CSE 468/568 Assignment 9
Grid Localization

The objective of this assignment is to make you familiar with Grid Localization which is a variant of discrete Bayes Localization. In this method, the map is an occupancy grid. At each time step, the algorithm finds out the probabilities of the robot presence at every grid cell. The grid cells with maximum probabilities at each step, characterize the robot's trajectory. Grid Localization runs in two iterative steps: Movement and Observation. For each movement, you should compute the probabilities of moving between grid cells. For each observation, you should find the most probable cells where that measurement could have occurred.

Map

The world is 20m x 20m, and it should be discretized into a grid of 400 cells (20 rows and 20 columns). Cells have the size of 1m x 1m and are represented with (X, Y) where (0, 0) would be the bottom left corner of the map.

Motion and Observation Model and Noise

Normally a simple motion consists of turning, and heading forward, and an observation could consist of a range and bearing (e.g. observing a landmark). Which means apart from discretizing the position into cells, the heading should also be discretized into pre-defined values. e.g. 0, 45, 90, 135, 180, 225, 270 and 315 degrees. Each turn will result in one of the 8 possible directions, with different probabilities. This means that a 3 dimensional map should be considered, where the third dimension covers the robot's heading. In this example. the map size would be (20 x 20 x 8), and the value of cell (5, 7, 3) means the probability of the robot pose being: x=5 y=7 heading=135

The choice of 45 degree was arbitrary, one can discretize with any desired value (10 degree, 20 degree, etc.) This is similar to discretizing a real map into a grid, where the size of a cell in meters defines the resolution.

The motion model of a simple robot is (rotation, translation). In Grid Localization, for the purpose of moving robot between cells, the motion noise should be adjusted. Robot needs a translation and rotation noise for movement and also a noise for its observations (e.g. range and bearing). A sample selection for this purpose is half the cell size. So for the example of 1m x 1m cells and 45 degree discretization, range and translation noises are 0.5m, and bearing and rotation noises are 22.5 degrees.

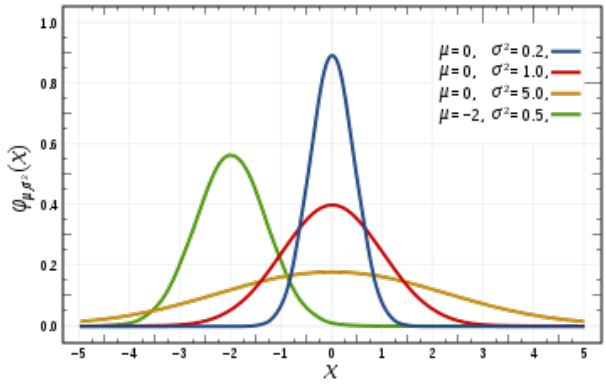

Figure 1: Gaussian Distribution

It is very common to make use of Gaussian Distribution (Normal Distribution) to add noise into motion. This is illustrated in Figure 1

$$f(x|\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$

- Two parameters: mean μ and standard deviation σ
- Given a data point x, one can get how *probable* the value is in the Gaussian distribution for a given mean and standard deviation
- For every control input, a single Gaussian distribution can be considered for each of the motion parameters (rotation, translation).
- This function can be implement, but library functions are also available. (e.g the numpy module)

Our Motion Model

In this assignment, we will consider a simplified discrete noise model. The robot can take a single step in any of the four directions. Each movement will be successful with probability of 0.6, it may fail moving the robot with probability of 0.2 (robot stays where it is) and it may take the robot too far (2 cells instead of 1 cell) with probability of 0.2. In case of proximity to the wall (boundary of the map), robot does not move to the unavailable cell. I.e. it hits the wall ans stays in the last cell. So the unavailable cell's probability should be allocated to the previous cell. Figure 2 shows the state transition in open area and proximity to the wall. In this figure the robot is making a move to the right.

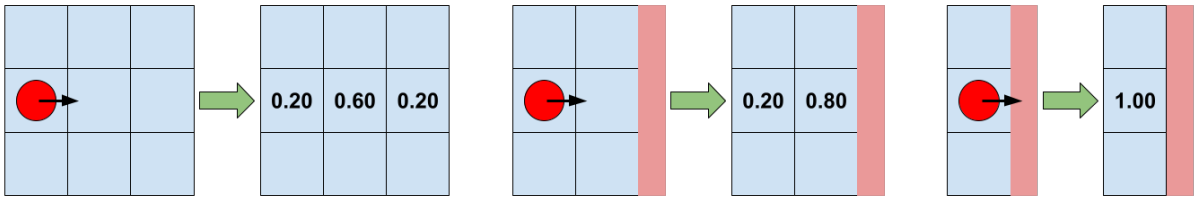

Figure 2: Motion model for taking a step to the right.

Our Observation Model

In this assignment, we will consider a simplified discrete noise model. Each observation is accurate with the probability of 0.4, it can be off by 1 cell with probability of 0.2, and it can be off by 2 cells with probability of 0.1. X and Y coordinates have independent observations, and both follow the same model. An example of an X coordinate observation is shown in Figure 3.

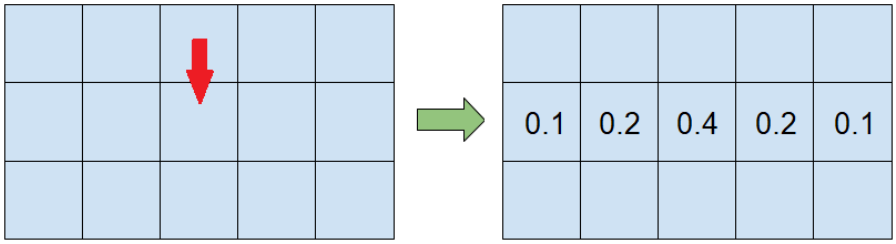

Figure 3: Observation model for an X coordinate.

Bayes Filter

Essentially, every iteration of the Bayes filter has two steps:

- Prediction Step - where you incorporate the control input(movement)
- Update Step - where you incorporate the observation

A pseudo algorithm for the Bayes filter is described in Figure 4. Prediction Step (Lines 9-11) increases uncertainty in your belief while the update step (Lines 2-8) reduces uncertainty in your belief. Ideally, we would want the observation to be the last message at the end of our entire state estimation process, as it aims to reduce uncertainty in our robot's belief.

1. Algorithm **Discrete_Bayes_filter**(*Bel(x), d*):
2. $\eta = 0$
3. If *d* is a perceptual data item *z* then
4. For all *x* do
5. $Bel\_new(x) = P(z|x)Bel(x)$
6. $\eta = \eta + Bel\_new(x)$
7. For all *x* do
8. $Bel\_new(x) = \eta^{-1}Bel\_new(x)$
9. Else if *d* is an action data item *u* then
10. For all *x* do
11. $Bel\_new(x) = \sum_{x'} P(x|u, x')\ Bel(x')$
12. Return *Bel_new(x)*

Figure 4: Bayes filter

Notice that in each iteration of the Bayes filter, you will need to go through all possible states to estimate the belief. If you consider the original problem with heading, and your discretization is such that you have $20 \times 20 \times 8$ possible states, then there is a lot of computations involved in each iteration of the Bayes filter. Hence, you should be efficient in writing your code, especially in Python, if you want your entire estimation process to run within a couple of minutes. Shorter running times may prove beneficial for tweaking and debugging your algorithm implementation. Some ways to reduce processing time:

- Reduce unnecessary interim variables. This can lead to really slow processing times, especially in Python.
- Use numpy for faster matrix operations instead of element wise operations. Numpy is faster if you can use matrix like operations because the processing happens in C.
- If the probability of a state (grid cell) is 0, then we can skip that state in the inner loops of the Bayes filter (i.e only in multiplicative terms, since multiplying any value with a 0 results in a 0). In fact, one may consider a non-zero threshold; if a state has a probability less than the threshold, you can skip it and hence you don't have to process all possible states. However, since some states are being skipped, the probabilities of all cells might no longer add up to 1. So you need to make sure to normalize the values at the end of both prediction step and update step, to maintain the sum of the probabilities across all states to be 1.

NOTE: In this assignment, you will consider the X coordinates and Y coordinates to be independent. This is a **very** unrealistic assumption for most applications. However, it will simplify this assignment. This means instead of keeping track of each individual cell, you can keep track of X and Y coordinates separately. X has a probability distribution between 0 and 19. Y has an independent probability distribution between 0 and 19. You can pick a probability threshold of 0.05 for each coordinate, before ignoring it. Remember that the total probability of each coordinate should still add up to 1.

Input and Output

Your program should subscribe to a topic `'robot'` which represents the action commands and sensor readings. Each message is of type `string`, which consists of a single character, followed by an integer. You can publish using command line when testing (`rostopic pub`). Your code then must parse the string to extract information. You can then use this along with noise model. (Suggestion: capitalize the first letter in your program).

Commands: Commands can start with any of the 4 letters `L, R, U, D` for Left, Right, Up, and Down. The following number means how many steps in that direction. E.g. `R4` means the robot is taking 4 steps to the right. You can break multiple steps into individual steps, and apply the prediction step multiple times.

Observations: Observations can start with either `X` or `Y`, followed by the coordinate value. E.g. `X6` means that the robot has an observation that its X coordinate is 6. Remember to apply the observation noise model. In this example, most likely the reading comes from X=6. But it may also come from 5 or 7 and less likely from 4 or 8.

Your program should show the initial map, with robot in the center of the map (10, 10). Upon receiving each message, the plot should be updated to represent the current belief of the robot. The output should be a graphical map using `matplotlib` similar to Figure 5.

NOTE: This map is for 2 dimensions, the required part of the assignment only needs you to implement in 1D!

Test to see if you can launch everything by running the command:

```
$ roslaunch lab9 lab9.launch
```

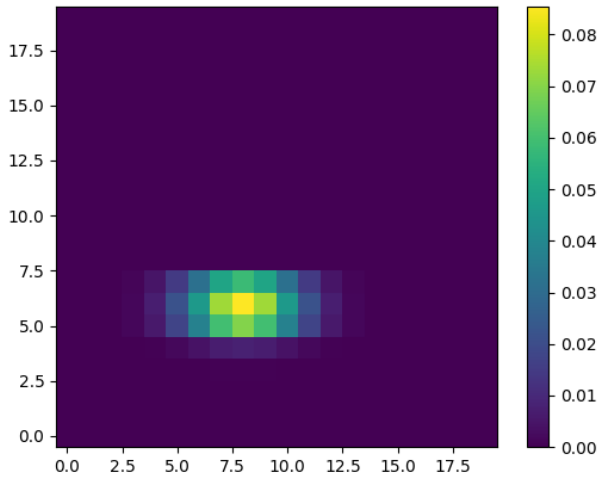

Figure 5: Sample Output

Submission Instructions

You will submit `lab9.zip`, a compressed archive file containing the lab9 package (folder). The folder should contain a launch file `lab9.launch` and your node in appropriate subfolders. The folder should compile if we drop it into our catkin workspace and call `catkin_make`.
Please take care to follow the instructions carefully so we can script our tests. Problems in running will result in loss of points.

Please use UB Learns for submission.
The assignment is due Friday, May 13 before midnight.