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' solutions or code. ***If you discuss your homework with someone, please indicate their name(s) in your submission****. 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 files on **Gradescope** by the deadline shown at the top right corner.

1. Upload your solutions to problems in pdf format: **FirstnameLastname_HW2.pdf**. 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. Upload all **python** code you have written for each problem in zip format: **FirstnameLastname_HW2.zip**. The zip file should also include a README file with a clear description of the files used for each problem and how they can be run.

Problems

1. [30 pts] In class, we showed that the deterministic shortest path (DSP) problem and the deterministic finite state (DFS) optimal control problem are equivalent. We used this equivalence to apply ideas from dynamic programming to the shortest path problem. In this problem, we will consider the reverse direction – applying shortest path algorithms to the DFS optimal control problem.
   (a) Derive a formulation of the best-first search (Dijkstra's) algorithm that applies to the DFS optimal control problem. Obtain a backward version of the algorithm that starts from the terminal time. Discuss how your formulation compares to the dynamic programming algorithm, e.g., does it investigate fewer states?
   (b) Is it possible to define an A^* algorithm for the DFS optimal control problem? What would a heuristic mean in this case? If it is possible, derive a backward version of the A^* algorithm; if not, explain what prevents the algorithm from being used.

2. [36 pts] In this problem, we will investigate the properties of heuristic functions. As a reminder, a heuristic function is **admissible** if $h(\mathbf{x}_i) \leq \mathbf{dist}(\mathbf{x}_i, \mathbf{x}_\tau)$ for every node i with coordinates $\mathbf{x}_i$, where $\mathbf{dist}(\mathbf{x}_i, \mathbf{x}_\tau)$ is the shortest distance from $\mathbf{x}_i$ to the goal $\mathbf{x}_\tau$. A heuristic function h is **consistent** if:
   - $h(\mathbf{x}_\tau) = 0$ for the goal node τ with coordinates $\mathbf{x}_\tau$,
   - $h(\mathbf{x}_i) \leq c(\mathbf{x}_i, \mathbf{x}_j) + h(\mathbf{x}_j)$ for every node i with coordinates $\mathbf{x}_i$ and its children j with coordinates $\mathbf{x}_j$.

   Finally, a heuristic h is **ϵ-consistent** with $\epsilon \geq 1$ if:
   - $h(\mathbf{x}_\tau) = 0$ for the goal node τ with coordinates $\mathbf{x}_\tau$,
   - $h(\mathbf{x}_i) \leq \epsilon c(\mathbf{x}_i, \mathbf{x}_j) + h(\mathbf{x}_j)$ for every node i with coordinates $\mathbf{x}_i$ and its children j with coordinates $\mathbf{x}_j$.

   (a) Prove that if $h^{(1)}$ and $h^{(2)}$ are consistent heuristics, then $h := \max\{h^{(1)}, h^{(2)}\}$ is also consistent.
   (b) Prove that if $h^{(1)}$ and $h^{(2)}$ are consistent heuristics, then $h := h^{(1)} + h^{(2)}$ is ϵ-consistent.
   (c) Prove that if h is consistent, then it is also admissible.
   (d) Given an example of an admissible heuristic h that is not consistent. Show explicitly why h violates consistency.
   (e) Prove that if the heuristic function used in A^* is consistent, then A^* will not re-open nodes. In other words, show that it is not possible to improve the label g_j of any node j that has already been expanded and placed in the CLOSED list.

(f) Consider the stage cost $c(\mathbf{x}_i, \mathbf{x}_j) := \|\mathbf{x}_i - \mathbf{x}_j\|_2$ and the heuristic function $h(\mathbf{x}_i) := \|\mathbf{x}_i - \mathbf{x}_\tau\|_\infty + 0.4\|\mathbf{x}_i - \mathbf{x}_\tau\|_{-\infty}$, where for a vector $\mathbf{x} \in \mathbb{R}^D$ with elements $\mathbf{x}[d]$, the functions $\|\cdot\|_\infty$ and $\|\cdot\|_{-\infty}$ are defined as:

$$\|\mathbf{x}\|_\infty := \max_d |\mathbf{x}[d]| \qquad \|\mathbf{x}\|_{-\infty} := \min_d |\mathbf{x}[d]|$$

Is h consistent? Is h admissible? Provide a proof or counter-example.

3. [42 pts] Suppose that the charge of your cell phone's battery takes on values $[n] := \{1, \dots, n\}$ from better to worse. Your browsing and chatting experience has a decreasing reward $r(i)$ in $i \in [n]$. Also, the worse your battery state is, the more likely it is to get worse. Formally, $P(i, j)$ describes the probability of transitioning from state i to state j and the quantity $\sum_{j=l}^{n} P(i, j)$ is increasing in i for all $l \in [n]$ (stochastic dominance). You have the option to recharge your cell phone, which resets its state back to 1 with electricity costs of c. Unfortunately, your cell phone company is not great at making chargers and your charger only works with probability $q \in (0, 1)$. If it happens that the charger does not work, the battery state remains unchanged and the electricity cost is 0.

   (a) Formulate the problem as minimizing an infinite-horizon γ-discounted cost. Write the Bellman equation.

   (b) Prove that the optimal value function $V^*(i)$ is increasing in i.

   (c) Prove that optimal policies are of threshold nature: continue if $i \leq t$ and recharge if $i > t$ for some threshold t.

4. [142 pts] In this problem, we will implement and compare the performance of the A^* and RRT planning algorithms under strict timing constraints. You are provided with a set of 3-D environments described by a rectangular outer boundary and a set of rectangular obstacle blocks (see Fig. 1). Each rectangle is described by a 9 dimensional vector, specifying its lower left corner $(x_{min}, y_{min}, z_{min})$, its upper right corner $(x_{max}, y_{max}, z_{max})$, and its RGB color. The start and goal coordinates are also specified for each of the available environments. Given the current robot position $x_t \in \mathbb{R}^3$, your objective is to implement a planner that produces the next robot move $x_{t+1} \in \mathbb{R}^3$ such that:

   - the robot **moves a small distance** at each time step, i.e., $\|x_{t+1} - x_t\| \leq 1$,
   - the robot remains **collision-free** at x_{t+1},
   - the next position x_{t+1} is produced within **2 seconds**,
   - the robot eventually **reaches the goal**.

   If your robot crashes, then the navigation task terminates unsuccessfully. If your planner takes more than 2 seconds to produce the next move, the robot will move randomly and might collide with the obstacles in the environment. Currently, you are provided with an implementation of a greedy planner that always moves the robot in the direction that decreases the distance between the robot and the goal. The greedy planner satisfies the first three requirements above but it is not guaranteed to reach the goal on every map. You are allowed to change both the `main.py` and the `RobotPlanner.py` files but you should not alter the overall logic of the assignment. For this problem, you should implement **both** a variant of search-based planning (e.g., based on A^*) and a variant of sampling-based planning (e.g., based on RRT). Compare the performance of the algorithms:

   - on a single planning episode in terms of quality of computed path, number of considered nodes, the effect of different parameters such as the choice of heuristic function or the choice of sampling and steer functions, etc.,
   - on the overall task in terms of their ability to reach the goal and the number of moves required.

   Write a project report describing your approach with the following sections:

   - **Introduction**: present a brief overview of the planning problem and 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 and sampling-based planning and the key ideas that enable your planner to scale to large maps and to compute solutions within the allotted time constraints. Discuss the computational complexity of your algorithms and comment on their completeness and optimality.
- **Results**: present any interesting details from your implementation and discuss the performance statistics of the two algorithms in detail – what worked as expected and what did not and why. Consider including images or videos of the performance of your planners on the various environments.

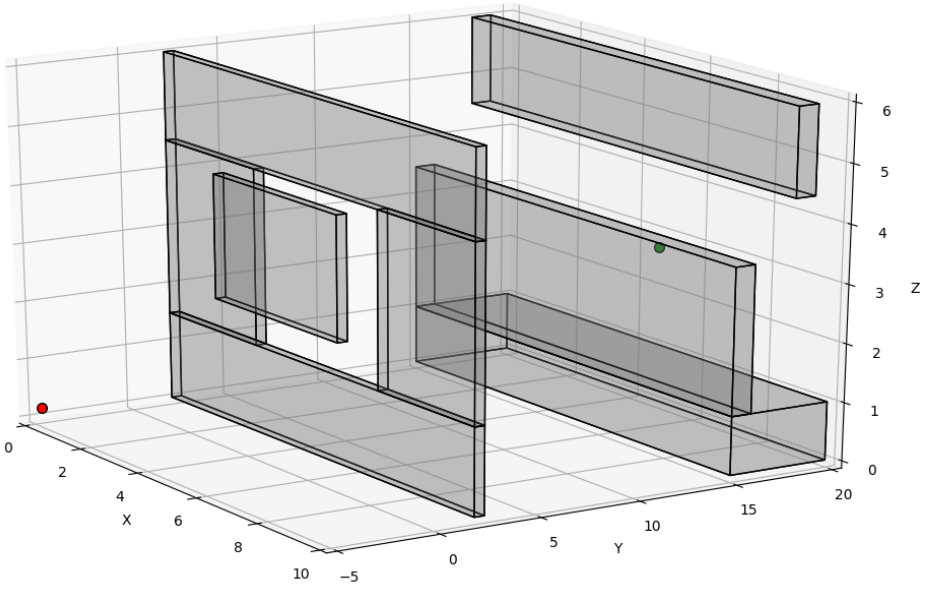

Figure 1: The **Window** planning environment showing the obstacles in gray, the start position in red, and the goal position in green.