

Principles of Robot Autonomy I

Homework 4

Due Friday, November 14th (5:00pm)

Starter code for this homework will all be through Google Colab. Relevant links are listed for each problem.

You will submit your homework to Gradescope. Your submission will consist of a single pdf with your answers for written questions and relevant plots from code.

Your submission must be typeset in \LaTeX .

Problem 1: Object Detection using PyTorch

Objective: Detect objects in a given image using a pre-trained Convolutional Neural Network (CNN) model from PyTorch.

Setup: The code for this problem is provided through this Google Colab notebook:




https://colab.research.google.com/drive/1QPjoZa7-2l8AsdXcBTsPjzQmhLuNp_hf?usp=sharing

You will not have edit access to this notebook. To create a copy in your own Google Drive, select **File > Save a Copy in Drive**. Accept the prompt to open your copy in a new tab. You will have edit access to this copy. If you need to find where in your drive the copy is, select **File > Locate in Drive**.

In this assignment, you are tasked with running a CNN model on an image to determine which objects are in it. Fortunately, the model is pre-trained and most of the setup is done for you as well.

You should follow along in the notebook `HW3_P2_Object_Detection.ipynb`, which has 3 distinct parts: Environment setup, Object detection, and Visualizing bounding boxes. To begin, take a look at the first code block to understand how the environment was setup. In parts 2 and 3, you get to apply PyTorch functions to integrate and run an “off-the-shelf” CNN model and then plot the boxes detected by the model.

When you are done with the notebook, return here and complete the following short answer questions.

- (i)  Include a screenshot of the cars image with bounding boxes with `score_threshold` of 0.6.
- (ii)  What happens to the bounding boxes if you increase `score_threshold` to 1? Why do you think this is the case? Feel free to try running your code on the other provided images to inform your response.
- (iii)  What happens to the bounding boxes if you decrease `score_threshold` to 0.01? Why do you think this is the case? Feel free to try running your code on the other provided images to inform your response.

Problem 2: Frontier Exploration

Objective: Implement a frontier exploration algorithm to decide where to travel in a map.

Setup: For this problem, we will use a Google Colab notebook linked here:

<https://drive.google.com/file/d/10-100KMwEDijUwd42H70yHF4YJe8NIio/view?usp=sharing>.

Please make a copy of this notebook to your drive to make changes.

Map Representation: We will use a Stochastic Occupancy grid that we've used in previous sections to keep track of the states that are known vs. unknown, and occupied vs. unoccupied.

For this problem, when a state in the grid is within the sensing radius of the robot, the state is known. When the state is known, it is assigned with a probabilistic value of whether there is an object occupying that grid cell. These probabilities range between 0 and 1, and for the purposes of this question, we will use the heuristic that a cell is considered occupied if the probability of occupancy is greater than or equal to 0.5. If a cell is unknown, (has not been sensed), then the value of the cell is -1.

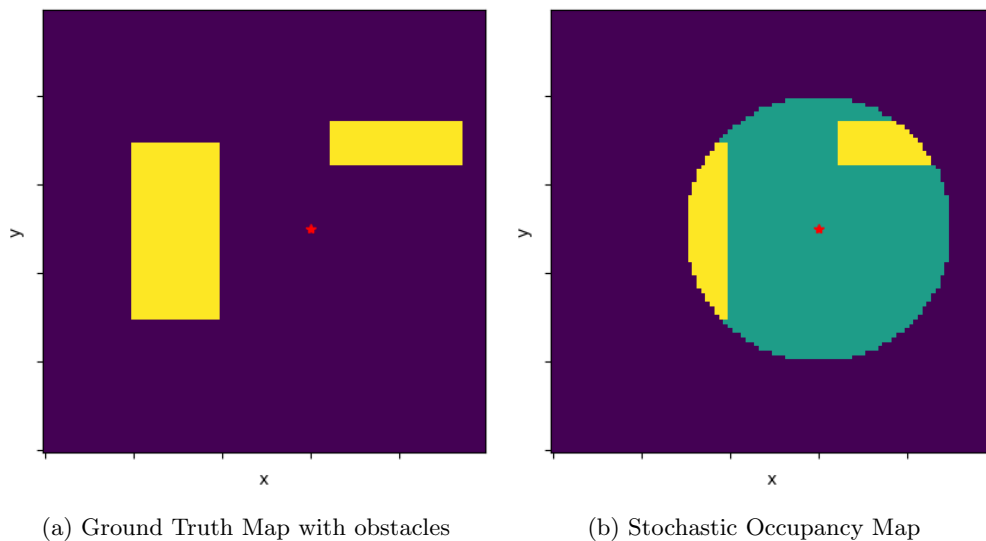


Figure 1: Obstacles are shown in yellow, the current state of the robot is marked with a red star.

Exploration Heuristics: With a given Stochastic Occupancy map, we want to decide what states to explore to balance gathering new information (traveling to unknown states), while remaining in regions where we don't believe there are any obstacles (traveling to unoccupied states). To balance these two things, we will look for states that satisfy the following conditions:

1. The percentage of unknown cells surrounding a cell in some window should be greater than or equal to 20% of the surrounding cells.
2. The number of known, occupied cells surrounding a cell in some window should be 0.
3. The percentage of known, unoccupied cells surrounding a cell in some window should be greater than or equal to 30% of the surrounding cells.

Cells that satisfy these three heuristics will be considered valid cells to explore the frontier. A visualization of an example for defining these windows is shown in Figure 2. Note that the heuristics in the simple example shown in Figure 2 are different from the heuristics listed in the problem statement.

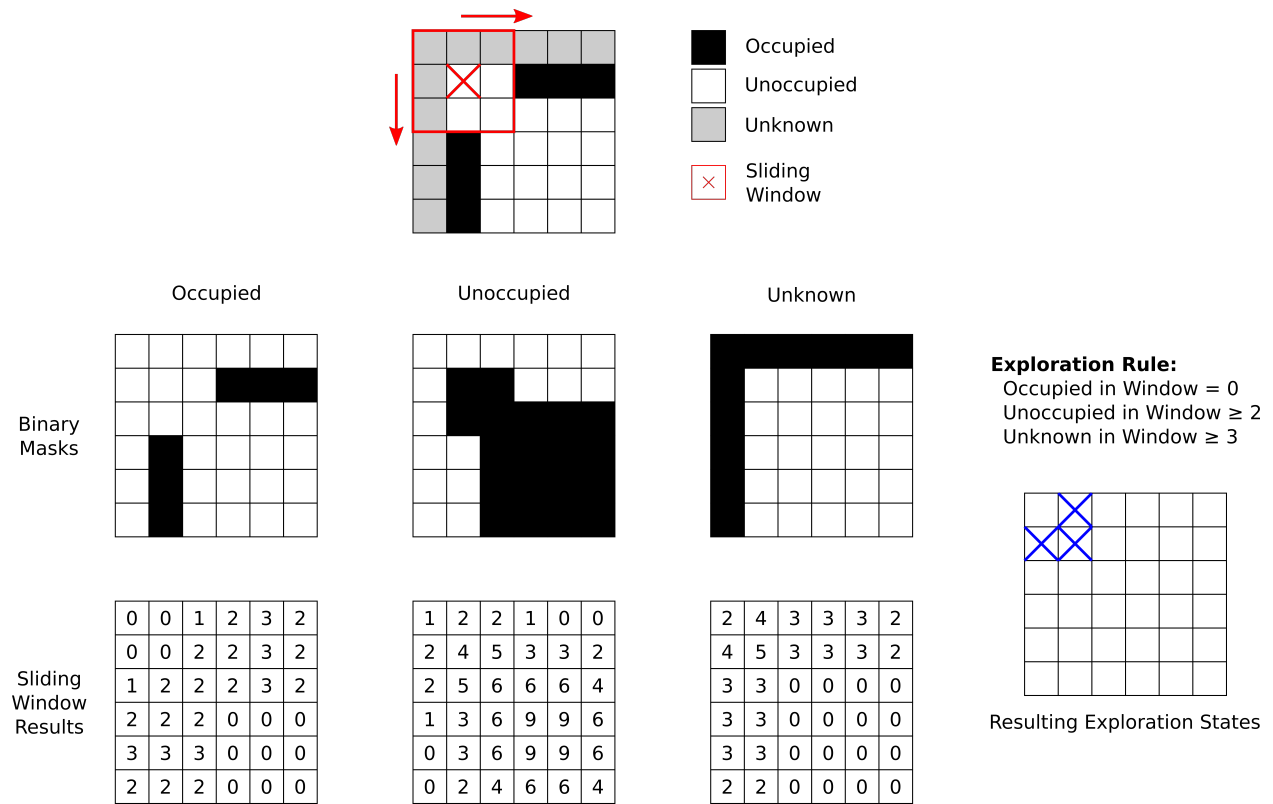


Figure 2: Visualization of using Convolutions for Heuristics for a sample problem

- Write the **explore** function in the notebook according to these heuristics and return the resulting plot, showing the robot's Stochastic Occupancy Map and the cells that satisfy the exploration heuristic.
- We will choose a state from the valid frontier states that is closest to the current state of the robot. What is the euclidean distance of the closest frontier state to the robot's current state? Include this calculation in your **explore** function, and report the distance value in your write-up submission.

[Section Prep]: Frontier Exploration

Note: This portion of the homework is **not graded**, but should be completed before Section on Week 9 to test in hardware. All the URLs are highlighted in [blue](#). Make sure you click on them as they are important references and documentation!

Please also run [update_tb_ws](#) (anywhere in your terminal) somewhat frequently. The provided library code [asl-tb3-utils](#) from the course staff is under rapid development changes.

Objective

In this assignment, you will practice writing ROS2 node and launch files from scratch, while adding autonomous exploration capabilities to the turtlebot stack you have been building throughout the quarter. **The requirements are as follows,**

1. Everything should be integrated into one single launch file, similar to the [navigator.launch.py](#) you wrote from HW2. Note that the launch file **does not** need to launch either the simulator or the hardware bringup on the turtlebot.
2. On launch, your robot should start exploring the map autonomously until the closed environment is mostly mapped out. The exploration does not have to end with a perfectly watertight map but there should not be any large unexplored regions inside the closed environment.
3. The robot should not run into any obstacles any time during the exploration.
4. The launch should also include RViz to visualize the reconstructed map in real time.
5. The autonomous exploration logic needs to be its own standalone ROS2 node.

The solution to this problem is quite open-ended as you have a lot of freedom when designing your own ROS2 node. We have provided the following guidelines to help you build this.

ROS2 Node

1. In your frontier exploration node constructor, you need to create various subscribers and publishers to interface with the navigator. Take a closer read at the [base navigator class](#) to see what topics are being subscribed and published.
2. Adding to the previous point, the navigator publishes a boolean message to the [/nav_success](#) topic. A true message is published when the robot reaches the commanded navigation pose, and a false message is published when planning (or re-planning) fails. Your exploration node needs to listen to this topic to figure out when to send out the next navigation command.
3. Code from Problem 2 can be ported into this ROS2 node with minimal edits. Make sure your code works well in the notebook from Problem 2 before testing this in the ROS2 stack as it will be harder to debug with everything running together.
4. You might want to subscribe to [/state](#) and [/map](#) topics to receive the latest robot pose and map information for calculating eligible frontier locations.
5. [StochOccupancyGrid2D](#) can be useful for processing map data. The base navigator class gives an example of how to [import this class](#) and [construct an occupancy object](#) given a map message.

ROS2 Launch File

1. The launch file should be largely similarly to `navigator.launch.py` from HW2.
2. Add your frontier exploration node into your launch file.
3. It may or may not be helpful to delay launching your exploration node with `launch.actions.TimerAction`. An example on how to use it can be found [here](#).

Testing in Simulation

You can test your implementations in several simulated maps. Up to this point, you should be familiar with launching the simulator. Here are some suggested sim environments to test with:

1. `ros2 launch asl_tb3_sim root.launch.py`
2. `ros2 launch asl_tb3_sim maze.launch.py`