Siyuan Yu

ME-495: ML&AI

Professor Argall

**HW1: Search and Navigation Part A**

**1) Building a Grid**

(See Code)

**2) Designing a Cost Function**

Since our algorithm is an “online” A\*, only nodes that are neighboring the node that the robot is currently in can be explored. This means that for any given grid location that the robot is in, only the 8 direct neighbors around that grid location can be expanded.

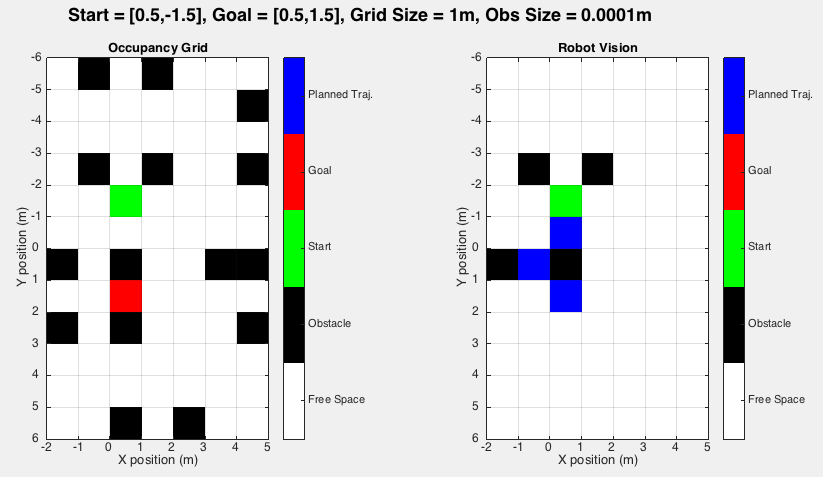
The simplest heuristic (h(n)) for our case would be to apply the distance formula on the current and goal grid indices (always integer amounts). Given that the true cost (g(n)=cumulative true cost) of entering an unoccupied grid is +1 and entering an occupied grid is +1000, our distance formula heuristic will always be less than or equal to this true cost. This means that the distance formula heuristic is admissible.

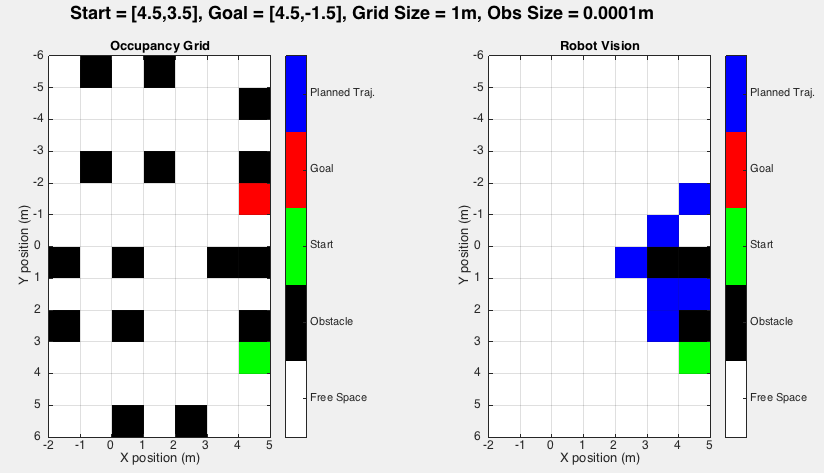


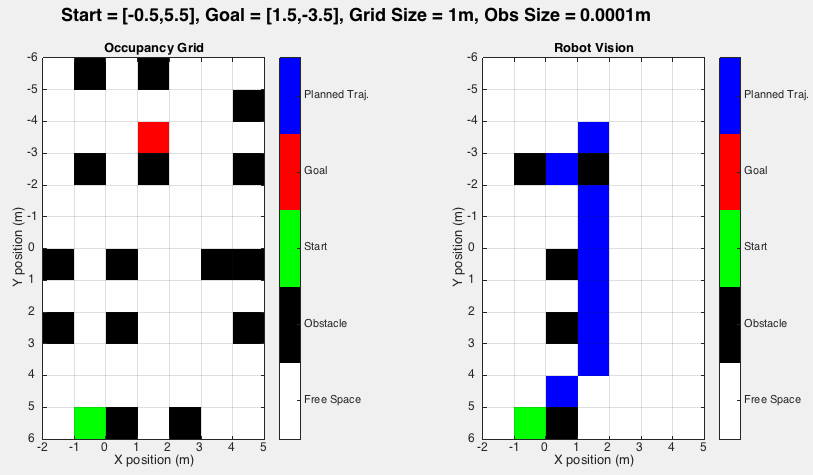


**(see next page)**

**3) Planning Paths (Grid Size = 1m)**



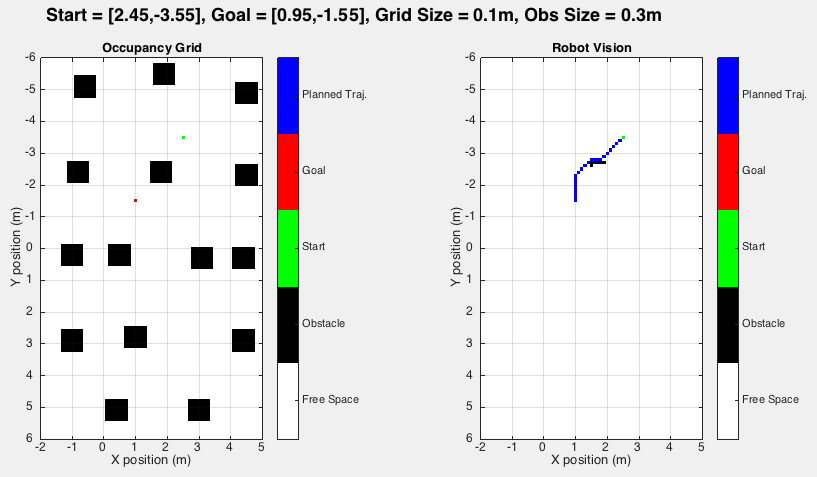


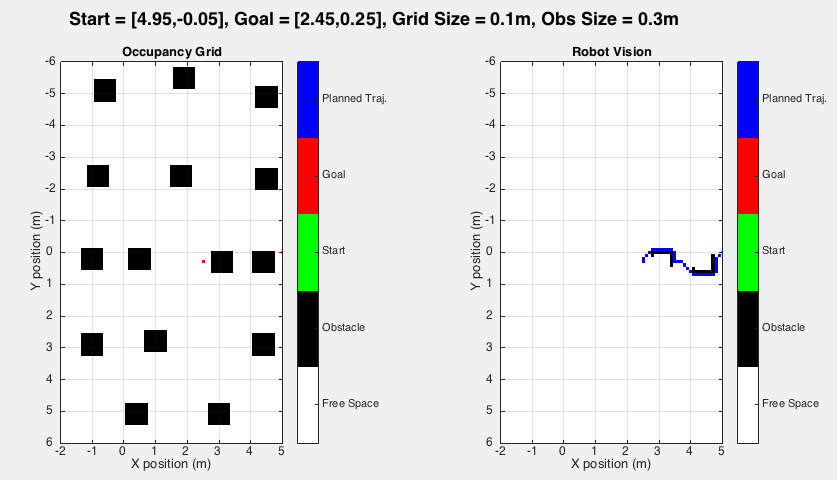


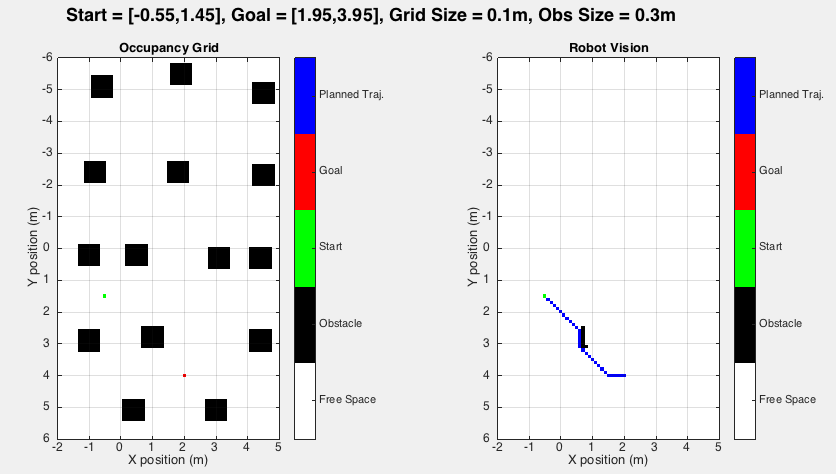
**4) Chasing Grid Size and Inflating Landmark Size**

(See Code)

**5) Planning Paths (Grid Size = 0.1m)**







**HW1: Search and Navigation Part B**

**6) Inverse Kinematic Controller**

For the inverse kinematic controller, I decided to separate all robot motions into individual sequences of either rotation only movements or forward only movements. The following pseudo code explains how my inverse kinematic controller operates.

Initial

Given: 

**//Target distance for forward movement**



**//Target heading for rotation movement (k depends on the quadrant)**



Inverse Kinematic Controller (applies for both v and w)

**//Time step**



**//Find the constant acceleration needed to traverse half the target distance**



**//Recalculates for a reasonable acceleration if it is over the max acceleration**

If 

Find a v’ that is less than v’max that can cover half the target distance in n\*ts, n=integer

End

**//Creates a series of [v,w] that achieves a forward sequence of distance dqt**



While 

If 



Else



End



End

Main Code

If 

Applies controller to calculate a forward motion sequence (v,w=0)

Else

Applies controller to calculate a rotation motion sequence (v=0,w)

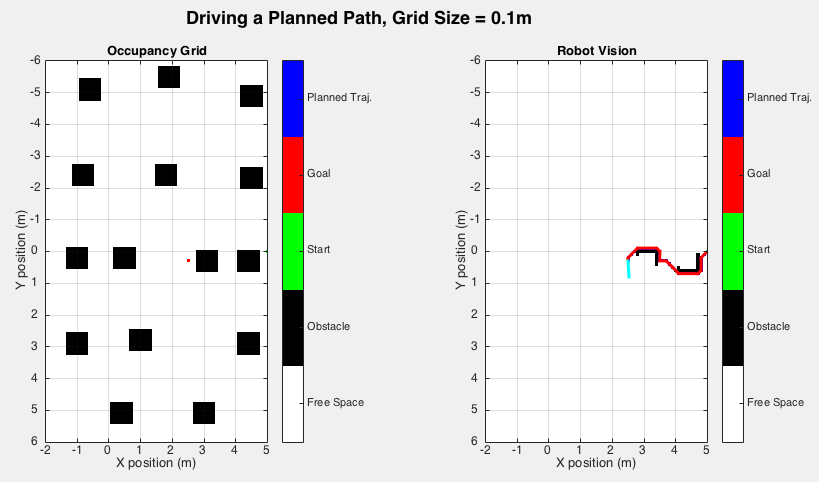
Applies controller to caluclate a forward motion sequence (v,w=0)

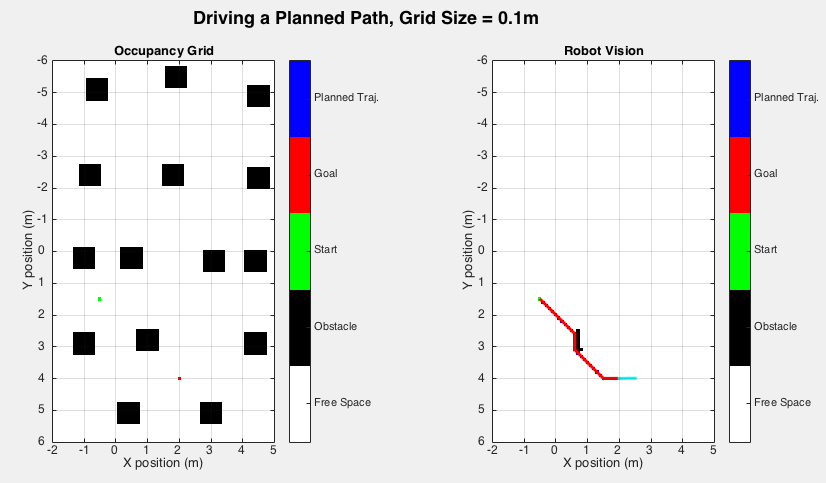
End

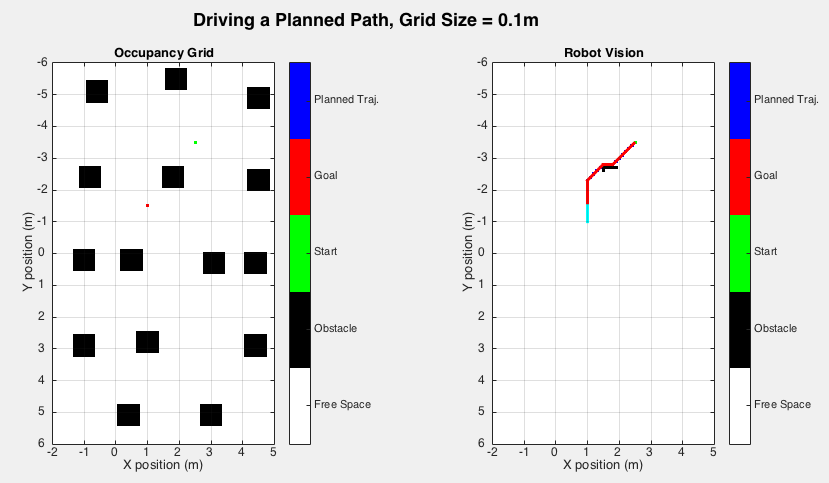
**7) Driving Planned Paths, Grid Size = 0.1m (SEE CODE FOR ANIMATION)**

Because we are given a planned path to operate with, forward motion sequences with the same headings can be chained together so that the robot will not have to stop between certain motion sequences. Like I mentioned earlier, the main challenge of driving the planned paths was making sure that the proper velocity controls were generated and given to the robot so that the robot would accelerate and decelerate appropriately to end up at the target grid location with no angular or translational velocity. As seen from the animations (see code), the controller did a good job in maintaining this behavior.

One surprising behavior is that some of the rotational sequences look very funky because my controller was not programmed to rotate efficiently and it does not choose the smaller angular motion every time.

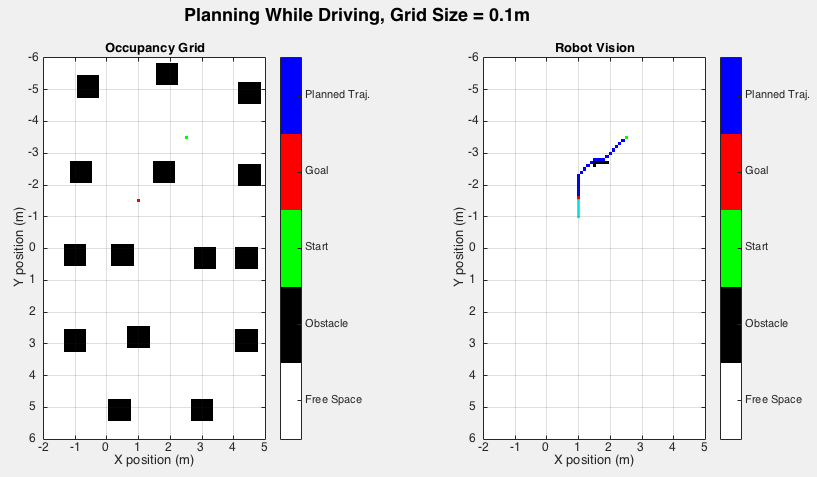


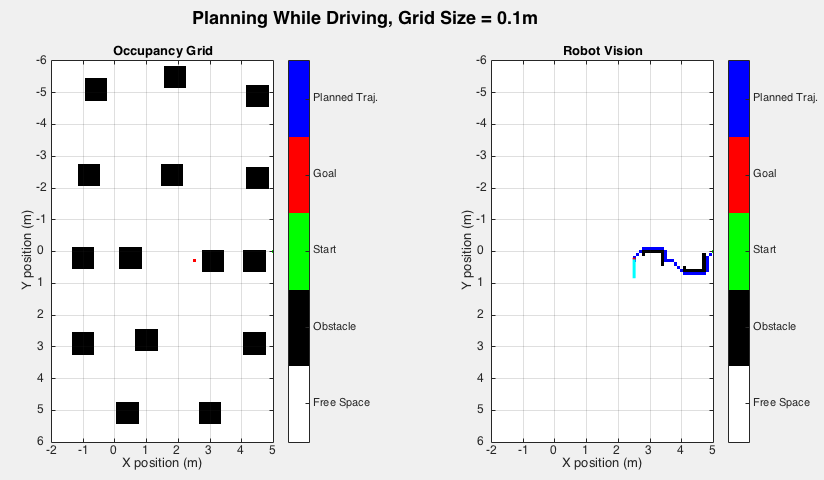


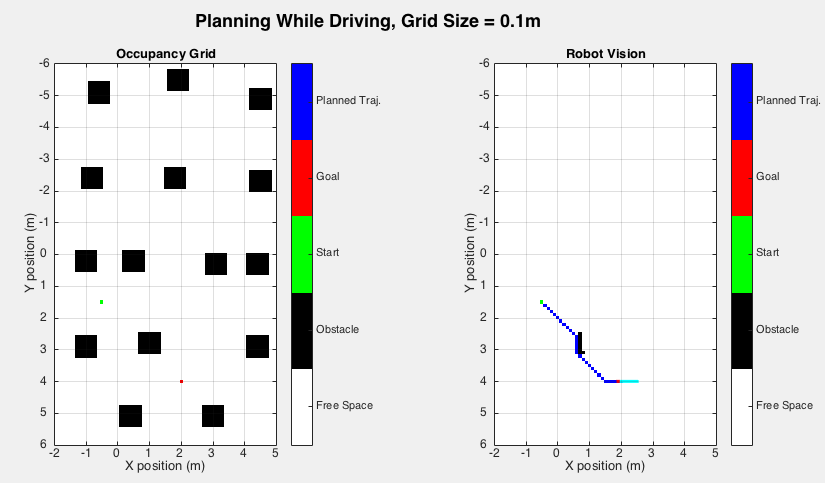


**8) Driving While Planning Paths, Grid Size = 0.1m (SEE CODE FOR ANIMATION)**

In this scenario, the robot can no longer look into the future and chain together similar motion sequences like it did previously in question 7. This means that between EVERY sequence of motions, the robot has to come to a complete stop to explore its neighbors. As seen in the animations (see code), this increased the robot’s operation time for each goal because the robot would have to stop at each grid, evaluate A\* to plan the next step, and then move to that next step before rinsing and repeating. Even though this method of operation is more time consuming, it is a more realistic application because sensors have limited ranges and onboard planning is more robust in terms of adapting to different environments.



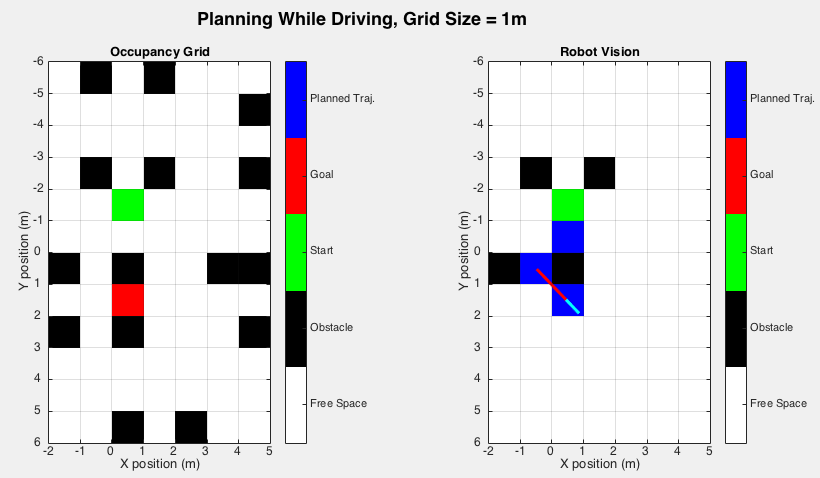


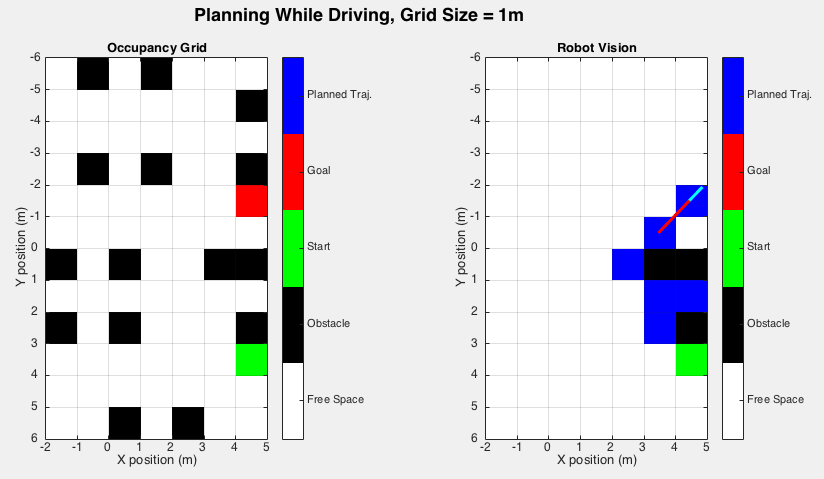


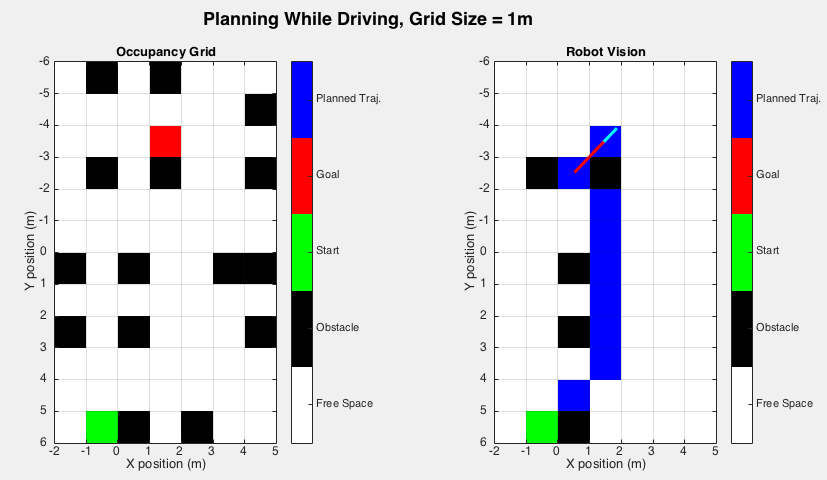
**9) Driving While Planning Paths, Grid Size = 1m (SEE CODE FOR ANIMATION)**

Driving in the larger grid resulted in faster operation times for the robot. Because there is a larger distance between grids and less cells in general, the number of times that the robot has to stop and plan is greatly reduced. Also, these greater distances between grids mean that the robot can accelerate to faster speeds before needing to decelerate again to a stop at the next grid location. All of this means that the robot can operate much faster with a grid size of 1m when compared to a grid size of 0.1m.

On the other hand, the advantage of the smaller grid size is that the A\* algorithm has more resolution and can plan a shorter and more efficient overall path for the robot to navigate to the goal.







**7-9) Overview**

|  |  |  |  |
| --- | --- | --- | --- |
| **Case** | **Status** | **Grid Size (m)** | **Time (s)** |
| **Problem 7: Step 5A** | **Planned** | **0.1** | **40.4** |
| **Problem 7: Step 5B** | **Planned** | **0.1** | **61.0** |
| **Problem 7: Step 5C** | **Planned** | **0.1** | **43.1** |
| **Problem 8: Step 5A** | **Onboard** | **0.1** | **102.6** |
| **Problem 8: Step 5B** | **Onboard** | **0.1** | **152.2** |
| **Problem 8: Step 5C** | **Onboard** | **0.1** | **136.3** |
| **Problem 9: Step 3A** | **Onboard** | **1.0** | **15.0** |
| **Problem 9: Step 3B** | **Onboard** | **1.0** | **34.6** |
| **Problem 9: Step 3C** | **Onboard** | **1.0** | **42.0** |

**10) Pros and Cons and Real World Factors**

The biggest advantage of this simple controller in a simulation is that it does not need to be tuned and will always stick to the planned path and avoid all obstacles perfectly. The biggest drawback of this controller is the fact that it does not create a smooth trajectory and is therefore less efficient in terms of time and expenditure of energy because the robot has to come to a complete stop between every sequence of motions.

In our scenario, we assumed that we had perfect world dynamics, a perfect motion model, true positions for the world and all of its obstacles, and no error in our sensors or odometry. In a real world environment, things like slip in the wheels, noise in odometry and sensor readings, and discrepancies between control inputs and actual robot performance can all confound the robot’s operation. More sensors (combined with probabilistic methods) should be applied to give a more accurate idea of the robot’s true position at any given time, instead of assuming perfect world dynamics and relying only on your motion model. In order to account for sensor noise, filters should be applied as well to provide more accurate information about the world.