Project 2: Motion Planning

Collaboration in the sense of discussion is allowed, however, the assignment is individual and the work you do should be entirely your own. See the collaboration and academic integrity statement here: https://natanaso.github.io/ece276b*. Books, websites, and papers may be consulted but not copied from. It is absolutely forbidden to copy or even look at anyone else's code.* ***Please acknowledge in writing people you discuss the problems with.***

Submission

Please submit the following files on **Gradescope** by the deadline shown at the top right corner.

1. **Programming assignment**: upload all code you have written for the project and a README file with a clear, concise description of the main file and how to run your code.

2. **Report**: upload your report in pdf format. You are encouraged but not required to use an IEEE conference template[1] for your report. Please refer to Slide 12 and 13 of ECE 276A Lecture 1 for the expected report structure and contents[2].

Problems

In square brackets are the points assigned to each part.

1. **Required Part.** This project requires you to develop a search-based motion planning algorithm to intercept a moving target. The main function of your planner should be `robotplanner.py`. Currently, the file contains a greedy planner that always moves the robot in the direction that decreases the distance between the robot and the target. The target planner tries to maximize the minimal distance the robot can achieve using a minimax decision rule[3]. The `robotplanner.py` function takes 3 inputs (envmap, robotpos, targetpos), where envmap is a map of the obstacles (envmap(x,y) = 0 is free, envmap(x,y) = 1 is occupied), robotpos is the current 2D position of the robot, and targetpos is the current 2D position of the target. The function should return the next position of the robot among the 8 cells adjacent to the current robotpos cell (including diagonally adjacent cells). The returned position cannot be an occupied cell or outside of the map boundaries (see the current `robotplanner.py` implementation to see how validity of the next robot position may be tested).

   Your planner is supposed to produce the next robot move within **2 seconds**. Within those 2 seconds, the target also makes one move but it can only move in four directions (no diagonal moves). If your planner takes longer than 2 seconds to plan, then the target will move by a longer distance. In other words, if the planner takes 2N seconds (rounded up) to plan the next move of the robot, then the target will move by N steps in the meantime. The file `main.py` contains examples of running the robot and target planners on several provided map files.

   Executing `main.py` will show that sometimes the robot does catch the target, and sometimes it does not. The robot position is indicated by a blue square, while the target position is indicated by a red square. Note that to evaluate the performance of your algorithm we might use different starting positions, a different map from the provided ones, or a different strategy of how the target moves (i.e., a different target planner). The only assumption you can make is that the target will only move in four directions. Note that your algorithm will be judged by its design, rather than how optimized its implementation is.

   Optional Part. After implementing the required search-based planning algorithm, if time allows and you are interested in trying alternative methods, you may compare your algorithm to a sampling-based motion planning algorithm. You may use the state-of-the-art sampling-based motion planning library OMPL (https://ompl.kavrakilab.org/). Various demos and tutorials are available on the OMPL

[1] https://www.ieee.org/conferences_events/conferences/publishing/templates.html
[2] https://natanaso.github.io/ece276a/ref/ECE276A_1_Introduction.pdf
[3] https://en.wikipedia.org/wiki/Minimax

website[4]. While OMPL is preferable, it might be challenging to install. Alternatively, you may use the Python motion planning library at https://github.com/motion-planning/rrt-algorithms. Examples are available in the repository.

Report and Code Submission. Write a project report describing your approach to the moving-target motion planning problem. Include the following sections:

(a) [5 pts] **Introduction:** provide a short introduction to the problem, its potential application to robotics, and your approach at a high level.

(b) [10 pts] **Problem Statement:** state the problem you are trying to solve in mathematical terms, including the elements of the shortest path problem.

(c) [35 pts] **Technical Approach:** describe your approach to search-based planning and the key ideas that enable your planner to scale to large maps and to compute solutions within alloted time constraints.

(d) [20 pts] **Results:** present any interesting details in your implementation and discuss your results (e.g., path length statistics for the performance of your algorithm, visualizations of the computed paths, etc.) – what worked as expected and what did not and why.

In addition to the report, your submission should include the code you have written to solve the moving-target motion planning problem:

(e) [30 pts] **Code:** your code will be evaluated based on *correctness* and *efficiency*. Correctness refers to whether your code implements the algorithm you present in your report and whether the algorithm you present actually solves the problem you formulated. Efficiency refers to your code being able to produce and execute paths for the agent in reasonable time. The results presented in the report should be consistent with the output of your code. **Copying, rephrasing, or even looking at anyone else's code is strictly forbidden. Writing your own code is the best way to avoid plagiarism.**

[4]Tutorials: https://ompl.kavrakilab.org/tutorials.html; Demos: https://ompl.kavrakilab.org/group__demos.html. Two useful examples can be found here: https://ompl.kavrakilab.org/OptimalPlanning_8py_source.html, https://ompl.kavrakilab.org/RandomWalkPlanner_8py_source.html