ROB 534: Sequential Decision Making in Robotics, Winter 2023
HW #1: Discrete and Sampling-based Planning
Due: 2/1/23 (midnight)

Questions

1. Search-based planning (20 pts)

(a) Consider the planning problem shown in Figure 1. Let '1' be the initial state, and let '6' be the goal state.

(i) By hand, use backward value iteration to determine the stationary cost-to-go (i.e, minimum cost between each state and the goal). **Show your work.**

(ii) Do the same but instead use forward value iteration. **Show your work.**

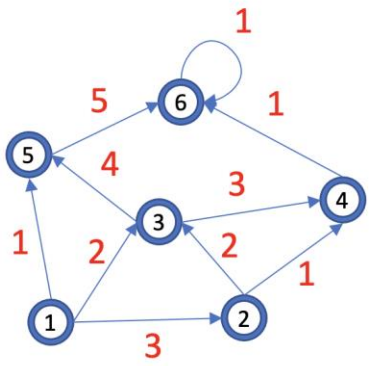
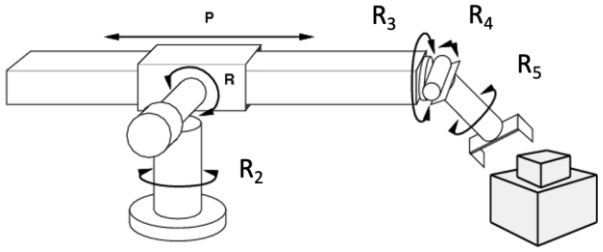


Figure 1: Six-state discrete planning problem.

(b) Consider the 6-DOF planning problem below. The goal is to get the end effector from a starting configuration (defined by five rotation angles R and one linear dimension P) to an end configuration (also defined in the 6D space). The 6D space is discretized, and the cost metric is defined as: $C = \Delta R + \Delta R_2 + \Delta R_3 + \Delta R_4 + \Delta R_5 + \Delta P$, where ΔR is the total angular distance travelled in the R degree of freedom, ΔP is the total linear distance travelled, etc.



Which of the following heuristics is admissible for A* search on this problem?

i. Euclidean distance in 6DOF space from start to goal

ii. Euclidean distance in 6DOF space from start to goal times 2

iii. Euclidean distance in 6DOF space from start to goal divided by 2

v. Sum of total 6DOF distance travelled by optimal solution to reach goal state

(c) For each heuristic that you deemed admissible in Part (b), order them from least informed to most informed. **Explain your answers.**

Programming Assignment (Do not start at last minute!) (60 points)

The zip files on canvas contain helper functions and data for this assignment. Download them in addition to this word document. You may use any programming language you wish to complete the homework. Helper functions are provided in the zip file if you choose to use Matlab or Python. The text below is written describing the Matlab code; you should substitute the appropriate functions in Python if you use those functions.

In your Matlab program, nodes should be opaque data structures with operations get_start.m, get_goal.m, and get_neighbors.m. Restrict the number of nodes expanded to 10,000 to avoid very long run times.

To make your job easier, we have provided a Matlab priority queue implementation in the archive. You are free to use this implementation or not, as you see fit. The priority queue has the following functions:
• pq init: Initialize a priority queue.
• pq set: Reset the priority of an element or insert it if it's not already there.
• pq pop: Remove and return the first element. (The first element is the one with the smallest numerical priority value). It breaks ties arbitrarily.
• pq test: Check whether an item is already on the priority queue.

We have also provided two mazes (maze1.pgm and maze2.pgm) along with helper functions to read the mazes, convert from state (x,y) to index, plot paths, find neighbors and other operations. Familiarize yourself with the provided code before getting started.

Step 1 (10 points): Before implementing any algorithms, draw a flowchart of the A* and RRT algorithms. These flowcharts should have text bubbles describing each step with arrows connecting them. Decisions should also be represented as branches with labels (e.g., yes/no). Include the flowcharts in your report.

Step 2 (30 points):

i. **Implement A* with an admissible heuristic. Which heuristic did you use? Provide an image of the path found by your A* implementation on both mazes.** Have the starting node be the top left cell: (x, y) coordinates (1, 1), and have the ending cell be the bottom right cell: (x,y) coordinates (Cols,Rows). Use the total distance traversed by the robot as the cost function.

ii. Allow A* to multiply your admissible heuristic by a constant value epsilon (greedy A*). **Set up a program that does the following:**

1. A* runs with a user-provided epsilon
2. Once search is completed it then sets a new_epsilon = epsilon – 0.5*(epsilon-1). If new_epsilon is less than 1.001, it sets new_epsilon = 1.
3. A* runs again with the new epsilon value

4. The loop continues with deflating epsilon until either (a) a running time limit is reached or (b) the search has completed with epsilon = 1.

Use a starting value of epsilon = 10. Record for running time limits 0.05 seconds, 0.25 seconds, and 1 second (on both maps and for each epsilon value where the search completed): number of nodes expanded and path length.

iii. **Implement the RRT algorithm to solve the 2D problem in continuous space on maze1.pgm and maze2.pgm [1].** You may use the knnsearch function in the statistics toolbox for nearest neighbors if using Matlab. Hint: you should sample the goal periodically to get goal directed behavior (do not use fully uniform sampling). You may assume the goal is reached if the x and y values are both within 1 of the goal. You may also assume the robot is modeled as a point. The provided check_hit.m gives an obstacle test. **Provide an image of a path from the RRT algorithm on both maps. Also provide the path length and running time.**

Step 3 (20 points): For this problem, we will expand the state space to 4D to incorporate the dynamics of the agent moving through the world. The agent's state will now consist of (x,y) coordinates as well as (dx,dy) velocities. Instead of controlling its movement directly, the agent now controls its acceleration and deceleration. We have restricted the problem as follows:

- The agent starts in the top left corner (x,y) = (1,1) with zero velocity (dx, dy) = (0, 0).
- The agent must move to the bottom right corner (x,y) = (Rows,Cols) and slow down to zero velocity.
- The agent can accelerate or decelerate by one in either dx or dy (or remain at the same velocity).
- The agent's velocity in both directions must always remain less than or equal to a maximum velocity provided in the variable maxV = 2. Both velocity components must also always remain positive.
- The agent cannot slow down by hitting an obstacle. If it is traveling at a high speed, it must slow down several squares before an obstacle to avoid collision. (You do not want to scratch your car on those pesky obstacles).

For example, if the agent is in state (x,y,dx,dy) = (1,1,2,0), it can choose to speed up dy, slow down dx, or remain at the same speed. It cannot speed up dx because it would break the speed limit, and it cannot slow down dy because it would be moving backwards. If it speeds up dy, it goes to (x, y, dx, dy) = (3,2,2,1). If it slows down dx, it goes to (x,y,dx,dy) = (2,1,1,0). If it remains at the same speed, it goes to (x, y, dx, dy) = (3, 1, 2, 0). For more clarification, see the file get_neighbors_dynamic.m in the archive.

The 4D problem uses the same maps as before, but they are now read in using the read_map_for_dynamics.m function. We have also provided you with the functions test_dynamic_neighbors.m and test_dynamic_path.m to help familiarize you with the format of the maps in 4D space.

For this problem, we have provided you with Matlab code for the functions get goal_dynamic.m, get_start_dynamic.m, and get_neighbors_dynamic.m. These will replace the corresponding functions which you implemented for the 2D problem. The functions dynamic_state_from_index.m and dynamic_index_from_state.m also provide the same functionality as the corresponding 2D functions.

i. **Replace the A* functions that you coded up for the 2D maze problem above, and run your search algorithms on the 4D problem to minimize time (not distance). Develop an informed and admissible heuristic for the 4D domain (you only need one). What was your heuristic? Provide an image of your path on both mazes.**

ii. **Apply your deflating heuristic program to the 4D problem. Use a starting value of epsilon = 10. Record for running time limits 0.05 seconds, 0.25 seconds, and 1 second (on both maps and for each epsilon value where the search completed): number of nodes expanded and path length.**

Discussion Questions (20 points): Address these questions (1-2 paragraphs each):
1) In this exercise, the collisions were easy to compute. Consider a version of the 4D problem where the collision cost dominates the computation time. How would you modify your algorithm to deal with this case? Hint: Look at the Lazy A* paper [1].
2) Consider using RRT-Connect on the 4D problem. What advantages would this have over standard RRT? What challenges would it lead to that would need to be overcome? Hint: Review the RRT*-Connect paper [3].
3) What modifications to the A* algorithm would you make if the robot discovered the environment as it went along (i.e., obstacles appeared and disappeared when the robot was near them)? You can assume the world itself is static, but the robot discovers the world as it moves. Do not provide full pseudocode, just a high-level description. Hint: Review the D*-Lite algorithm [4].
4) What modifications to the RRT algorithm would you make if the robot's position were uncertain (partially observable)? Do not provide full pseudocode, just a high-level description. Hint: Review the RRBT algorithm [5].

References

[1] A. Mandalika et al., "Lazy Receding Horizon A* for Efficient Path Planning in Graphs with Expensive-to-Evaluate Edges," ICAPS 2018.
[2] R. Mashayekhi, M.Y.I. Idris, M.H. Anisi, I. Ahmedy and I. Ali. "Informed RRT*-connect: An asymptotically optimal single-query path planning method." IEEE Access, vol. 8, pp. 19842-19852, 2020.
[3] S. Koenig and M. Likhachev, "D* Lite," Proc. AAAI/IAAI, Vol. 15, 2002.
[4] A. Bry, and N. Roy. "Rapidly-exploring random belief trees for motion planning under uncertainty." Proc. IEEE Conference on Robotics and Automation (ICRA), 2011.