Matthew Daniel
mrd89

# Homework 5

MAE 4180 / 5180
2020-April-10

## Mapping - Occupancy Grid with Bump Sensor

1. Please see Figure 1 for the robot trajectory with bump sensing shown in the plot

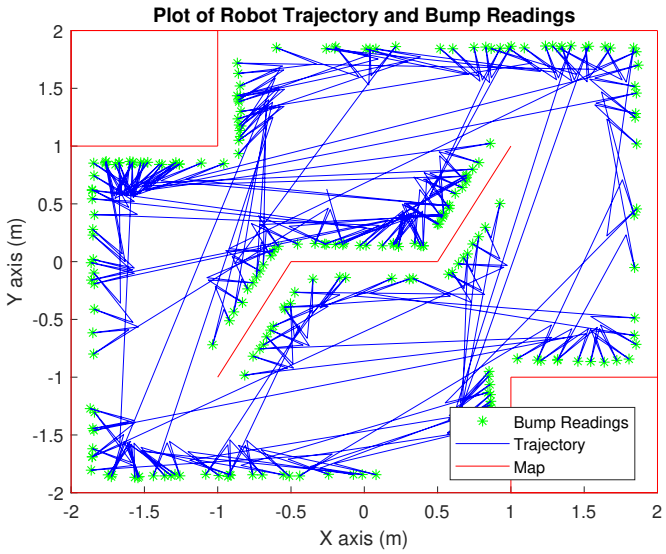



Figure 1: Trajectory of the Robot with Bump sensor Readings and Map

2. The inputs are:

   - **datastore**: struct from running SimulatorGUI
   - $l_0$: initial log odds
   - **NumCellsX**:number of cells in the x direction (integer)
   - **NumCellsY**:number of cells in the y direction (integer)
   - **boundaryX**:boundary of the environment in the x direction. 1 x 2 array [$min_y$ $max_y$]
   - **boundaryY**: boundary of the environment in the Y direction. 1 x 2 array [$min_y$ $max_y$]

   The Output is

   - **Map**:[x,y] matrix of the map with boundaries

3. For this problem, I used the idea that the sensor was extremely accurate while writing this code. For the depth sensing, I added intmax to the probability if there was a wall and if it was a free cell, I used the value intmax /4. For the bump sensing, I used intmax for both the adding and subtracting of probability. Following this, I normalized the map matrix to scale from 0 to 1.

4. Please see my Matlab programs for code! I made time step 1 to be $\frac{1}{3}$ of the robot trajectory, timestep 2 is $\frac{2}{3}$, and time step 3 is the whole trajectory. Figures 3 and 3 look nearly identical due to the robot trajectory changing direction in between the two time steps and so minimal useful data was collected there. There is a very clear distinction between 3 and 4 as the robot finishes it's trajectory.

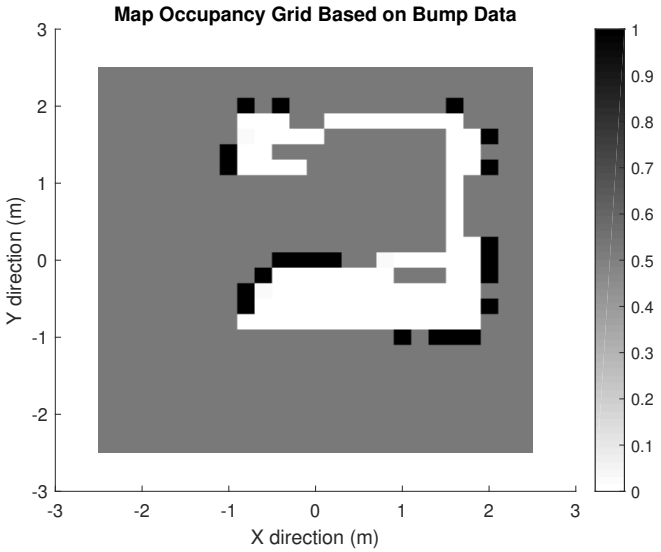



Figure 2: Bump Occupancy Grid at Timestep 1,[25x25]

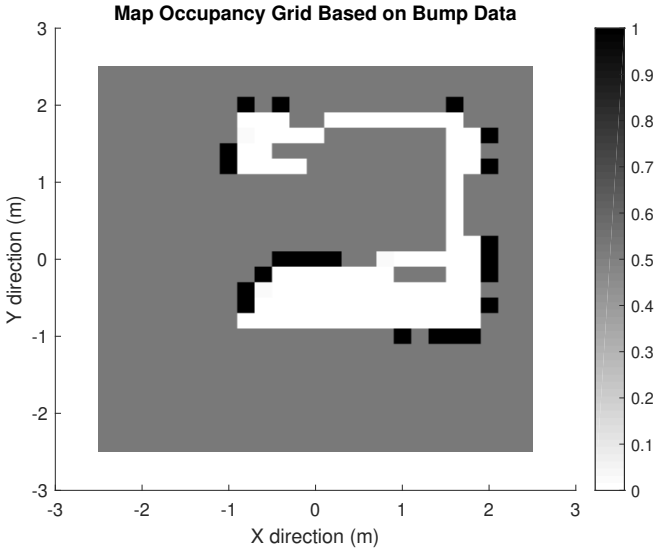



Figure 3: Bump Occupancy Grid at Timestep 2,[25x25]

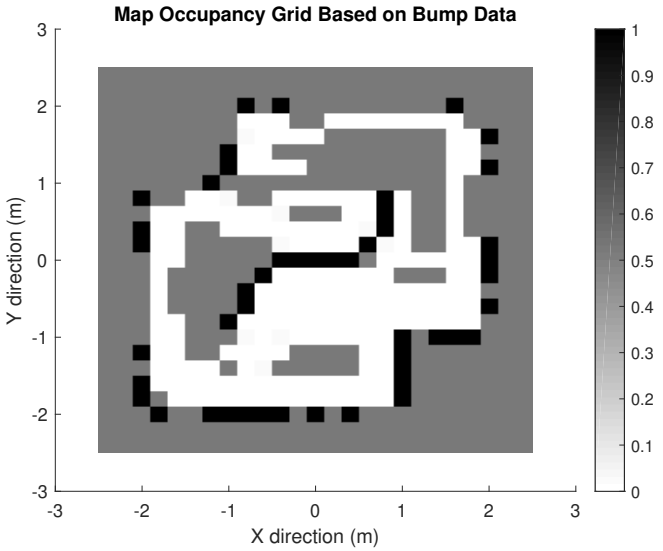
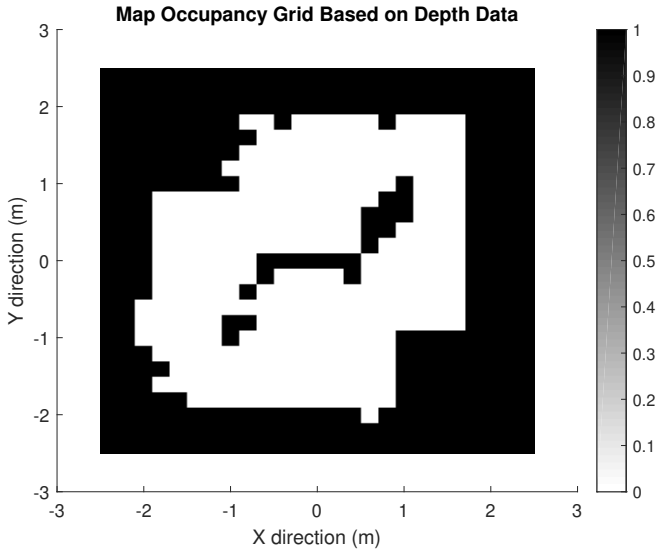



Figure 4: Bump Occupancy Grid at Timestep 3,[25x25]

5. For this case, I would apply a probability greater than a specific threshold that made sense for the sensor model. In this case, I was able to use the threshold as .5, which means the program can simply round each probability.As we can see, the unknown areas are assumed to be walls which makes sense as there is no way for the robot to make it past the walls. I also did this for both the 25 and 50 to see if they both worked well!




t

Figure 5: Bump Occupancy Grid Occupied or Free,[25x25]

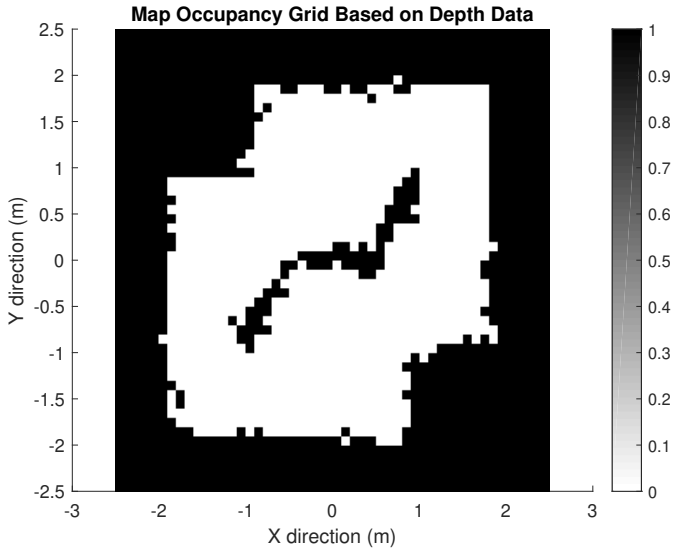



Figure 6: Bump Occupancy Grid Occupied or Free,[50x50]

6. Please see Figures 7, 8, and 9 for plots! With the 50x50 plot, I notice a great amount of space that appears unknown due to the robot not driving over each grid. Although each bump location is more precise (less variance), this accuracy leaves many gaps in the wall as the robot was unable to hit every single grid. This map would need to be processed with a script that could assume walls in between very close wall-occupied grids. Otherwise, the 50x50 would need the robot to bump into the walls even more times to get a full description of the map.

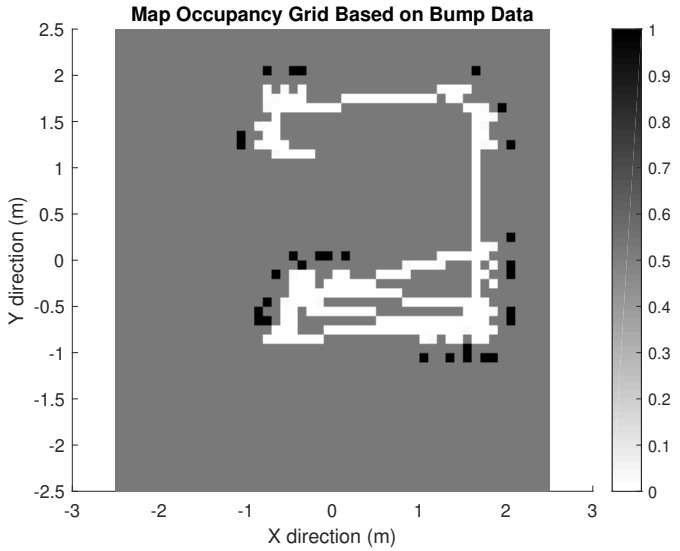



Figure 7: Bump Occupancy Grid at Timestep 1,[50x50]

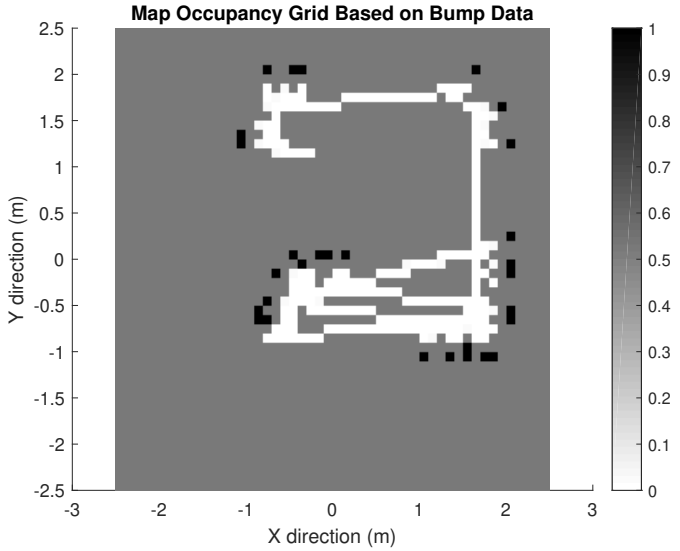



Figure 8: Bump Occupancy Grid at Timestep 2,[50x50]


Figure 9: Bump Occupancy Grid at Timestep 3,[50x50]

7. The first difference that I would see is slippage and a time delay from when the robot touches the wall to when the bump sensor is activated. In Lab 1 and 2, I noticed significant delay where the robot would hit the wall with the wheels slipping as it pushes against the barrier. Additionally, there would be an uncertainty of the position which was not present in the simulation. Finally, there is a possibility of both of the bump sensors being activated at once which would make the program confused as to where the wall was exactly. I checked both lab 1 and 2 data and did not see this event occur, but I believe it is possible for this event to occur.

# Mapping - Occupancy Grid with Depth Information

1. The inputs are:
    - **datastore**: struct from running SimulatorGUI
    - $l_0$: initial log odds
    - **NumCellsX**:number of cells in the x direction (integer)
    - **NumCellsY**:number of cells in the y direction (integer)
    - **boundaryX**:boundary of the environment in the x direction. 1 x 2 array [$min_y$ $max_y$]
    - **boundaryY**: boundary of the environment in the Y direction. 1 x 2 array [$min_y$ $max_y$]

The Output is

- **Map**:[x,y] matrix of the map with boundaries

2. I used the same type of sensor model as in the bump occupancy grid program. For every wall detected, the program adds the intmax and for all grids in between the robot and wall the program subtracts $\frac{intmax}{4}$. This makes sense in the simulation because of the range sensor is exact in its' measurements and thus we have an extremely likely case that the sensor is correct. Additionally, I tested out a few different numbers to try and this worked the best and produced some very nice-looking plots as seen below!

3. For the depth sensing, I decided to use four different time steps which are $\frac{1}{15}$,$\frac{1}{3}$, $\frac{2}{3}$, and the full duration. I did this due to the accuracy of depth sensing in addition to the length of time I allowed my simulation to run (25 minutes). What might look odd is how the timesteps 1 and 2 look very similar. This is due to the robot trajectory I created; Half way through the program, the robot changes directions and thus follows a similar path backwards.

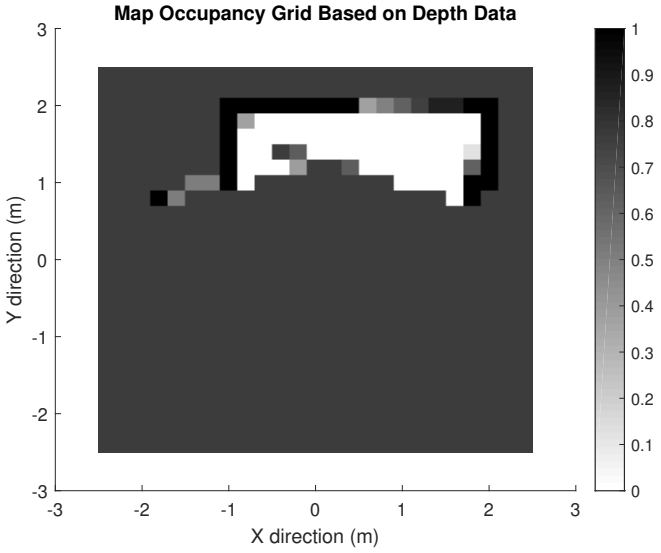



Figure 10: Depth Occupancy Grid at Timestep 0,[25x25]

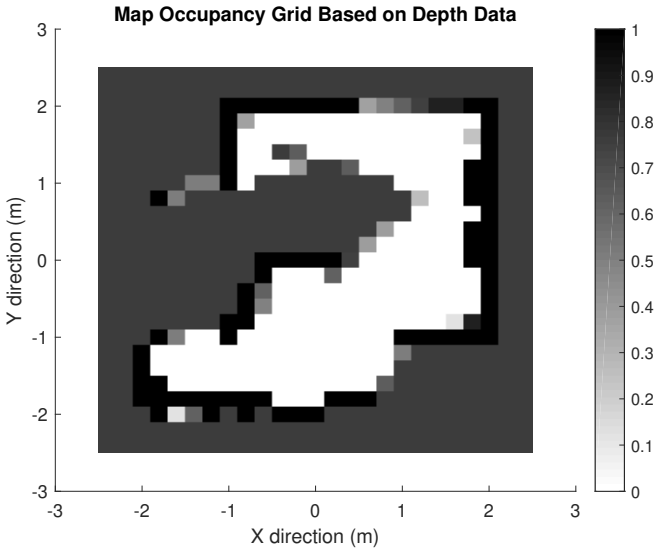



Figure 11: Depth Occupancy Grid at Timestep 1,[25x25]

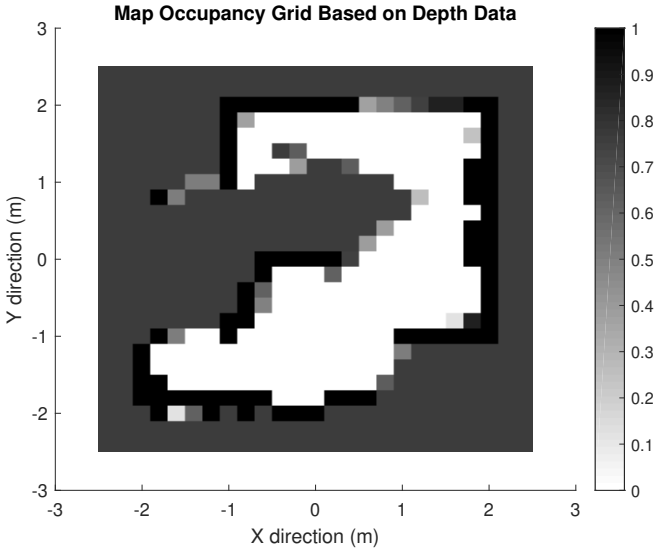



Figure 12: Depth Occupancy Grid at Timestep 2,[25x25]

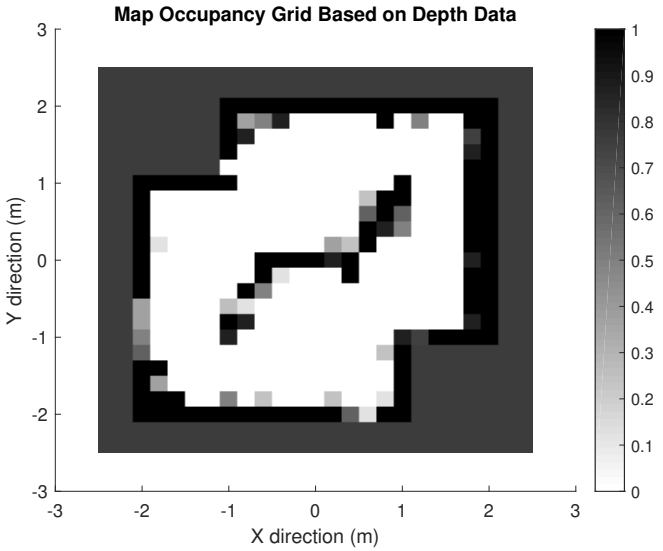



Figure 13: Depth Occupancy Grid at Timestep 3,[25x25]

4. The notes from Part 3 apply here as well! The depth occupancy grid using 50x50 looks incredible to me! I was really happy how accurate this program was with the larger number of grids. The walls appear to be more precise and have less uncertainty to the exact location of the walls. To see if this scales, I decided to try to 100x100 as well and it took around a minute to run. As shown in Figure 18, it appears that the depth sensor occupancy grid does scale fairly well, but there is a huge trade off of efficiency.

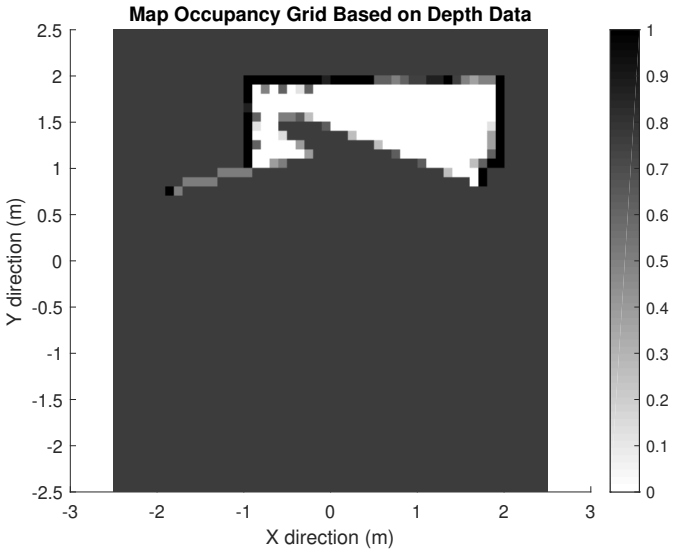



Figure 14: Depth Occupancy Grid at Timestep 0,[50x50]

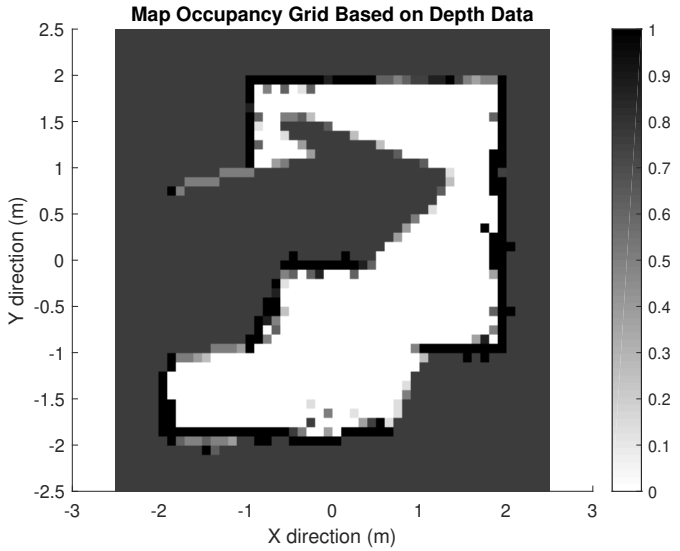



Figure 15: Depth Occupancy Grid at Timestep 1,[50x50]

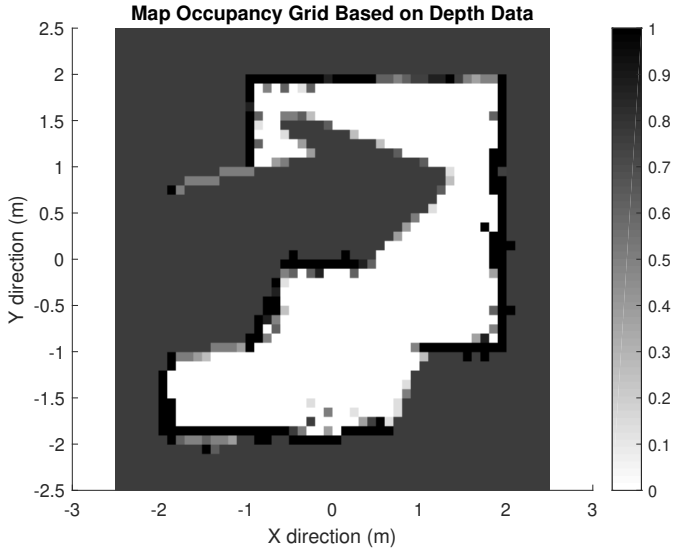



Figure 16: Depth Occupancy Grid at Timestep 2,[50x50]

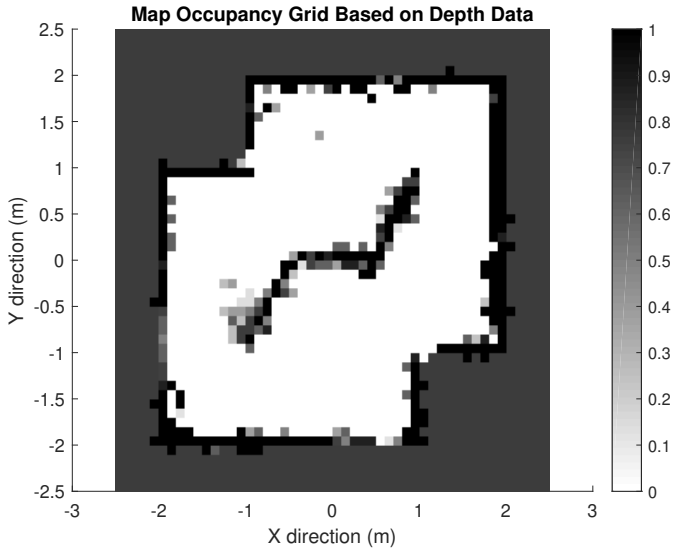



Figure 17: Depth Occupancy Grid at Timestep 3,[50x50]

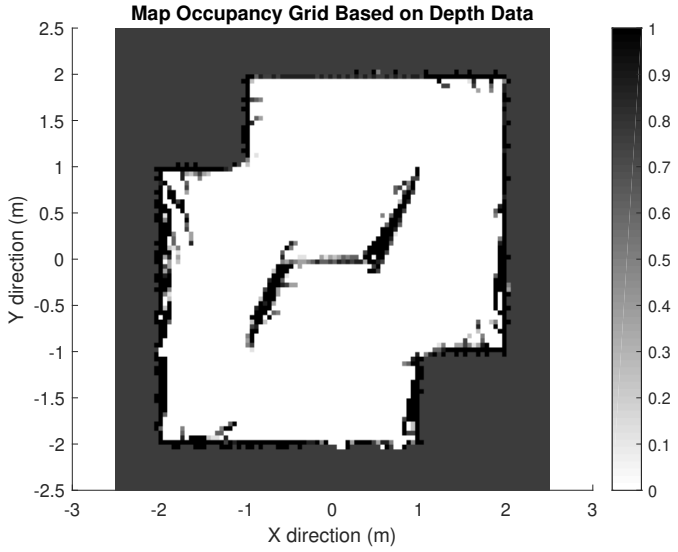



Figure 18: Depth Occupancy Grid at Timestep 3,[100x100]

5. As briefly mentioned in section 4 and shown in Figure 18, the occupancy grid looks nicer and is more accurate as you increase the number of rows and columns. Additionally since this write up takes a while, I decided to run a 200x200 grid to see if I could see any issues with scaling up! My hypothesis was that there could be areas that were never covered by the range sensor due to the large number of grids. To explain by example, I imagine if you have a low number of grids, the range sensors will overlap on grids and

predict that they are free. In contrast, if the number of grids was massive, the range sensors would not overlap and have lines of free columns that are no longer marked as free. Additionally, this program took over 5 minutes to complete!

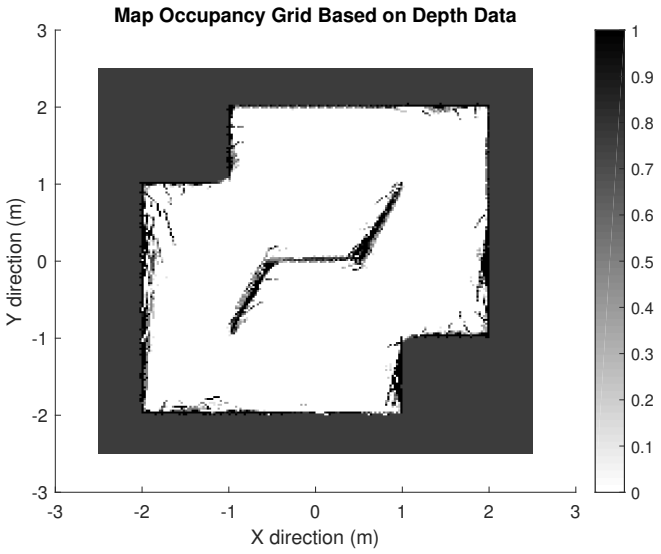



Figure 19: Depth Occupancy Grid ,[200x200]

When we compare Figures 17, 18, and 19, we see that Figures 18 and 19 are able to be more precise for the wall location specifically with the middle wall. On the other hand, if we focus our attention to [x,y] = [-2,.5] we see areas of uncertainty near the wall in straight lines. I attribute this to the range sensor "missing" some of the free grids due to the extreme number of grids (40,000!). As shown, there are various trade offs to having too large of a number of grids in the map in addition to strictly effeciency.

6. The occupancy grid using range sensor was much better overall in wall accuracy and marking free cells as free. The bump occupancy grid ran into the issue that the robot has to walk over every single grid in the map to fully explore it. On the other hand, the range sensor was able to pick up many grids in one shot and thus was able to fill in the free grids much faster.

7. In the simulation, the range sensor is extremely accurate and there were no "unexpected" or inconsistent wall readings that would be present on a real robot. With a real depth sensor, there is a normal noise around each measurement in addition to the need to be calibrated correctly. For a real sensor, we would need to use a much more precise inverse sensor model than we used here.

# Homework 5

## Test function

Please see the matlab files for the test grid program.