Homework 2: Motion Planning

Collaboration in the sense of discussion is allowed, however, the work you turn in should be your own - you should not split parts of the assignments with other students and you should certainly not copy other students' code or papers. See the collaboration and academic integrity statement here: https://natanaso.github.io/ece276b. *Books may be consulted but not copied from.*

Submission

You should submit the following four files on **Gradescope** by the deadline shown on the top right corner.

1. **FirstnameLastname_HW1_P1-3.pdf**: upload your solutions to Problems 1-3. You may use latex, scanned handwritten notes (write legibly!), or any other method to prepare a pdf file. Do not just write the final result. Present your work in detail, explaining your approach at every step.

2. **FirstnameLastname_HW1_P4.zip**: upload the code you have written for Problem 4 in a zip file with the following strucutre:

   ```
   src/
   main.py
   README.txt
   ```

   The src directory should contain all files necessary to run your code, including the `input_*.txt` and `coords_*.txt` files. The README file should contain information about the python version and dependencies needed. The `main.py` file should run your program with the following syntax: "python main.py" and should produce the two output files: `output_costs.txt` and `output_numiters.txt`.

3. **FirstnameLastname_HW1_P5.zip**: upload the code you have written for Problem 5 in a zip file with the following strucutre:

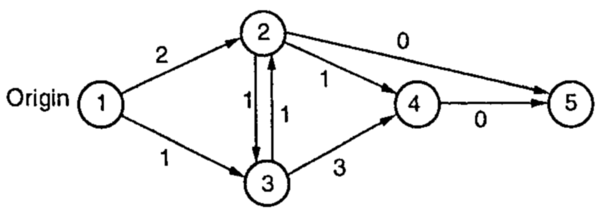
   ```
   src/
   robotplanner.py
   targetplanner.py
   main.py
   README.txt
   ```

   The src directory should contain all files necessary to run your code, including the `map*.txt` files. The README file should contain information about the python version and dependencies needed. The `main.py` file should run your program with the following syntax: "python main.py" and should visualize the movement of the robot and the target.

4. **FirstnameLastname_HW1_P6.pdf**: upload the report for Problem 6. You are encouraged but not required to use an IEEE conference template[1] for your report.

Problems

1. [5 pts] Find a shortest path from node 1 to node 5 for the graph shown below by using the label correcting algorithm. The transitions costs are indicated above the edges.



[1] https://www.ieee.org/conferences_events/conferences/publishing/templates.html

2. [5 pts] Suppose a triangular robot is operating in a 2D workspace. The robot can translate in any direction but cannot rotate. The configuration of the robot is specified by the position of the leftmost vertex of the triangle (marked by a circle in Fig. 1). The workspace has three polygonal obstacles. Our goal is to find the shortest path from the initial configuration $(0, 0)$ of the robot (marked in red) to the goal configuration $(10, 10)$ (marked in green) while avoiding the obstacles. We will do it by hand.

   (a) Draw the configuration space for this problem (either by hand or by writing a program to do so). Make sure to mark the start and goal configurations as well as the inflated workspace obstacles.

   (b) Find the minimum length path from the starting configuration to the goal configuration while avoiding the C-space obstacles (either by hand or by writing a program to do so).

   Make sure your submission includes a plot of the C-space obstacles and the shortest path.

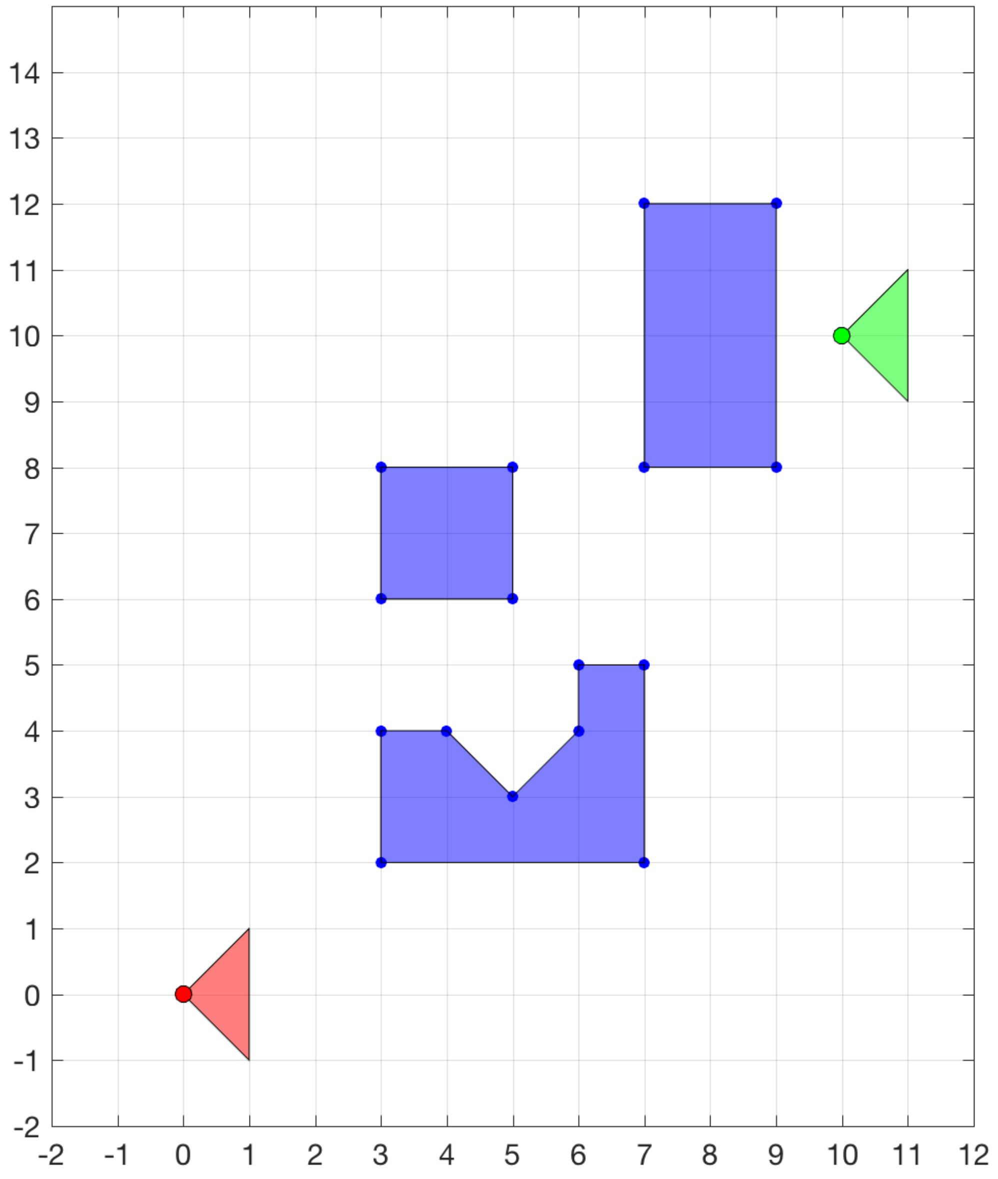


Figure 1: Workspace for a translating triangular robot. The start position is shown in red and the goal position is shown in green. The three workspace obstacles are shown in blue.

3. [5 pts] Consider a graph whose nodes represent the coordinates of a 2-D integer grid. Let G_4 be a graph for which each node has edges to at most four of its neighbors (N,W,S,E) with N representing the North neighbor, W – the west neighbor, and so on. Let G_8 be a graph for which each node has edges to at most eight of its neighbors (N,NW,W,SW,S,SE,E,NE). The cost of an edge is the Euclidean distance between the two nodes, e.g., 1 for N and $\sqrt{2}$ for SW. The cost of a path is the sum of the costs of the edges along the path. Recall the definition of an admissible (i.e., under-estimate) heuristic. A heuristic function, h, is *admissible* if for any node i, h_i is less than or equal to the cost of the optimal path starting from i and terminating at the goal τ. Consider a heuristic function based on Manhattan distance:

$$h_i = |x_i - x_\tau| + |y_i - y_\tau| \tag{1}$$

where (x_i, y_i) are the grid coordinates of node i and (x_τ, y_τ) are grid coordinates of the goal node τ.

   (a) Is h an admissible heuristic for G_4? If yes, justify. If not, give a counter-example of h leading to over-estimate.

   (b) Is h an admissible heuristic for G_8? If yes, justify. If not, give a counter-example of h leading to over-estimate.

4. [25 pts] **Programming Assignment** Implement and compare the performance of Dijkstra, A*, and Weighted A* for solving the shortest path problem on a graph. In this problem, we will consider only 2-D grid graphs in which the edge costs are the distances between the associated nodes. Each node can be connected to at most eight of its neighbors (N,NW,W,SW,S,SE,E,NE).

   Input Description. You are given three graphs that represent the same environment but with increasing grid resolutions (Fig. 2). Each graph is specified by two input files: (1) the `input_*.txt` file specifies the input graph along with the start node and goal node in the same format as Problem 5 in Homework 1; (2) the `coords_*.txt` file lists the x and y coordinates of the nodes starting with the node numbered 1 on the first line. For example, `input_1.txt` contains 99 nodes and `coords_1.txt` specifies the actual coordinates of these 99 nodes, sequentially from node 1 to node 99. The node coordinates can be used to compute a heuristic (Euclidean distance to the goal node) and visualize the graphs via the provided `viz_graph.py` file. You can also use `viz_graph.py` to display the paths produced by your algorithms.

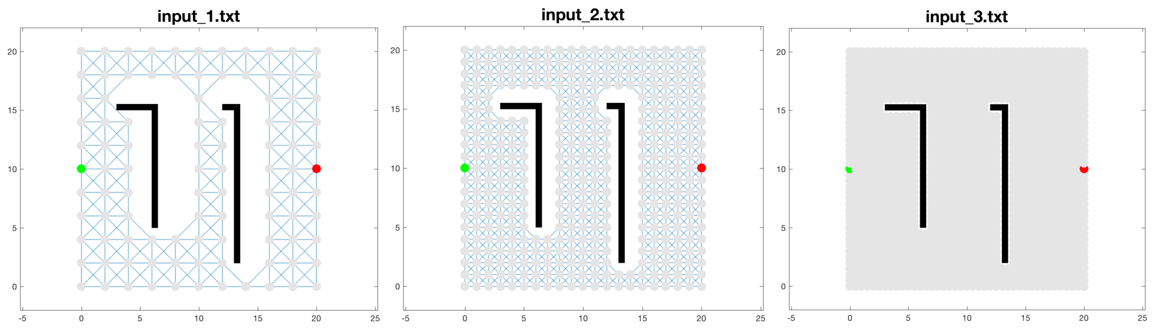


Figure 2: The three input files represent the same grid environment with increasing grid resolutions. The start (green) and the goal (red) nodes are at the same grid position in all three graphs but have different node numbers since the graph sizes are different.

   Output Description. Your implementation must use a single gateway script `main.py` as described in the submission instructions. Calling `main.py` should reads in all six input files (saved in the `src` directory) and produce two output files named `output_costs.txt` and `output_numiters.txt`. Each output file must contain a 3×6 table (i.e, 3 lines with 6 space delimited numbers) with the first row (i.e., first line of the file) corresponding to `input_1.txt`, the second row for `input_2.txt`, and the third row for `input_3.txt`. Each line must list six space-delimited numbers: for `output_costs.txt` these should give the cost of the paths returned by Dijkstra, A* with Euclidean distance as heuristic, weighted A* with $\epsilon = 2$, weighted

A* with $\epsilon = 3$, weighted A* with $\epsilon = 4$, and weighted A* with $\epsilon = 5$; for `output_numiters.txt`, the six numbers should correspond to the number of iterations taken by the algorithms to find the output path, listed in the same order. The number of iterations refer to the number of times we remove an entry from the OPEN list which is also the same as the size of the CLOSED list at the algorithms' termination.

Note that instead of writing six separate functions, you can just write one function which can be configured to execute all six algorithms.

5. [45 pts] **Programming Assignment** Your task is to develop a motion planner that intercepts 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 `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 position of the robot, and targetpos is the current 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 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.**

6. [15 pts] Write a project report describing your approach to the two programming assignments (Problems 4 and 5). Your report should include the following sections:
   - **Introduction**: present a brief overview of your approach
   - **Problem Formulation**: state the problem you are trying to solve in mathematical terms. This section should be short and clear and should define the quantities you are interested in. In particular, you should include the basic elements of the shortest path problems you are trying to solve.
   - **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. Discuss the computational complexity of your algorithms and comment on their completeness and optimality.
   - **Results**: present any interesting details in your implementation and discuss your results (e.g., the two tables of costs and number of iterations for Problem 4, similar statistics for the performance of your algorithm on Problem 5, visualizations of the computed paths, etc.) in detail – what worked as expected and what did not and why.