# EECS 465/ROB 422: Introduction to Algorithmic Robotics
# Fall 2023
# Homework Assignment #3
# Due 10/25/2021 at 11:59pm

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. The goal of this homework is to develop your understanding of motion planning methods. You should use python to implement solutions. You may not use any other language, only python will be accepted.
4. Submit your python files along with a `pdf` of your answers to Gradescope. Do not paste your code into your `pdf`.
5. Remember that copying-and-pasting code from other sources is not allowed.

## Questions

1. (15 points) AI book, exercise 3.3
2. (10 points) AI book, exercise 4.1
3. (10 points) LaValle book, Chapter 4 exercise 5
4. (10 points) LaValle book, Chapter 4 exercise 16
5. (10 points) LaValle book, Chapter 5 exercise 18

## Software

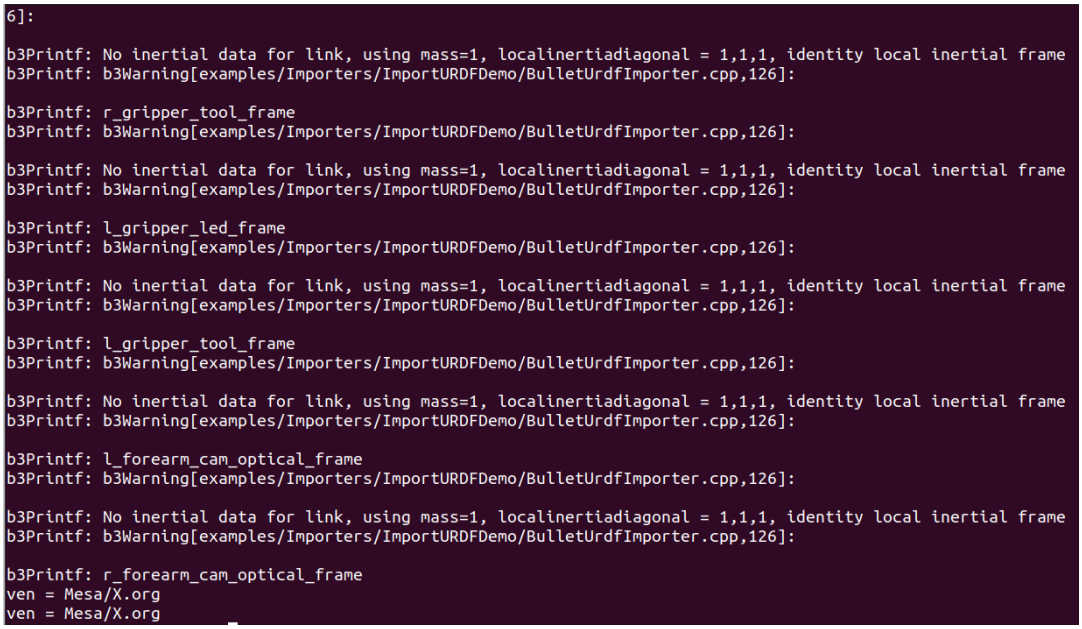
1. Starting from this homework, we will be writing code with PyBullet. Our code runs in Ubuntu 20.04 and we cannot guarantee that this code works with a different OS. Because of this, you will need to set up VirtualBox if you don't already have access to an Ubuntu 20.04 computer. Altenatively you can dual-boot into Ubuntu 20.04 but this is only recommended for people who are comfortable modifying partitions on their hard-drive. If you (unwisely) choose to use a different set-up, e.g. using a different version of Ubuntu, using MacOS, or using WSL, we will not be able to help you or answer your questions if you have problems.

   VirtualBox lets you run an Ubuntu virtual machine on your computer without installing Ubuntu directly. You will need to download and install VirtualBox as well as download the Ubuntu 20.04 ISO (download the ubuntu-20.04.6-desktop-amd64.iso file). The ISO file contains the Ubuntu operating system.

   From there, you can follow this tutorial to finish the setup.

2. Install `pybullet` by doing `pip3 install pybullet`

3. Download and unzip `HW3files.zip`. Run `demo.py` and verify that you see the PR2 robot in a scene with a table. You will be asked to press enter multiple times to demo some `pybullet` capabilities. When you run `pybullet` with the PR2 robot, you will see warnings in the terminal that look like:




   These are issues with the definitions of the robot links/frames and will not affect your code in any way, so please ignore them.

## Quick Information about PyBullet

`pybullet` is a physics engine that is used for simulating both soft and rigid bodies, including robots. We will NOT be using pybullet for physics simulation, but only for visualization and collision checking. To do this we've provided you with several helper functions in `utils.py` so that you can implement your motion planners and visualize the results. These helper functions should make it so that you don't have to worry about how `pybullet` works. You should never need to use an actual `pybullet` function to complete the assignment, the only interface should be through the helper functions we provide for you. To show you how to use these functions we've created a demo script called `demo.py`. You can refer to this script when writing your code to see how to use the helper functions.
Below are some tips for using `pybullet` that can help with your coding/debug process:

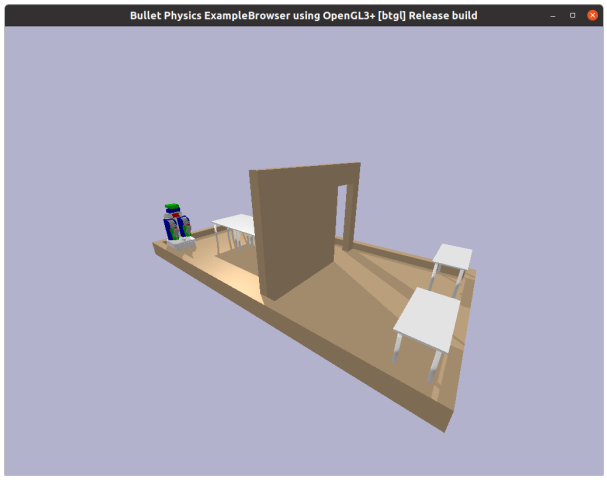
- **Joint Groups**: A robot can have many degrees of freedom (e.g. a humanoid), but you may not want to plan for all of those DOFs at the same time. For instance, you may want to drive the mobile base of a robot but not move the robot's arms. You can see in the templates where the names of the relevant joints for the given problem are selected. Note that we sometimes refer to the "base joints" of the robot to denote the position and rotation of the base (a more accurate term would be "base DOF").

- **Viewer**: The `pybullet` viewer comes with some handy functionality to allow you to zoom in and out and focus on different parts of the scene:
  - Scroll Wheel or (CTRL (or ALT) + Right Mouse drag): Zoom camera in and out
  - CTRL (or ALT) + Left Mouse drag: Rotate camera
  - CTRL + Middle Mouse drag: Translate camera
- **Virtual Machines**: If you are running `pybullet` on a virtual machine (VM), make sure to allow the VM to use at least 2 cores and as much video memory as possible. This will make the rendering faster and will allow you to control the GUI more fluidly).

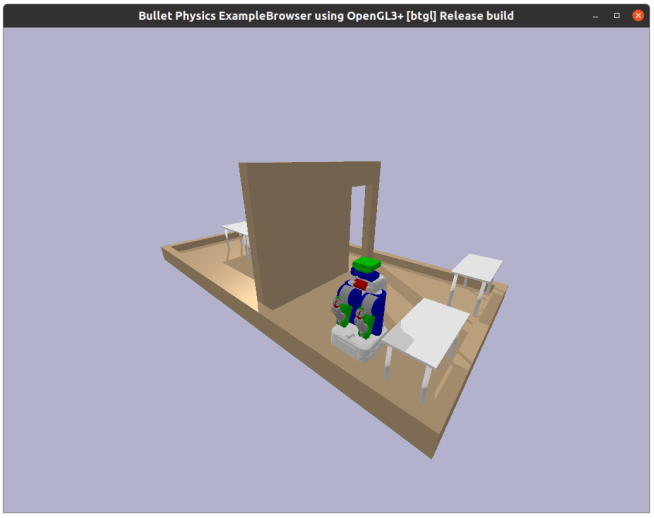
## Implementation

The following implementation problems should be done in python starting from the provided templates. Only edit what is inside the `### YOUR CODE HERE ###` block.

1. Starting from the `astar_template.py` 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 $\theta$ 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.

The action cost of moving from node $n$ to node $m$ should be:

$$c(n,m) = \sqrt{(n_x - m_x)^2 + (n_y - m_y)^2 + min(|n_\theta - m_\theta|, 2\pi - |n_\theta - m_\theta|)^2}.$$

The heuristic for node $n$ should likewise be:

$$h(n) = \sqrt{(n_x - g_x)^2 + (n_y - g_y)^2 + min(|n_\theta - g_\theta|, 2\pi - |n_\theta - g_\theta|)^2},$$

where $g$ is the goal configuration.

*Hint*: Remember to consider the topology of the $\theta$ DOF when expanding new nodes.

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 two variants of the A* algorithm:

a. "4-connected" neighbors

b. "8-connected" neighbors

For each variant record the the computation time to find a path and the path cost (using the action cost function defined above) 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`.

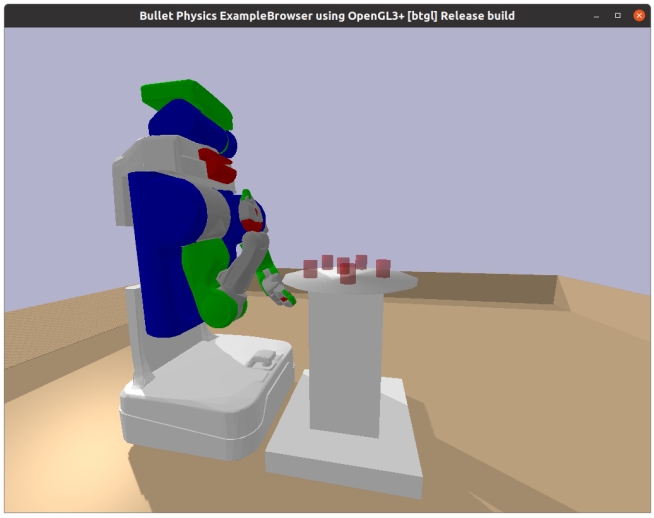
Note that there are multiple configurations with the same X and Y locations (because $\theta$ varies), don't worry about which one is shown on top (and thus visible). We will be looking at the points to see a general pattern of how A* explores, we don't care so much about an individual point being a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

(50 points) 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 10 minutes.

(10 points) Which variant performs better in terms of computation time? Which variant performs better in terms of path cost? Explain why in the `pdf`.

2. (40 points) Implement the RRT-Connect algorithm to search for collision-free paths in configuration space from the current configuration of the robot to a goal configuration. To speed up your code, you need only check collision between the robot and the environment; checking self-collision is not necessary. Put your code in `rrt_template.py` and do not modify code outside the indicated blocks for your code. Use a step size of 0.05rad and a goal bias of 10%. You will be planning for 6 DOF of the PR2 robot's left arm (the 7th DOF, which is the wrist roll joint, is not used).

   Once you have computed the path from start to goal




(a) Start configuration


(b) Goal configuration

a. Draw the position of the left end-effector of the robot for every configuration along the path in red in the viewer (see `demo.py` for how to get this position). You should see that the points along the path are no more than a few centimeters apart. Include a screenshot showing the path you computed in your `pdf`.

b. Execute your path and verify that the arm indeed reaches the goal (doesn't hit obstacles).

(20 points) Implement the shortcut smoothing algorithm to shorten the path. Use 150 iterations. Draw the original path of the end-effector computed by the RRT in red and the shortcut-smoothed path of the end-effector in blue in the viewer. Include a screenshot showing the two paths in your `pdf`.

To grade your work, we will run the command `python3 rrt_template.py` in a folder where we have extracted your source code. When we run `python3 rrt_template.py`, we should see the execution of the smoothed path in the viewer. We will not run any other command or any modifications of this command. We will run the code three times and your code should output the solution in under 10 minutes at least once.