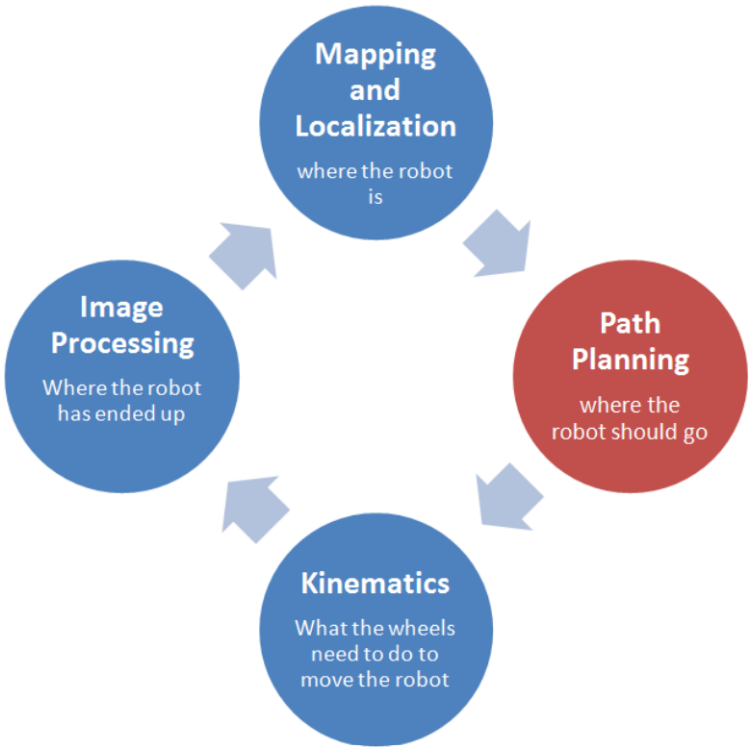
EXPERIMENT 5: PATH PLANNING (Updated Version)

Topics Covered

- SLAM (simultaneous localization and mapping) – simultaneous constructing a map from scratch and localizing robot's pose in the map.
- Path Planning – Contructing optimal path from robot's initial position to a desired set point.
- Path-tracking controller – Applying a controller to track sub waypoints of the constructed path.

Prerequisites

- The QBot 2e has been setup and tested. See the QBot 2e Quick Start Guide for details.
- You have access to the QBot 2e User Manual.
- You are familiar with the basics of MATLAB and SIMULINK.



1 Background

The Quanser QBot 2e Mobile Platform comes with a Microsoft Kinect sensor that has the ability to generate a depth map of the environment. This information, along with the location and orientation of the robot chassis, can be used for autonomous map building. The 2D map generated using this data will be processed to locate various obstacles in the map and create the "occupancy grid map" of the environment. In this lab we only focus on the path planning portion of this process.

Generally, global path planning is performed in robot's *configuration space*, where the robot is considered as a point object and the obstacle dimensions are increased by the maximum centroidtoperiphery dimension of the robot (in the case of the QBot 2, by the 18 cm radius of the body). Path planning in the

configuration space provides a safety margin between the way points and the obstacle grids. An example of an occupancy grid map is shown in Figure 1.1 where the workspace is represented in pixels.

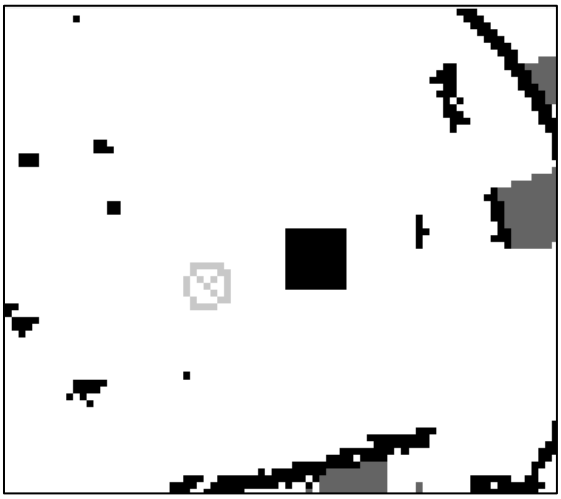
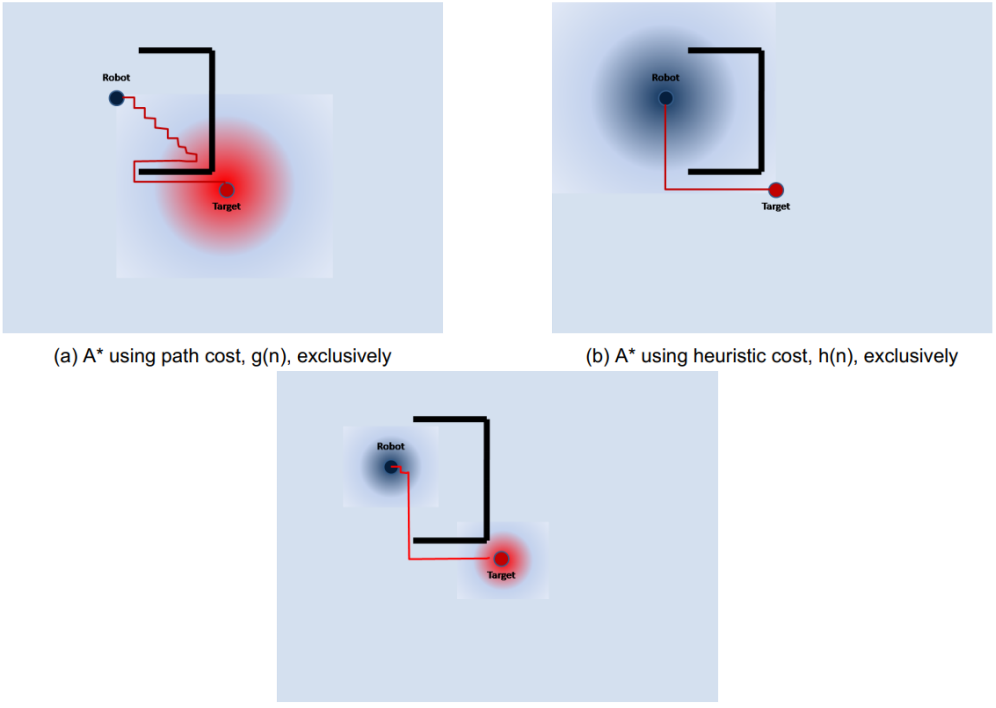


Figure 1: An example of an occupancy grid map.

Global path planning methods search for the connected grid components between the robot and target. There exist several techniques for global path planning; this laboratory will only describe the A* method.

1.1 A* algorithm

A* is the one of the most popular choices for path planning, because it's fairly simple and flexible. This algorithm considers the map as a twodimensional grid with each tile being a node or a vertex. A* is based on graph search methods and works by visiting vertices (nodes) in a graph starting with the current position of the robot all the way to the goal. The key to the algorithm is identifying the appropriate successor node at each step.



(a) A* using path cost, g(n), exclusively

(b) A* using heuristic cost, h(n), exclusively

(c) A* complete

Figure 1.1: A* algorithm components and complete algorithm

Given the information regarding the goal node, the current node, and the obstacle nodes, we can make an educated guess to find the best next node and add it to the list. A* uses a heuristic algorithm to guide the search while ensuring that it will compute a path with minimum cost. A* calculates the distance (also called the cost) between the current location in the workspace, or the current node, and the target location. It then evaluates all the adjacent nodes that are open (i.e., not an obstacle nor already visited) for the expected distance or the heuristic estimated cost from them to the target, also called the heuristic cost, h(n). It also determines the cost to move from the current node to the next node, called the path cost, g(n). Therefore, the total cost to get to the target node, f(n) = h(n) + g(n), is calculated for each successor node and the node with the smallest cost is chosen as the next point.

Using either the path cost g(n) or heuristic cost h(n) exclusively will result on nonoptimized paths as shown in Figure 1.1. Figure 1.1a and Figure 1.1b show the result of using only g(n) and h(n) respectively. Together, an optimized path from the current node to the target node can be achieved as shown in Figure 1.1c.

2 In-Lab Exercise

This exercise consists of the following four steps:

1. Creating an occupancy grid map around an obstacle using the model `QBot2e_2D_Mapping_Keyboard.mdl`.
2. Processing the created map to detect the obstacle using the MATLAB script `process_map.m`.
3. Running path planning algorithms using the MATLAB script `astar_path_planning_8dir.m`.
4. Performing robot motion using the MATLAB script `path_execution.m` and model `QBot2e_Path_Planning_Motion_Control.m`.

2.1 Setting up the environment and creating an occupancy grid map

You will have to first perform an occupancy grid map generation so that the QBot 2e is aware of its surrounding obstacles. The controller model for the mapping, shown in Figure 2.1, is called `QBot2e_2D_Mapping_Keyboard.m`.

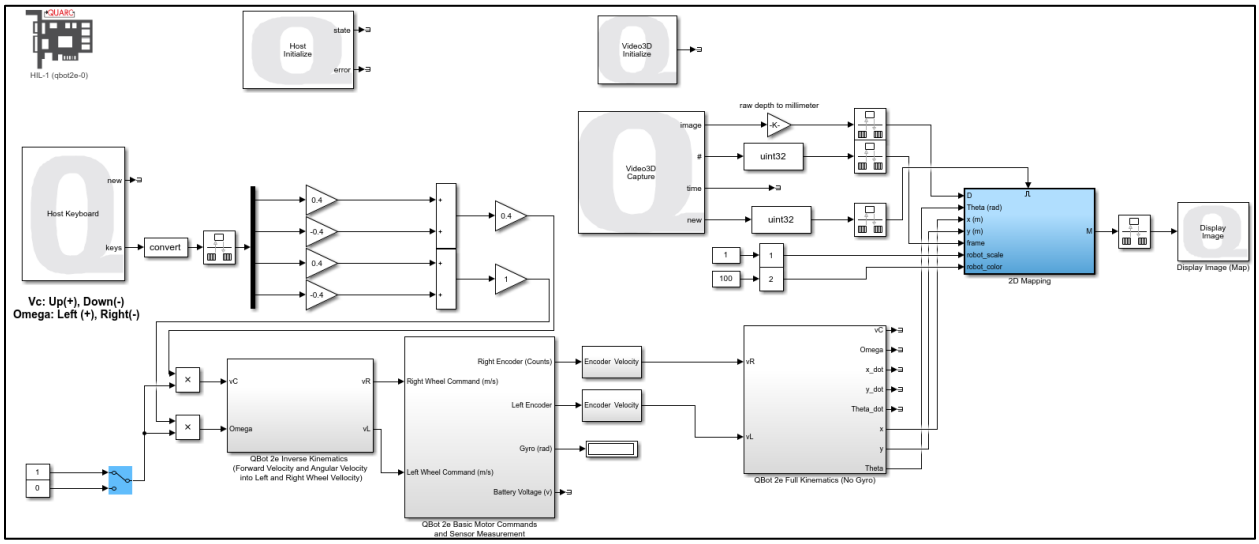


Figure 2.1: Snapshot of the controller model.

Before performing the experiment, make sure the Kinect sensor is flat so that it is parallel with the ground plane and not tilted. The resulting vector is used in the QUARC models to match the depth data to realworld

coordinates. Then open the `QBot2e_2D_Mapping_Keyboard.m` model; make sure you change the IP address to your robot's IP address in QUARCPreferences menu option.

1. Find a zero pose for your QBot 2e where you have at least a 2.5 m×2.5 m free space around it. Mark this initial position and orientation of the the QBot 2e as reference for the next steps. Find a box of at least 0.5 m×0.5 m×0.5 m and place it about 1 m in front of the robot as in Figure 2.2. Put the robot in a known initial configuration (Pose 1), shown in Figure 2.2), run the model and enable the manual switch once the robot is ready. Use the keyboard (up and down for linear motion; left and right for rotation) to move the robot around the obstacle, to poses 24, and then move it back to the initial start point. At each pose try to rotate the robot 360 degrees around itself so it can find all the free space around the obstacles. Note: Make sure the robot always stays at least 1 meter away from the obstacle as in Figure 2.2. Do not drive the robot closer to the obstacle.

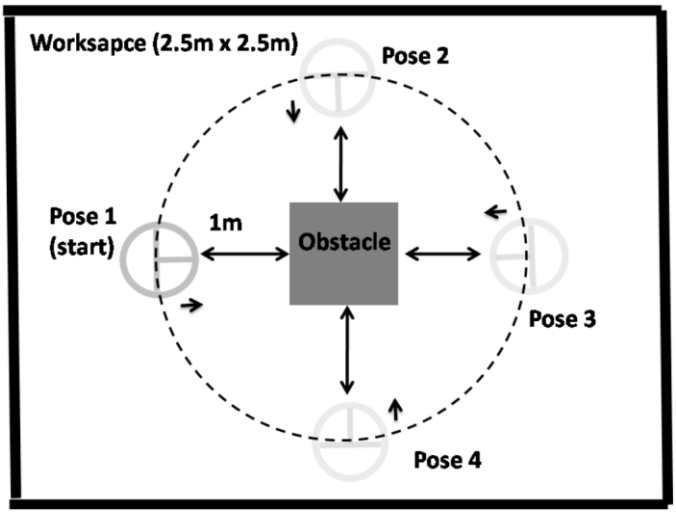


Figure 2.2: Configuration of the robot and obstacle for occupancy grid mapping.

2. When the robot is back to the initial pose, you may see the box in the created map similar to Figure 2.3.

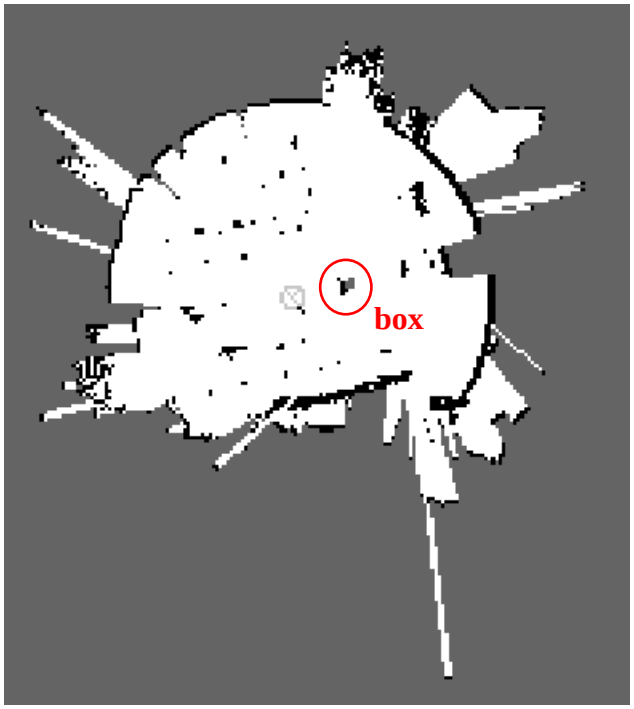


Figure 2.3: Configuration of the robot and obstacle for occupancy grid mapping required for path planning.

3. Right-click on the map figure and select save to workspace. Save it as `map_obs`. Go to the MATLAB workspace, find this item, rightclick on it and save it as `map_obs.mat` in the same folder as the code is for future reference.

2.2 Process map for path planning

Open and run `process_map.m`, various prompt will appear that allow user to customize the recorded map before executing path planning

1. "`Resize the map? Full map takes longer to process algorithms. Type 1 = yes, 0 = no`". It is advised to take option #1 and resize the map since the actual workspace is much smaller than the full map and resized map takes much less time to compute path planning algorithms. User can adjust resize range directly in the script.

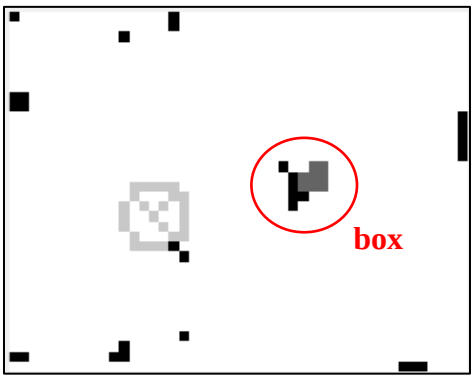


Figure 2.4: Resized map.

2. "`Reinsert the box on map? Type 1 for yes, 0 for no`". It is advised to take option #1 if the box is not clearly shown on the map. Click on supposed position of the box on the map to insert it. User can adjust box size directly in the script.

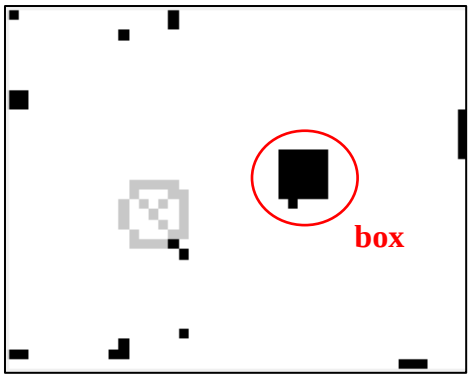


Figure 2.5: Reinsert the box.

3. "`Clear noise? Type 1 for yes, 0 for no.`" It is advised to take option #1 since the camera and hall sensors might have some measurement errors. User can adjust to which extend an occupied pixel is defined as noise (i.e. `if not_free_count < 2`) and cleared automatically, or manually remove the pixel by themselves. Also, the map is automatically padded with free pixels for further operation, user can adjust whether the padding pixels are free or occupied (i.e.: `map = padarray(map,[padding_dist padding_dist],free_val,'both');`).

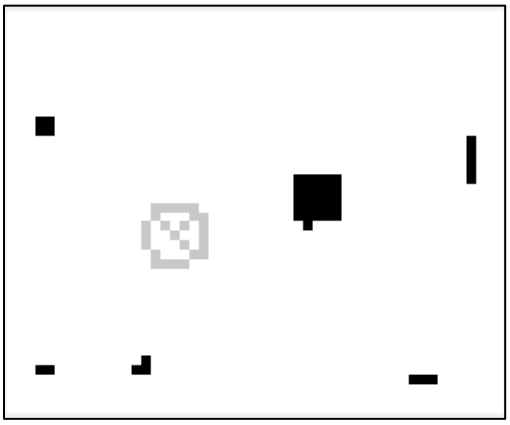


Figure 2.6: Noise filtered and padded image.

4. "`Define safe regions? Type 1 for yes, 0 for no.1`". It is *strongly* advised to take option #1 for safety purpose. This function will amplify the occupied zone, the smallest amplifying distance must be larger than robot's radius (18 cm). User can define the size of amplifying distance (i.e.: `safe_size = 0.2;`)

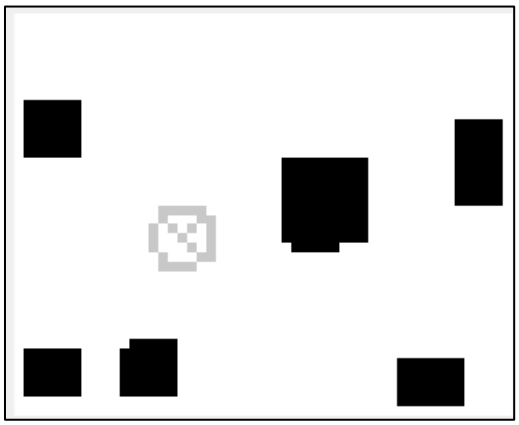


Figure 2.7: safe zone embedded.

5. "`Click to record robot initial position on map`", click on center of the robot.

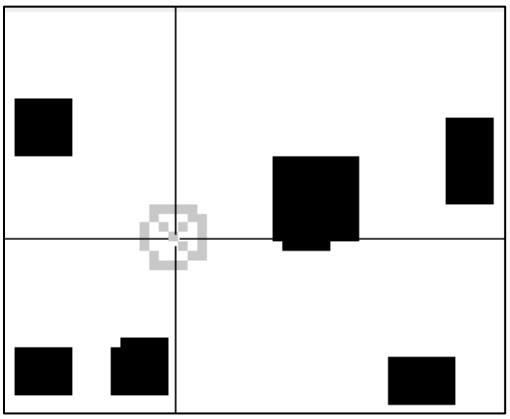


Figure 2.8: Click on initial position of robot.

6. "`Extracted the MATLAB workspace. This file can be closed now`". The information for path planning is recorded as `map_extracted.mat`.

2.3 Path planning

Open and run `astar_path_planning_8dir.m` for A* algorithm (that specify 8 directions to move for the robot). The previous map will be opened and user can click on desired position.

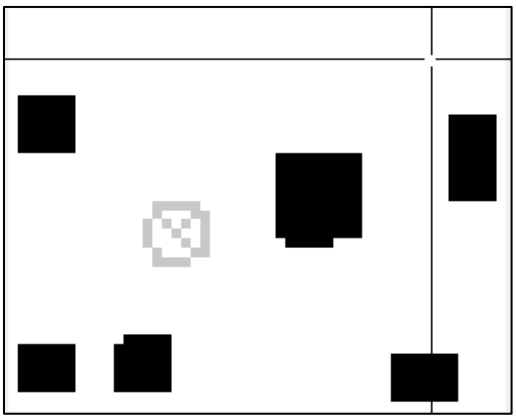


Figure 2.9: Choose destination.

The path is then computed and plotted.

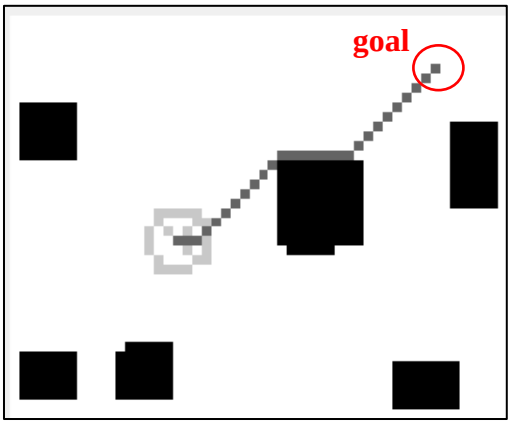


Figure 2.10: Path planned from initial position to goal.

Open and run `path_execution.m` to construct sub goals and store them in workspace.

Open and run `QBot2e_Path_Planning_Motion_Control.mdl` to let the robot moves through sub goals, until the final goal. Enable the switch.

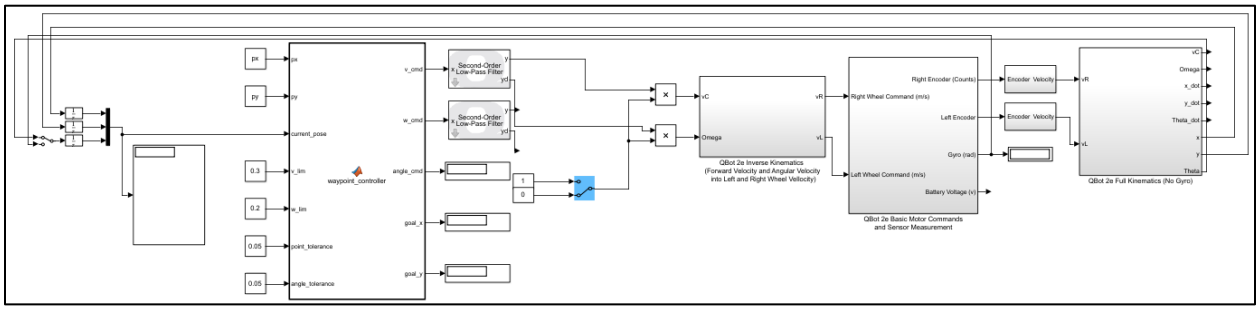


Figure 2.12: Point Tracking Controller.

Question

As you follow the instructions above please document your own data (graphs, figures, etc) at each step and explain thoroughly how the process of mapping using Occupancy Grid Map, path planning using A* algorithm, and point tracking controller work in driving a robot from a start to end point?