ROB 520: Motion Planning
Winter 2022
Homework Assignment #2
Due 2/14/2022 at 2:59pm

Rules:

1. **All homework must be done individually, but you are encouraged to post questions on Piazza.**
2. No late homework will be accepted.
3. **You must show all your work to receive credit for your solutions.**
4. The goal of this homework is to practice some motion planning fundamentals as well as implementations of discrete planning methods.
5. Submit your python code along with a pdf of your answers in a zip file to Canvas.

Questions

1. (15 points) Chapter 3 #3
2. (15 points) Chapter 3 #7
3. (15 points) Chapter 3 #8
4. (15 points) Chapter 5 #5
5. (15 points) Chapter 5 #15
6. (15 points) Chapter 5 #17
7. (15 points) Chapter 5 #18

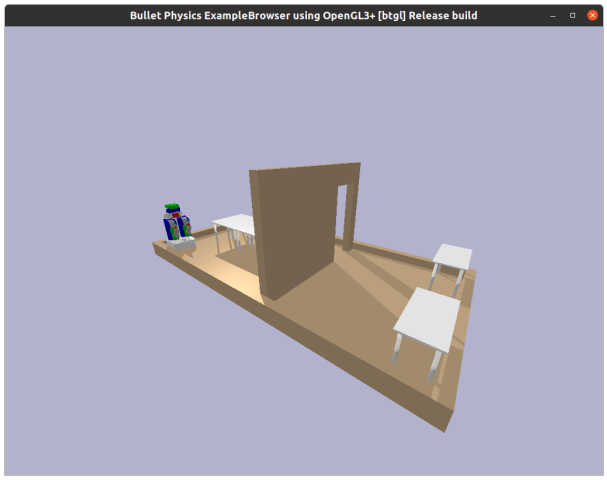
Software

1. Download and unzip `pybullet_interface.zip` (this is the same code as was used for HW1). Run `demo.py` and verify that you see the PR2 robot in a scene with a table.
2. Download the HW2 template here and unzip the files to the `pybullet_interface` directory.

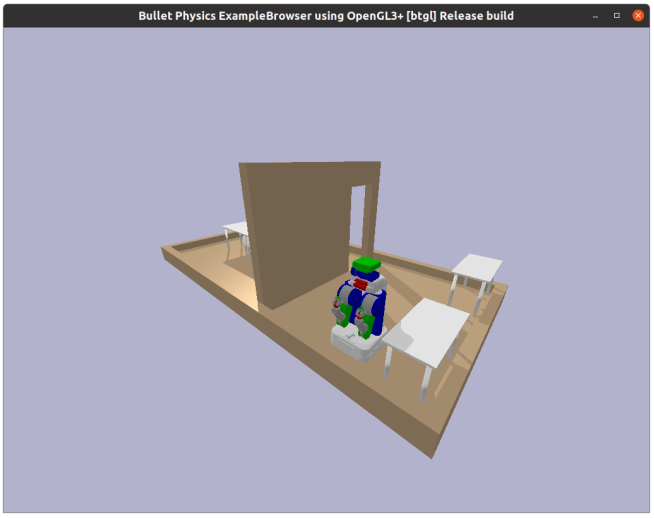
Implementation

The following implementation problems should be done in python starting from the provided template. Only edit what is inside the `### YOUR CODE HERE ###` block. Feel free to look around the internet for example code for reference, but you should implement your own code from scratch unless explicitly stated otherwise.

1. (50 points) Starting from the provided template, implement the A* algorithm to find the shortest collision-free path for the PR2's base from the start to the goal in the given environment (see figure below). Only edit what is inside the `### YOUR CODE/IMPORTS GO HERE ###` blocks.



(a) Start configuration



(b) Goal configuration

You will be planning for translation of the base in X and Y as well as rotation θ about the Z axis of the robot, for 3 DOF total. If no path exists, the algorithm should print out `No Solution Found`. You will need to use a priority queue in the A* algorithm. See the example in `priorityqueue_example.py` to see how to do this in python.

Hint: Remember to consider the topology of the θ DOF when expanding new nodes and computing both the g and h functions.

You do not need to collision-check edges but you must collision-check nodes. You will need to determine reasonable discretizations for each DOF (too small: the algorithm will be very slow, too large: the robot will go through obstacles).

Implement four variants of the A* algorithm:

a. "4-connected" space, with the Manhattan distance heuristic
b. "4-connected" space, with the Euclidean distance heuristic
c. "8-connected" space, with the Manhattan distance heuristic
d. "8-connected" space, with the Euclidean distance heuristic

For each variant record the the computation time to find a path and the path cost in your `pdf`.

For each variant, also save a screenshot including:

- The computed path drawn in the viewer in black (of course you will not be able to draw the rotation along the path, so just draw the X and Y components of the configurations in the path).
- The X and Y components of the collision-free configurations explored by A* in blue.
- The X and Y components of the colliding configurations explored by A* in red.

Make sure to draw the points slightly above the floor so that you can see them. Include these screenshots in your `pdf`.

Note that there are multiple configurations with the same X and Y locations (because θ varies), don't worry about which one is shown on top (and thus visible). We will be looking at the points to see a general pattern of how A* explores, we don't care so much about an individual point being a specific color.

To grade your work, we will run the command `python3 astar_template.py` in a folder where we have extracted your source code. We will not run any other command or any modifications of this command. When this command is run, your code should plan using variant (b) and execute a trajectory for the robot and we should see the robot following this trajectory in the viewer. Your code should output the solution in **under 4 minutes**.

Include answers to the following in your `pdf`:

(a) (10 points) Which heuristic seems superior for 4-connected? Explain your answer.

(b) (10 points) Which heuristic seems superior for 8-connected? Explain your answer.

(c) (10 points) For the four variants above, which, if any, variants have an admissible heuristic and which, if any, do not? Explain your answers.

2. (40 points) Make a copy of the unedited template and call it `anastar.py`. Implement the ANA* algorithm described in this paper.

   You will be able to re-use much of the code you wrote for A*. Use "4-connected" space, with the Euclidean distance heuristic. Use the same discretization as you used for Problem 1. Set a large time-out for the algorithm so that you are sure it will converge to the optimal path.

   Save a screenshot including the same information and using the same colors as those in Problem 1. Include this screenshot in your `pdf`.

   To grade your work, we will run the command `python3 anastar.py` in a folder where we've extracted your source code. We will not run any other command or any modifications of this command. When this command is run, your code should plan and execute a trajectory for the robot and we should see the robot following this trajectory in the viewer.

   (a) (15 points) Generate a plot similar to Figure 3(a) in the paper showing the solution path cost vs. time. Include the solution cost and computation time for A* variant (b) in the plot. How long does it take ANA* to converge to a solution with the same cost as the A* solution? Explain why this computation time makes sense. Include the graph and explanation in your `pdf`.

   (b) (15 points) Move some of the tables in the environment to make the problem more difficult to solve. It should take ANA* longer to find the best solution path than before. Include a screenshot of the environment where you have drawn each solution path using a different

color in your `pdf`. Clearly describe which color corresponds to which solution path. Explain why this problem was more difficult for ANA* to solve; i.e. what feature(s) of the environment made it more difficult? Include these explanations in your `pdf`.

3. (Extra Credit: 35 points) Make another copy of the template called `footsteps.py` and use it to implement the footstep planner described in this paper. Remove the PR2 robot from the environment. Replace the tables with solid boxes that are roughly the same size. You do not need to load in a humanoid robot to actually execute the footstep sequence, for this problem only planning the sequence is enough.

   - Assume each foot is a rectangle of size 0.3m by 0.15m.
   - Assume the start pose of the PR2 in the template is the start pose for the footsteps; i.e. the X and Y position specifies the location of the point between the feet. The feet should be pointing toward the wall with the doorway.
   - Assume there is a small goal region around the goal pose (around 0.05m in X and Y and 0.1rad in θ). Placing the point between the feet anywhere in this region counts as completing the task.
   - You will need to choose a reasonable transition model of possible footsteps for a humanoid. This model is basically a list of locations where the robot can put one foot while keeping the other foot static.
   - For collision-checking, only consider if a footstep will collide with an object in the environment (don't worry about the whole robot).
   - Assume a transition between two footsteps is collision-free if both footsteps are collision-free.

   Produce a screenshot similar to Figure 6 in the paper showing the left (red) and right (blue) footsteps from start to goal and include it in your `pdf`. Describe the transition model you used in your `pdf`.

   To grade your work, we will run the command `python3 footsteps.py` in a folder where we've extracted your source code. We will not run any other command or any modifications of this command. When this command is run, your code should plan a sequence of footsteps and produce a visualization of the sequence as described for the screenshot above.