# CSE 571 - Robotics
# Homework 3 - Motion Planning

Due Thursday June 4 @ 11:59pm

The key goal of this homework is to get an understanding of motion planning methods including A*, RRT, and RRT*. There are no written assignments for this homework. For the programming assignment, you will be implementing A*, RRT, and RRT* for 2D motion planning in a grid world, and RRT for a non-holonomic system (car). The zip file containing the code for this homework can be found on the class website (`https://courses.cs.washington.edu/courses/cse571/20sp/`).

*Collaboration:* Students can discuss questions, but each student MUST write up their own solution, and code their own solution. We will be checking code/PDFs for plagiarism.

*Late Policy:* This assignment may be handed in up to 5 days late (Tuesday June 9 @ 11:59pm), at a penalty of 10% of the maximum grade per day.

## 1 Programming Assignments

In this section you will be required to implement different motion planning algorithms and study the parameters that govern their behaviors.

### 1.1 Code Overview

The starter code is written in Python and depends on NumPy and Matplotlib. This section gives a brief overview.

- `run.py` - Contains the main function. Note the command-line arguments that you can provide.
- `MapEnvironment.py` - Environment-specific functions for the 2D point robot. Some of them have to be filled in by you.
- `CarEnvironment.py` - Environment-specific functions for the non-holonomic car.
- `map1.txt, map2.txt, car_map.txt` - Maps that you will work with. The numbers denote the occupancy status of the cell (1 - occupied, 0 - unoccupied).
- `AStarPlanner.py` - A* Planner. Logic to be filled in by you.

- `RRTPlanner.py` - RRT planner. Logic to be filled in by you.
- `RRTStarPlanner.py` - RRT* planner. Logic to be filled in by you.
- `RRTPlannerNonholonomic.py` - RRT planner for non-holonomic car system. Logic to be filled in by you.
- `RRTTree.py` - Contains datastructure that can be useful for your implementation of RRT and RRT*.

## 1.2 2D Point Robot

### 1.2.1 Environment Modeling

The planning consists of a 2D map. You have been provided with two maps `map1.txt` ($9 \times 9$) and `map2.txt` ($473 \times 436$). You can use the former to *test* your implementation but report all results on the latter with start and goal to be [321, 148] and [106,202]. Note that environement-specific functions need to be filled in by you in `MapEnvironment.py` file. Once you complete implementing the environment-specific functions, you are ready to implement the planners.

Note that in case of search algorithms like A*, the environment is considered to be a discrete grid while in sampling-based techniques the environment is assumed to be continuous. However, in this case since the underlying world is given to be grid, you can snap any continuous sample points onto the grid.

The cost of the final path is simply the length.

To run A* on map 2, you would run
```
$ python run.py -m map2.txt -p astar -s 321 148 -g 106 202
```

### 1.2.2 A* Implementation [25 points]

You will be implementing the weighted version of A-star where the heuristic is weighted by a factor of $\epsilon$. Setting $\epsilon = 1$ gives vanilla A*. The algorithm is to be implemented in `AStarPlanner.py` file.

1. Use an 8-connected neighbourhood structure so that diagonal actions are also allowed. Each action has a cost equal to the length of the action i.e. cost of action $(dx, dy) = \sqrt{dx^2 + dy^2}$. Note that diagonal actions cost $\sqrt{2}$.
2. Use the Euclidean distance from the goal as the heuristic function.
3. Try out different values of epsilon to see how the behavior changes. Report the final cost of the path and the number of states expanded for $\epsilon = 1,\ 10,\ 20$. Discuss the effect of $\epsilon$ on the solution quality.
4. Visualize the final path in each setting and the states visited.

#### 1.2.3 RRT Implementation [25 points]

You will be implementing a Rapidly-Exploring Random Tree (RRT) for the same 2D world. The algorithm is to be implemented in `RRTPlanner.py` file. Note that since this method is non-deterministic, you will need to provide statistical results (mean and standard deviation over 10 runs, at least).

1. Bias the sampling to pick the goal with 5%, 20% probability. Report the performance (cost, time) and include figures showing the final state of the tree for both values.
2. For this assignment, you can assume the point robot to be able to move in arbitrarily any direction i.e. you can extend states via a straight line. You will implement two versions of the `extend()` function:
   - the nearest neighbor tries to extend to the sampled point only by a step-size $\eta$. Set $\eta = 0.5$ and report results in your write-up.
   - the nearest neighbor tries to extend all the way till the sampled point (i.e. $\eta = 1$).

   As before, report the performance (cost, time) and include a figure showing the final state of the tree for each setting. Which strategy would you employ in practice?

#### 1.2.4 RRT$^*$ Implementation [25 points]

You will be implementing RRT$^*$ for the same 2D world. You can implement this in top of your RRT planner with consideration for rewiring the tree whenever necessary.

Compare the performance of RRT$^*$ with RRT on points 1 and 2 in Section 1.2.3 (RRT implementation). In summary, make sure to:

1. Bias the goal sampling to 5% and 20%, and compare it with RRT.
2. Set $\eta \in [0.5, 1]$ and compare it with RRT.

Again, report the performance (cost, time) in terms of statistical results (mean and standard deviation over 10 runs, at least), and provide figures showing the final state of the tree for each setting.

### 1.3 Non-holonomic Car

For this problem, we have a car-like system with non-holonomic constraints (i.e the system cannot instantaneously move to any state from any other state). The state of the car is $[x, y, \theta]$ where $(x, y)$ are the coordinates of the center of the car and $\theta$ is the orientation/heading. The controls that the system can apply are $(v, \omega)$ where $v$ is the linear velocity of the car and $\omega$ the steering angle. Note that the car can move both forward and backward as well as turn right and left.

Unlike the previous problem where we extended the graph by moving along the straight line to the random state, we will be sampling controls to generate possible motion "rollouts" from the graph.

Given a randomly sampled state, we will find the closest graph node. Next, in the `extend()` function, we randomly sample control sequences and simulate forward from this graph node and find the one that gets us close to the randomly sampled state. The end-point of this "best" rollout is the next state we add to the graph. More details and pseudocode can be found in this slide deck: `http://people.duke.edu/~kh269/teaching/i400/16dynamicplanning.pptx`.

### 1.3.1 Environment Modeling

You have been provided with a 2D map `car_map.txt` ($400 \times 400$) that you will test your implementation in. This map includes a few rectangular obstacles for the car to avoid. Report all results with the start and goal as [40, 100, $\frac{3\pi}{2}$] and [350, 150, $\frac{\pi}{2}$], respectively. Note that environment-specific functions have been provided in `CarEnvironment.py` such as dynamics simulation, action sampling, and distance computation. Unlike the 2D point robot problem, all implementation for the environment has been provided, and you may investigate the code for your own understanding.

The cost of the final path is the (simulated) time taken to execute the trajectory. See the `simulate_car()` method for more details.

To run the non-holonomic version of RRT, you would run

```
$ python run.py -m car_map.txt -p nonholrrt -s 40 100 4.71 -g 350 150 1.57
```

### 1.3.2 RRT Implementation [25 points]

You will be implementing RRT for this question. Note that since this method is non-deterministic, you must provid statistical results (mean and standard deviation over 10 runs, at least). We provide an example solution in Figure 1. Make sure to read the provided comments in the `extend()` method in detail before implementing the code.

- Bias the sampling to pick the goal with 5%, 20% probability. Report the performance (cost, time) and include figures showing the final state of the tree for both values.
- Try implementing the distance function $d(s_1, s_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + \alpha(\theta_1, \theta_2)^2}$, where $\alpha(\theta_1, \theta_2) = |\theta_1 - \theta_2| \frac{180}{\pi}$ is the angular difference in degrees. Run the planner a few times with this distance function and different seeds. What are the downfalls of this distance function? How does it compare (cost, time) to the provided distance function? Investigate the results and compare them to the provided distance function.

## 1.4 Notes

- Please do not import any extra libraries. This will break the autograder.

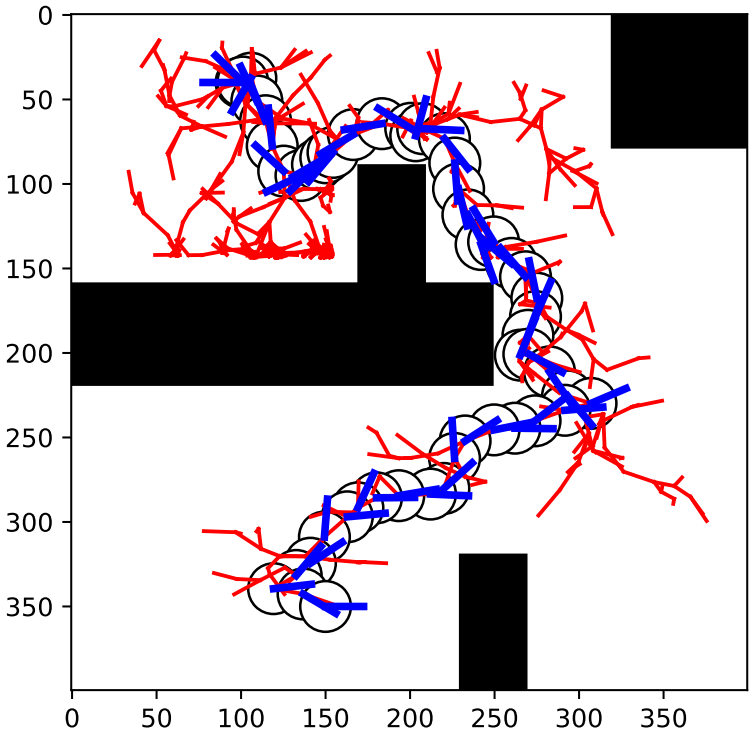


Figure 1: Example RRT solution of non-holonomic car system.

- For the non-deterministic algorithms (RRT/RRT*), do NOT write any code for sampling (e.g. calls to `np.random`). We have provided all sampling methods (location sampling, action sampling, etc) required in the environment code and planning code for this. Please use the provided sampling methods. Otherwise, the autograder will detect incorrect results for your submission.
- Please read all function/method comments before starting to code. These will provide you with the required input/output data types.

## 2 Submission

You will be using Gradescope `https://www.gradescope.com/` to submit the homework. You will be submitting your PDF with analysis and plots under "HW3 Written". For the code, please submit: **MapEnvironment.py, AStarPlanner.py, RRTPlanner.py, RRTStarPlanner.py, and RRTPlannerNonholonomic.py** to "HW3 Code".