Homework 1: Search Algorithms

Advanced Robotics: Manipulation Algorithms

CSE 599 R1, Fall 2017

Due Date: October 25, 2017

TA: Aditya Vamsikrishna (adityavk@uw.edu)

1 Introduction

In this homework you will be exploring the use of discrete planners for three different configuration spaces. You will implement an A-Star Planner on two/three dimensional configuration spaces. You will then move to the higher dimensional space of the WAM arm on the HERB robot.

2 Discretizing the Space

In order to implement discrete planners we will need a mapping from continuous configuration space to discrete space. In this part of the homework you will implement this mapping. Note that for 2D and 3D configuration spaces, a discrete world has been provided to you as a .graphml file. However you will still be required to use the following functions since the input configurations of start and goal need not be a part of the discrete world.

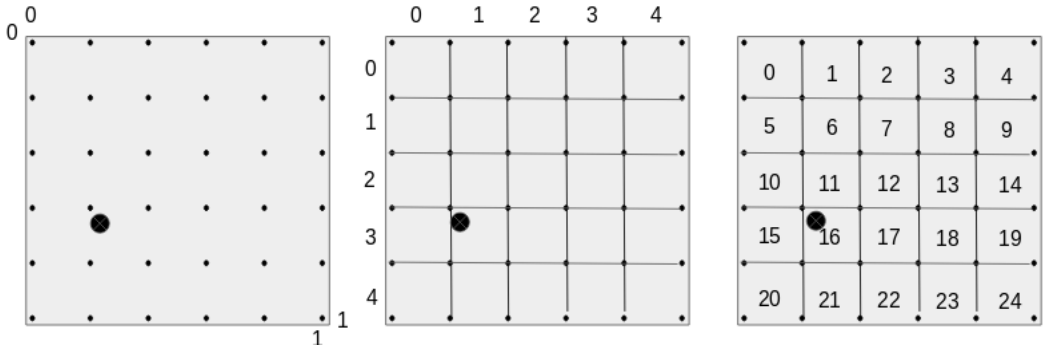

Figure 1: Representation of a point in continuous and discrete worlds

A point in your configuration space can be represented in three ways:

1. Continuous Configuration: Represent the point in the continuous space (ex: [0.21, 0.62])
2. Grid Coordinate: Discretize the space into grids and assign each point the appropriate grid (ex: [1,3])
3. Node ID: Each grid ID can be assigned a node ID (ex: 16)

Refer to Figure 1 for an illustration.

Implement this Mapping in the DiscreteEnvironment.py file. The functions have already been provided to you, however, you will need to implement the logic for each of them. Implement the functions using the resolution parameter.

3 Code

You will find a file named run.py. The following command should show you the command line arguments you can pass:

python run.py –help

4 Lower Dimensional Spaces

In this part, you will implement the planner on two and three dimensional configuration spaces. The obstacles have been defined in the obstacles folder for the configuration spaces. The obstacles are hypercubes defined with the lower extreme and upper extreme corner points. For instance, for a rectangular shaped obstacle in a 2D space the bottom left and the top right corners are defined in the file as (x_lower y_lower x_upper y_upper). Run the following command to generate the default plan which is a straight line between the start and goal. Your goal in this part is to implement the A-Star planner to generate a collision-free shortest path in the discrete world.

4.1 Simple Environment

You have been provided with the graphml files that define the grid world you will be working with. Note that the resolution of this grid world is 0.025. It is recommended that you skim the graphml file to understand how the grid is structured. Before designing the planner you will implement the functions in this file to query the graph appropriately.

1. *get_successors* - Returns list of neighbors for the node represented by given ID.
2. state_validity_checker - Checks the validity of a state by performing a collision check.
3. edge_validity_checker - Checks the validity of an edge by representing it as a sequence of states and checking each of those states for collision.
4. *compute_distance* - Computes the distance between two nodes.
5. *get_heuristic* - Computes the heuristic cost between two nodes with the given IDs.

5 Planner

You will implement the A-Star planner for this assignment. Implement the planner in the *Plan* function in the AStarPlanner.py file. Visualize the planner and see how the edges are being selected to ensure that your planner does as expected and return the shortest path.

6 7-DOF manipulator

Once you have a working version of the algorithm on the lower dimensional spaces, you should be able to implement the same on HERB. Running the following command will execute a default plan which connects the start to goal using a straight line.

python run.py -r simple

You will notice that the arm goes through the table. Your goal for this part is to use the planner to execute a trajectory while avoiding the obstacles.

6.1 HERB Environment

The file HerbEnvironment.py implements 7DOF WAM arm configuration space. Implement the following three functions in this file:

1. *get_successors* - Returns list of neighbors for the node represented by given ID. You can choose how to define a neighbor. One recommendation is to change only one joint at a time.
2. state_validity_checker - Checks the validity of a state by performing a collision check.

3. edge_validity_checker - Checks the validity of an edge by representing it as a sequence of states and checking each of those states for collision.
4. *compute_distance* - Computes the distance between two nodes.
5. *get_heuristic* - Computes the heuristic cost between two nodes with the given IDs.

Note: Generating these plans will take a good amount longer than for any of your previous planners.

7 Extra Credit

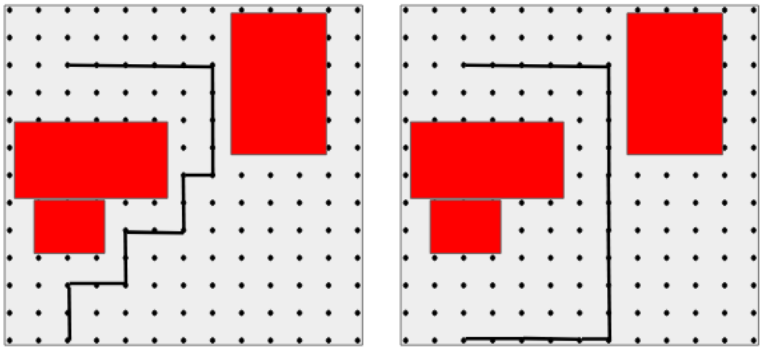

Figure 2: Mode Switches

1. You will notice that the planner can potentially return a shortest path with many mode switches as shown in Fig.(2). In many instances, for environments that are modelled as grids with uniform costs, it is not desirable for the shortest path to have multiple mode switches. Your goal is to determine the shortest path with minimal mode switches.
2. In some environments, there can be regions of the configuration space which are not obstacles but risky. For instance consider the case of road navigation. If there are two routes from the source to the goal, say, the longer route guarantees GPS at all times while the shorter route does not for a part of the route. The longer you travel without GPS, the greater the risk of losing your way. Given this risk, which route should your planner suggest you to take? This can be modelled as a variant of the algorithm you have already implemented. Refer to this paper for more information about an algorithm that addresses this problem.

   Use the same graph as was used for 2D configuration space with the simple obstacles defined in the file riskmap.txt under the obstacles folder. The first entry in each line defines obstacle (0) or risk region (1). The rest of the entries are as before. Use the exponential cost assuming each edge to have a cost of 0.1 instead of the euclidean distance between vertices. Compute the shortest path between the points (0.08,0.71) and (0.91,0.53). You can use most of the code from the previous files except for the planner. You can copy the file contents of AStarPlanner.py into a RAMP.py file and rewrite the Plan method.

8 Deliverables

- Please turn in a zip file (named hw1_<UW ID>.zip) containing your code, a PDF writeup and the videos mentioned below. Make sure you mention your name in the report as well. The write-up needs to be in NIPS format. You should be able to download the required files here. The deliverables for this homework are:
  - Run the A-Star planner for 2D and 3D configuration spaces with heuristic weighted at 0, 1 and 5 (6 runs in total). Report path length, plan time, number of nodes expanded and the number of edges checked for collision in each case. Report your understanding about the weighted heuristic and submit a visualization as the search progresses on the 2D problem. [10 pts]

- Run the A-Star planner for the WAM arm with a resolution of 0.1 and step_size of 0.05 and 0.03 for checking the validity of edges and heuristic weighted at 1 and 5 (4 runs in total). Report path length, plan time, number of nodes expanded and the number of edges checked for collision. Submit a video of the run. [10 pts]

- (Extra Credit) Run the A-Star planner minimizing the mode switches on the 2D and WAM configuration spaces. Submit a video of one of the runs on WAM and a visualization of the final path for the 2D problem. [5 pts]

- (Extra Credit) Run the RAMP algorithm on the 2D example and report the shortest path. What would have been the shortest path if there was no risk region? [5 pts]

Note: The functions given in SimpleEnvironment.py and HerbEnvironment.py are simply a suggestion to help guide your implementation. You can change these functions in any way you see fit (example: different inputs and outputs). However, leave the interface for the Plan method intact.