CSE 468/568: Robotic Algorithms

Grid Localization using Bayes Filter

VERSION 1
PA3, Fall 2019
DEADLINE: November 7, 2019 11:59 pm
INSTRUCTOR: Vivek Thangavelu
WRITTEN BY: Vivek Thangavelu and Zijian An

I. Objective

The objective of this assignment is to make you familiar with Grid Localization which is a variant of discrete Bayes Localization. In addition you will learn how to use ROS Bags and visualize in Rviz..

Please read the entire document at least once before you begin your implementation. Refer to the lecture slides for more information on Bayes filter. As always, there is a FAQ section towards the end of the handout.

II. Description

1. Supplied Files

The **ros_pa3.tar.gz** compressed file contains:

- **Grid.bag -** The ROS bag file that contains the observation and motion data as messages
- **markers_example.py -** An example python node that contains helper functions to display *markers* in rviz. You may use the helper functions directly in your code and tweak them accordingly. Please read through the comments in the code and test it out to understand how you can use the helper functions in your code.

2. Deliverables

You need to write a node to implement grid localization. Grid Localization 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's presence at every grid cell. The grid cells with maximum probabilities at each step, characterize the robot's trajectory. Grid Localization using Bayes filter runs in two iterative steps that utilize the movement data (prediction step) and observation data (update step). You will submit **ros_pa3.tar.gz**,

a compressed archive file containing the **ros_pa3** package folder, the bag file and all the nodes required to perform grid localization using Bayes filter. The folder should contain:

- **pa3.launch:** Running it should read the bagfile, run the grid localization algorithm and display visualizations in rviz as follows:

    1. A cube plotted for each of the tags (landmarks) at the appropriate locations as given in the description.

    2. A marker of type LINE_STRIP which represents the path/trajectory that the robot has taken. For each step in the Bayes Filter (i.e after each measurement update), add the point with the maximum probability to the line strip and publish to rviz.

- **estimation.txt**: Each line in the text file contains the most probable state after incorporating each movement and measurement data.
  For the most probable state after incorporating the movement data, output a line of the format ('P' stands for prediction):

  `P: (x, y, θ)`

  For the most probable state after incorporating the measurement data, output a line of the format ('U' stands for update):

  `U: (x, y, θ)`

  Here `(x, y, θ)` is the robot's pose expressed in the continuous world.

III. Initial Setup

1. Download and extract the supplied files for PA3
   (SHA256: f55442cd947818032be741f986a934647bdb2931a73bcdd2cb6deb29060ee748)

2. Create a new package called **ros_pa3**

```
cd ~/catkin_ws/src
catkin_create_pkg ros_pa3 std_msgs geometry_msgs rospy roscpp
cd ~/catkin_ws
catkin_make
source ~/.bashrc
```

IV. Concepts

1. Occupancy Grid

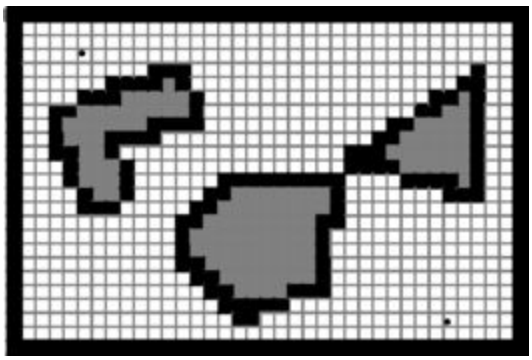


Fig.1: Occupancy Grid map

Grid Localization discretizes the world that's usually represented in a continuous form. Figure 1 is an example of a 2D occupancy grid. For PA3 you should make a grid for **7m×7m** coverage. Your cell size should be **20cm×20cm**. You also should assume a dimension for the robot's heading. So your grid is 3 dimensional. The third dimension covers the robot's heading. You can discretize that with some specific value (10 degrees, 20 degrees or more). The initial position of your robot within the grid is **(12, 28)** and the initial heading is **200.52 degrees**. The robot's first pose in the 3rd dimension depends on your selected discretization size. For example, if your discretization size is 90 degrees, the cell number in the 3rd dimension will be 200.52/90+1 = 3. So the robot's initial pose is (12 , 28, 3). (Notice that it is starting from one).

Implement methods to convert between the continuous world representation and your grid system. Each cell in you grid stores the belief of how probable that state is. Hence, the sum of the probabilities of each grid cell should be equal to 1. For initial conditions, you may assume a point mass probability distribution i.e. the initial cell to have a probability of 1 and 0 everywhere else. An other possible initialization would be to initialize a gaussian around the initial cell with the initial cell being the mean.

2. Landmarks

For PA3, there are six landmarks in the robot map, and they are at the following locations:

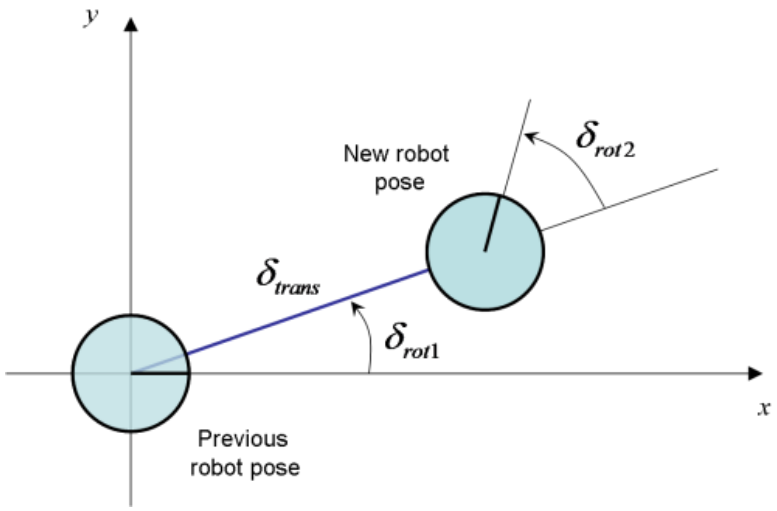
- **Tag 0:** x=1.25m, y=5.25m

- **Tag 1:** x=1.25m, y=3.25m
- **Tag 2:** x=1.25m, y=1.25m
- **Tag 3:** x=4.25m, y=1.25m
- **Tag 4:** x=4.25m, y=3.25m
- **Tag 5:** x=4.25m, y=5.25m

The robot moves in the area within these landmarks, observing some at any given time.

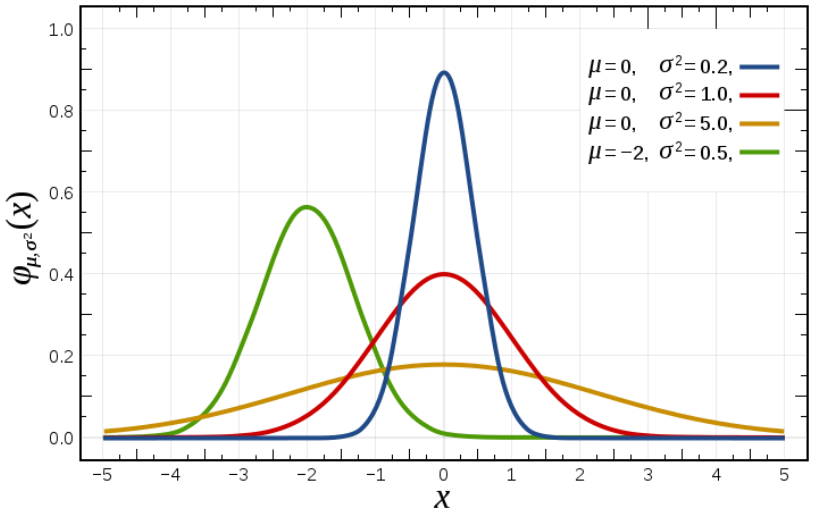
3. Sensor and Motion Model

The **observation data** consists of *range, bearing* and *tag_num*. The robot observes a specific tag identified by the *tag_num* (0-5) where *range* is the distance between the specific landmark and the robot, and *bearing* is the angle between the landmark and the robot heading direction.



You should utilize the **odometry motion mode**l for this assignment as the movement data is given by the relative odometry information described by the motion parameters *rotation1, translation* and *rotation2*.

4. Noise



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

In PA3 we make use of Gaussian Distribution (Normal Distribution) to add noise into Motion and Observation:

- Two parameters: mean μ and standard deviation σ
- Given a data point x, we can get how "probable" the value is in the gaussian distribution for a given mean and standard deviation
- For every control input, we can consider a single gaussian distribution for each of the three motion parameters (rotation1, translation, rotation2).
- Similarly, for every measurement data, you can consider a gaussian each for *range* and *bearing*
- You can implement this function with your own code or use a library function. (For Python you can use the *numpy* module)

In Grid Localization, for the purpose of moving robot between cells, the motion and observation noise should be adjusted accordingly. You need a translation and rotation noise for movement, and range and bearing noise for your observation. A good choice for this purpose is half the cell size. So for the example of 20×20 cells and 90 degree discretization, range and translation noises are 10cm, and bearing and rotation noises are 45 degrees. You will need to tweak the noise parameters to improve your state estimation. Usually measurement data is "less noisier" than movement data.

5. ROS Bag

A Rosbag is used for dumping the messages published on certain topics to files for later use. It helps us to work offline by running a physical/simulated robot once and collect the data during the run. For this assignment, we have provided a bag file for you which includes the movement and observation information from one scenario. You need to read the bag file to simulate your localization algorithm running on a robot executing this scenario. To get information about the bag, use the rosbag command line tool: `rosbag info <bag_name>.bag`

In the bag file there are two types of messages:

1. **Motion.msg** (for control data)

   int32 timeTag

   geometry_msgs/Quaternion rotation1

   float64 translation

   geometry_msgs/Quaternion rotation2

2. **Observation.msg** (for observation data)

   int32 timeTag

   int32 tagNum

   float64 range

   geometry_msgs/Quaternion bearing

There are two ways to read the messages in the ROS Bag:

Approach 1: Play rosbag from command line using `rosbag play --clock <bag_file>.bag`. This will publish the messages in the corresponding topics in the order they were recorded. Write a node to subscribe to these topics and use the messages for state estimation. You will need to write new ROS message types as described above for this approach. This is a more involved approach and can be particularly difficult when debugging.

Approach 2*: Open the rosbag in you node (similar to how you would open a file) and directly read your rosbag for messages in specific topics. This method is particularly easy in Python because you do not have to even define the new message types (In Python, it should take you less than 5 lines to do this!!! Please try the running reference code for the given ROS bag. Reading the messages should be pretty straightforward).

6. 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 Algorithm 1. 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 (check your bag file which was the last message).

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

IMPLEMENTATION CONSIDERATIONS:

Notice that in each iteration of the bayes filter, you will need to go through all possible states to estimate the belief. If your discretization is such that you have 35×35×10 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 mins. A good rule of thumb is to keep the total running time for PA3 below 12 minutes. However, 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, if a state has a probability less than say 10^5, we can skip those states and hence we don't have to process all possible states. However, since you are skipping some states, the sum of the probabilities of each cell might no longer sum to 1. So you need to make sure you normalize at the end of your prediction step and update step (which you already do as per the algorithm), to maintain the sum of the probabilities across all states to be 1.

7. Rviz Visualization

Rviz is a visualization tool in ROS. It is essentially a node that can subscribe to certain types of messages and visualize them accordingly. Marker message is a type of ROS message that can be published from other nodes and visualized in Rviz. Markers may be cubes, arrows, cylinders, list of lines, list of cubes etc. The supplied file **markers_example.py** has example code to publish **marker messages (**list of cubes and list of lines). To visualize the messages, launch Rviz (either by including it in your launch file or typing "rviz" in your command line). In Rviz, under the *Display* panel, click *Add* and then select the *By Topic* tab. In the window, now you can select the topics that publish the marker messages (if any). The topic name is based on the topic you are publishing to in your publishing nodes (Check Section VII.4 for troubleshooting).

V. Implementation Tips

Implementation in robotics can be a daunting task with multiple sub-objectives. It is always a good idea to list out the sub-objectives for your task and modularize your code accordingly. Below is a set of sub-objectives that might help you organize your work for this PA. You do not have to follow this strictly and is presented here purely as an example.

1. Read the ROS bag (Use Approach 2. If you are using C++, you will have to define new messages types)
2. Write functions to convert a robot pose from discrete to continuous and vice-versa
3. Define a function for odom motion model
4. Define a function for sensor model
5. Implement Bayes filter

6. Use the Rviz helper functions to display the trajectory of the robot using the beliefs calculated in your Bayes filter (the most probable state is the grid cell with the highest probability). This will also help you debug your code and tune your noise parameters.

VI. Submission Instructions

We will be using autolab in this course to submit your programming assignments. In the autolab course page, select **pa3** and then submit a tarball of your entire **ros_pa3** folder. From the file viewer, you can right click **ros_pa3** and click compress to get a tarball of the entire folder. The deadline for submission is **November 7, 2019 11:59 pm**. The deadline will be strictly enforced so please submit once much earlier to test out the system. You are allowed to make multiple submissions. We will use the final submission version to grade your programming assignment.

Late Submission Policy

You may choose to submit the assignment late by a maximum of 2 days. Each day you lose 25% of the full grade. Hence, by submitting one day late, you lose 25% and by day 2, you lose 50%. The deadline for each extra day is **11:59 pm**.

VII. FAQ

1. **What will I learn from this PA?**

   At the end of PA3, you will have learned:

   a. How to make use of ROS Bags
   b. How to probabilistically estimate the state of a robot with noisy sensors and movements
   c. How to use Rviz for visualization

2. **How to transform from quaternion to euler?**

   ROS uses quaternions to track and apply rotations. A quaternion has 4 components (x,y,z,w). A quaternion is an another way to express a 3d rotation in space. You can convert a 3d orientation from quaternion to euler angles and vice versa. You may refer to the following links for example code:

   For Python: https://answers.ros.org/question/69754/quaternion-transformations-in-python/

   For C++: https://gist.github.com/marcoarruda/f931232fe3490b7fa20dbb38da1195ac

3. **How much noise is the right amount of noise?**

Ideally, we get the best results by empirically estimating or learning the noise parameters from the specific motion or observation model. This way we can refrain from using high noise models for an accurate sensor or low noise models for a less accurate sensor. In case of grid localization, the motion and observation noise should be adjusted taking into account the grid model.
If the amount of noise added is very small compared to grid cell size, you basically have a model with zero noise in the grid world. You can use a gaussian or any other noise model to generate noise given the data (For example, if you use gaussian distribution, the mean is the expected value and standard deviation signifies the amount of noise)

4. **I cannot see any markers in Rviz?**
    a. Check if Rviz is running
    b. Make sure the *Fixed Frame* under *Global Options* in Rviz's *Display* panel is the same as the coordinate frame of the marker messages

    `marker.header.frame_id = "/map"`

    (Refer the example marker code for more information)
    c. Make sure you keep publishing marker messages to Rviz since the markers are time dependent and will 'decay' over time
    d. Make sure you understand how a LINE_STRIP maker works as indicated in the helper code and the Rviz marker wiki
    e. Make sure you are subscribing to the right topics in Rviz based on the topics that your node is publishing to

5. **How do write to a text file?**
    Checkout https://www.w3schools.com/python/python_file_write.asp

6. **How to import/include a message module?**
    Every message is defined inside a package. So you must first know the name of the package and the message type. For example, /cmd_vel is of type geometry_msgs/Twist. geometry_msgs is the name of the package and Twist is the type of message.
    To import in Python:

    ```
    from <pkg_name>.msg import <msg_type>
    ```

    To include in C++:

    ```
    #include "<pkg_name>/<msg_type>.h"
    ```

7. **rosrun does not run my node**

Python:

a. Make sure your python script is in the scripts folder of your ROS package
b. Make sure you have given executable privileges to your python script:

```
chmod +x <script_name>.py
```

c. Add the python header to your script:

```
#!/usr/bin/env python
```

C++:

a. Make the necessary changes to the CMakeLists.txt file inside your package folder
b. Make sure you ran catkin_make and sourced your bashrc file:

```
catkin_make
source ~/.bashrc
```

VIII. References:

1. ROS Package: http://wiki.ros.org/ROS/Tutorials/CreatingPackage
2. ROS Parameter: http://wiki.ros.org/Parameter%20Server
3. ROS Bag: http://wiki.ros.org/rosbag
4. Creating Messages: http://wiki.ros.org/ROS/Tutorials/CreatingMsgAndSrv
5. Rviz Makers: http://wiki.ros.org/rviz/DisplayTypes/Marker