3630 Intro to Robotics and Perception
Release Date: April 12, 2024
Due Date: April 23, 2024

Project 6: Warehouse Automation

Overview

In this lab, your robot will be tasked with supervising a warehouse. The robot will have to map out its environment (using a lot of the same algorithms you have employed throughout the class), while avoiding obstacles (walls). If an obstacle is hit, the run is terminated. Your goal is to visit all of the five landmarks within 6 minutes (360 seconds). All the important files are present under the folder *controllers/exploration_controller/*. Specifically, focus on *controllers/exploration_controller/exploration.py* file.

Exploration task

Map Layout

The map is organized as an occupancy grid, where each cell in the grid is either labeled “free”, “obstacle”, or “landmark”. In addition, every cell in the map starts out as unexplored (grey area), and as grid cells enter the robot’s field of vision, the cells are detected by the robot’s sensors and should be marked as explored.

Localization

Markers can be examined by sensors, and a reading of their position and heading is acquired for these markers once they’re in the robot’s field of view. All landmarks must be visited in order for the exploration phase to be awarded full credit.

Planning

You will implement the *frontier_planning()* as well as the *exploration_state_machine()* functions in *controllers/exploration_controller/exploration.py* in order to traverse the map and visit all of the landmarks. Your frontiers are the unexplored cells in the map that are adjacent to the explored cells, and your algorithm should move to the centroid of the frontiers.

As Project 6 will be released prior to the final due date of Project 5, and in Project 5 you are implementing RRT, you will copy your implementation of RRT to use when navigating to your computed frontiers (*grid.py*). However, you will not be submitting this file – it will be only to test locally. To ensure that all students can still get started on Project 6, we provide the RRT implementation on Gradescope for grading. Once Project 5 is **closed for everyone, taking late submissions into account**, we will provide you with this RRT implementation for local testing as well.

Local Testing:

We provide 3 maps for your local testing under *controllers/exploration_controller/maps* that can be used for local testing. We also provide `autograder.py` file that you can run for local grading. Example usage of `autograder.py`: "*python3 autograder.py maps/maze1.json*".
You can visualize the code locally, by running *exploration_controller/robot_gui.py* with the desired map specified. While there are 5 markers in each map, this file will list the total number of markers as greater than 5 due to the discretization of the world, as you can observe via the GUI. Note that each map is still worth a total of 10 points.

Make sure to take advantage of functions in `utils.py` and remember that your run for either phase will terminate if you collide with an obstacle.

Grading

Rubric	
Found all five markers in the map	10 pts/map

Autograder will grade your exploration on 10 maps.

Submission Materials
Submit your `exploration.py` file to gradescope.
If you relied significantly on any external resources to complete the lab, please reference these in the submission comments.

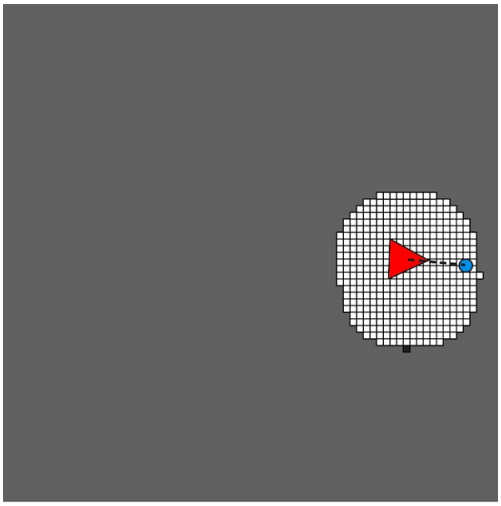
Running with Webots

Once the exploration.py is complete, you can validate that your algorithm works with the WeBots simulator.

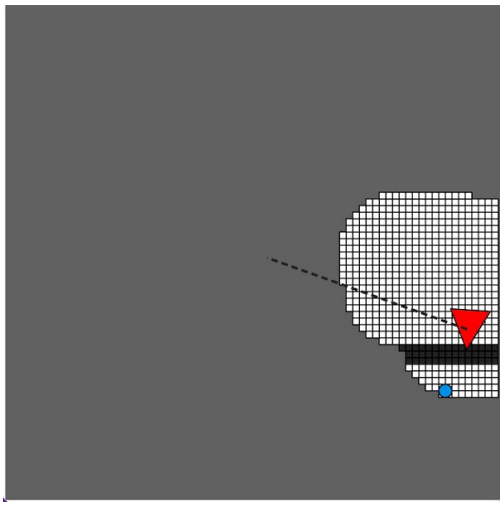
1. Open exploration_controller.py and **ensure that the `maze_name` (line: 255) is set to the map you would like to test on**. The map names are "maze1", "maze2", and "maze3". A working implementation should work on all three maps.
2. Launch WeBots. Click File > Open World, then navigate to the "worlds" subfolder. Open the `.wbt` file which matches the **`maze_name`** from step 1.
3. A GUI window should appear which shows your exploration code running in a simple 2D approximation of the world to find a path. Then, the robot in WeBots should start moving along this path.
    a. We recommend selecting the "DEF e-puck Robot" object via the sidebar in the top-left of the WeBots screen. This will allow you to see the coordinate frame of the robot while it moves through the maze.
    b. Note that the 2D GUI window should automatically close after the robot is done moving.
    c. Also, the webots execution takes significantly longer time to run compared to GUI and autograder version.

Examples (Robot-GUI)

Robots should navigate towards the frontier



Robot can use RRT to get around the obstacle



Blue dot signifies the goal point
Dotted line signifies the path to the next waypoint in the RRT path