RRT with 1/2 - Car Dynamics

Rose Gebhardt

March 15, 2022

1 Strategy

For the RRT algorithm, a graph data structure was written in Python. The nodes corresponded to the states of waypoints, including the time the waypoint is reached, the (x, y) position of the waypoint where $x, y \in [-50, 50]$, the orientation defined by $\theta \in [0, 2\pi)$, and the instantaneous linear and angular velocities v, ω with $v \in [-5, 5]$ and $\omega \in [-\pi/2, \pi/2]$. The edges corresponded to the trajectories between waypoints. They store the start and end waypoints, the linear and angular velocities a, α with $a \in [-2, 2]$ and $\alpha \in [-\pi/2, \pi/2]$ which are assumed to be constant over the trajectory, and the time evolution of each element of the state.

The graph is initialized with one waypoint, whose position and orientation are specified in the problem statement. To expand the graph, a point is uniformly randomly sampled from $[-50, 50] \times [-50, 50] \in R^2$, and the existing node with the smallest Euclidean distance to this point is chosen as initial waypoint. A trajectory from this waypoint is then found using the waypoint as an initial state, a constant linear and angular acceleration, and iterating forward in time with Euler's method until an end condition is met.

In order to reduce the chance of exceeding the maximum allowable velocities, the accelerations were enforced to be negative for large positive initial velocities and enforced to be positive for large negative initial velocities. For initial velocities small in magnitude, the accelerations were sampled uniformly from the range of allowable accelerations.

The possible end conditions are colliding with an obstacle, exceeding a maximum velocity, or moving a distance greater than ϵ (given by the problem statement) from the initial waypoint. Euler's method was done using a time-step size $\Delta t = 0.1$ and end conditions were checked at every Δt. Since the maximum allowable speed is 5, this step size ensured that collision detection occurred after moving $\delta \leq 0.5$, as specified by the problem statement.

Two obstacle collision detectors were written. The function `isFreeQuick` treats the vehicle as a circle with radius 1.0 around the center of the vehicle and checks each obstacle for collision. This circle encloses all the points on the vehicle and it eliminates the need for a for loop and additional computations related to the orientation, but it occasionally falsely reports collision. The function `isFree` checks each obstacle and point on the vehicle for collision. It is slower, but does not falsely report collision, which is important for moving through tight regions. For the first four cases, `isFreeQuick` was sufficient to find a path before timing out. The algorithm had to be rerun using `isFree` for the fifth case.

Once a trajectory that reaches the goal region is created, the trajectories explored and the feasible trajectory found are output as CSV files which are used to create the plots below in MATLAB. Additionally, a text file is generated that stores information about the search tree in the form specified by the problem statement. These files are saved as `output_tree_N.txt` in the outputs folder.

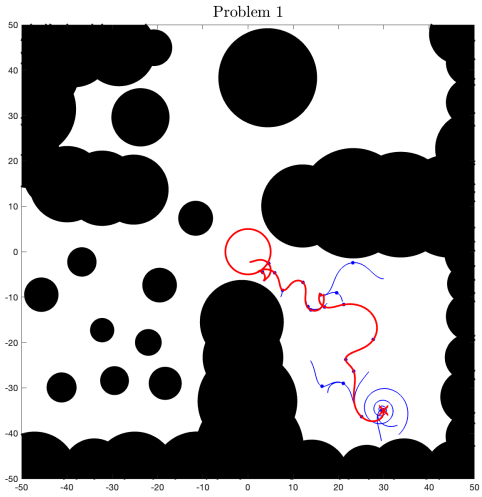
2 Code Structure

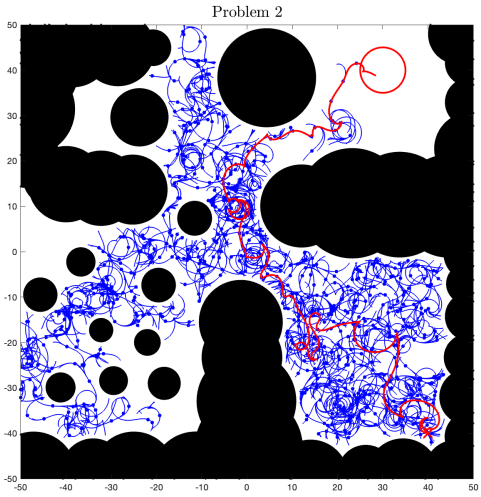
- `H3.txt` contains the problem statements and directions. It can be be disregared when running the code.
- `H3_robot.txt` gives a set of points which specify the geometry of the the vehicle. These points are used for collision detection.
- `obstacles.txt` gives a list of circular obstacles, defined by the Cartesian coordinates of the center and the radius of the obstacle. These values are used for collision detection.
- `RRT.py` is the Python script that implements RRT. It includes the definition of the graph structure, all necessary functions for implementing RRT, and exports the results as text files.
- `results.m` plots the search tree, the feasible path found, the start node, the goal region, and the obstacles. It then saves the figure corresponding to each problem as a `.png` image.

To get the search path and feasible path in a text file, run `python RRT.py` in terminal. To plot these results, run the `results.m` MATLAB script.

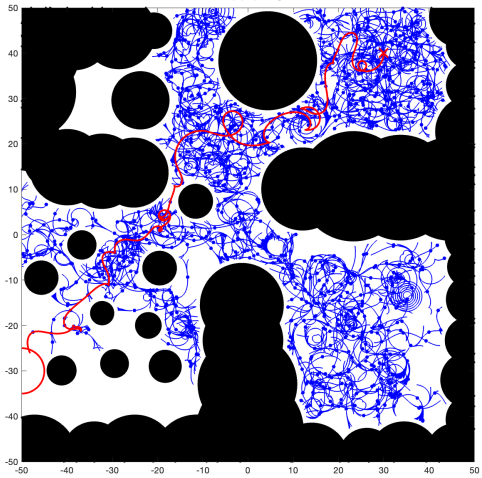
3 Results

Results from each problem are shown below. The red cross indicates the starting position, the red circle indicates the goal region, the areas filled in black are obstacles, the blue line is the search tree, and the red line is the feasible path found. Each case had a solution, but if no solution was found in an allotted number of tries it would be indicated in the terminal when running the Python file.

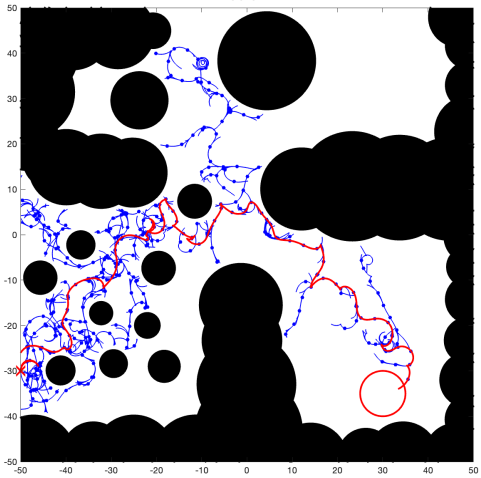




Problem 3



Problem 4



Problem 5