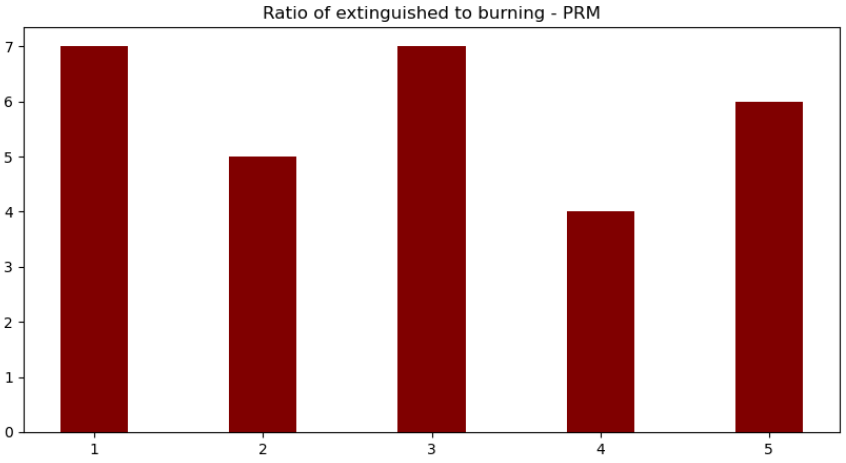
We stored these points in a $KD-Tree$, as suggested in the lecture video. The tree once initialized, does not change, i.e it is immutable.

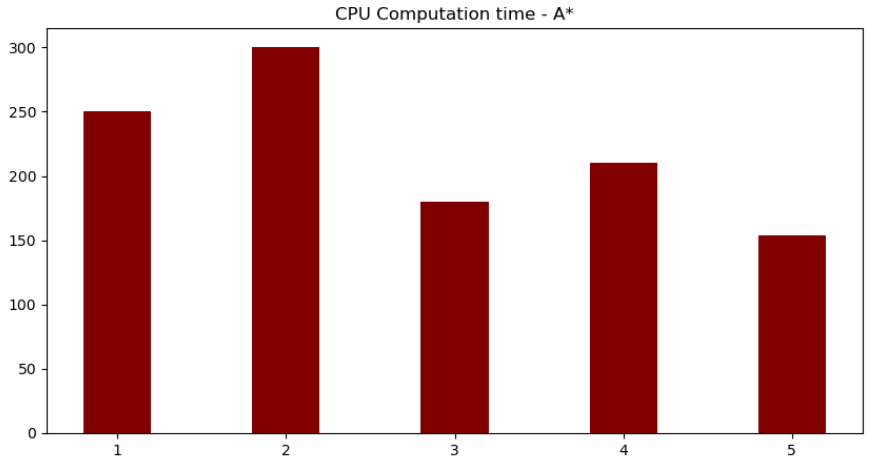
From here, we executed A* across the PRM points, with the assumption that each node held a connection with the 5 closest nodes as decided by the $KD-Tree$. Combined with a euclidean distance heuristic for cost, our planner quickly finds the points between nodes within the PRM.

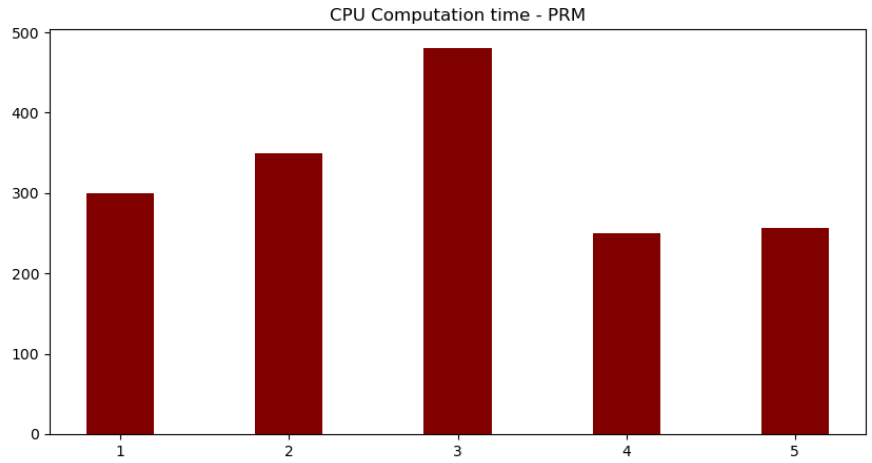
After finding the closest node to a burning obstacle, the PRM A* planner is used to find the path to the current goal location. Further, a second A* planner in which, instead of caclulating from the vehicle's location to the goal, the planner creates a kinematic path between nodes in the PRM.

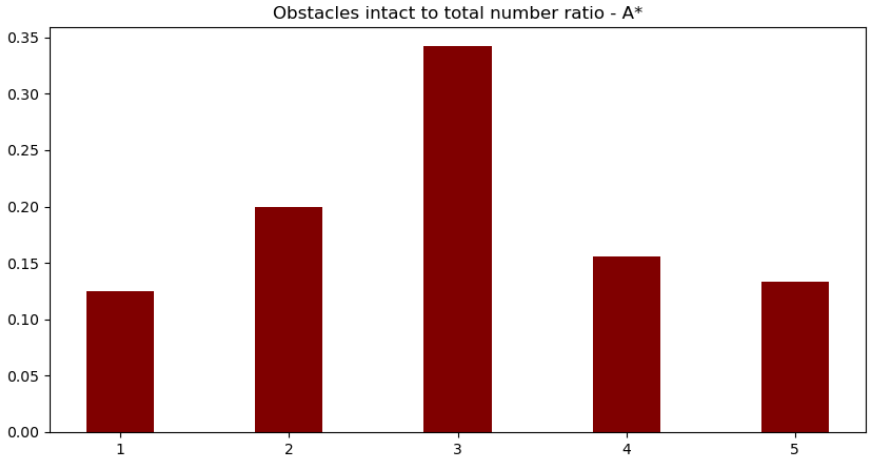
Results and Graphs

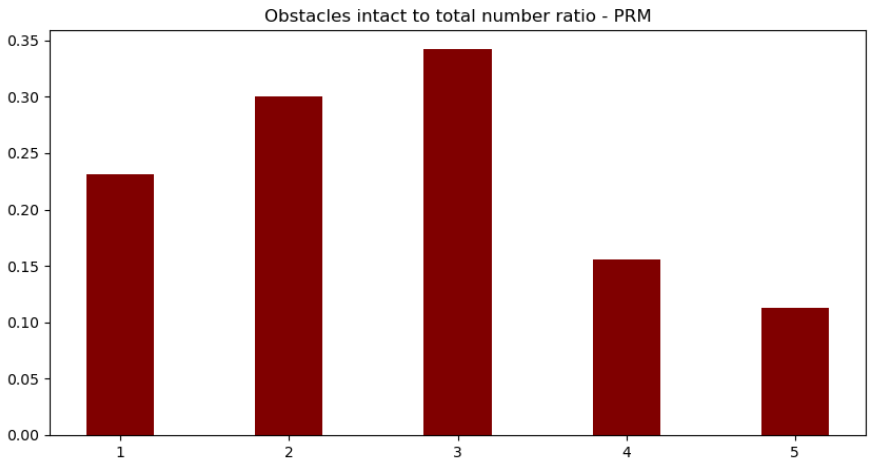












From the above plots, we can conclude that PRM is a better choice for fire fighting but takes more computation time compared to A*.

Instructions to run code

no argument, both algorithms run simultaneously
argument - 1, A* algorithm runs
argument -2, PRM algorithm runs

```
python3 run.py --algorithm 2
```

Final Notes and Conclusions

The obstacles used in this assignment have been inflated during path planning, in order to tackle the collision problem.

Due to simulation and deadline constraints, the wumpus and the firetruck are shown by the same vehicle in the animation videos. First the wumpus lights the fire at a chosen tetromino, then the firetruck plans the path to the chosen tetromino. The wumpus is not shown separately.

"run.py" is the only file which has to be run, give arguments to the file as to run either A* or PRM separately. Otherwise, nor arguments need to be given to the file.

Further, each program is not optimized to run in the fastest time possible, hence, it will take some time to run, depending on the machine on which the code is being run.

This assignment implementation is heavily adapted from https://github.com/hlfshell. Planner implementations as well as certain simulation portions has been directly adapted from this repository. Other components such as obstacle generation, data collection, separate classes for vehicle and obstacle tracking has been written by me.