PS4
Grid Localization

Out: Thursday, 5 October
Due: Thursday, 12 October

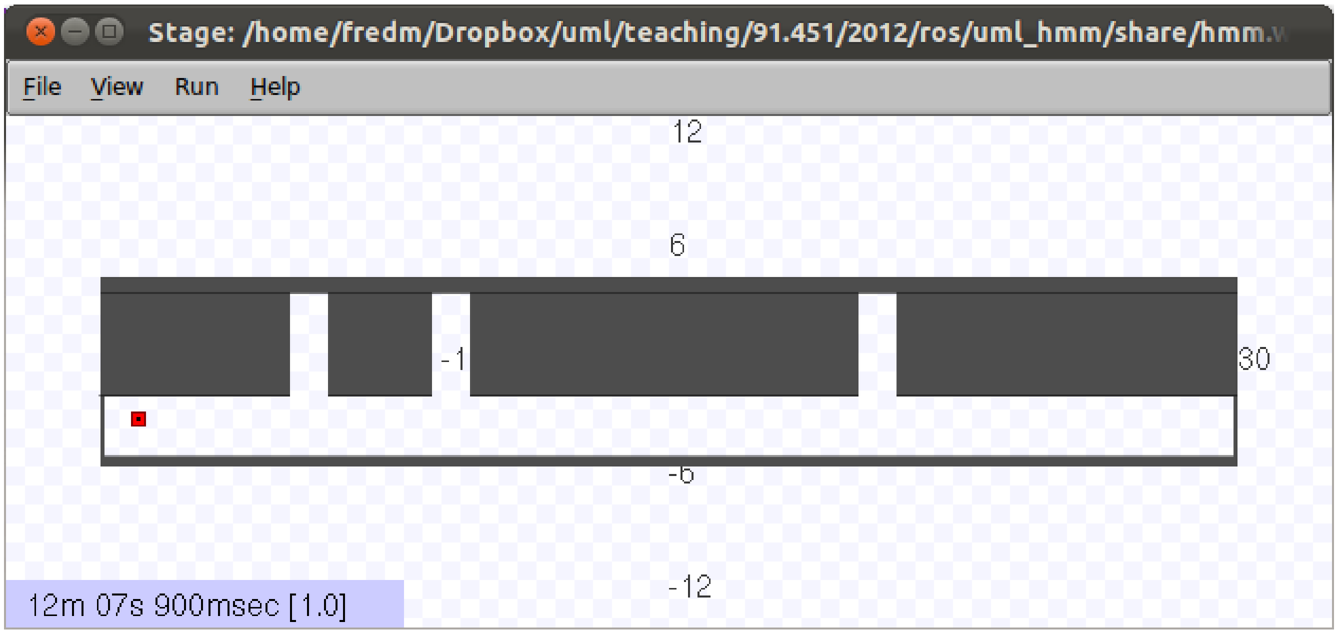
Submission
Copy your assignment from VLabs to the cs.uml.edu server (via the scp command https://www.uml.edu/Sciences/computer-science/CS-Infrastructure/unix-scp-sftp.aspx), then submit material you wish to deliver electronically via:

```
zip -r yourfullname_ps4.zip your_ps4_pkg
submit jkuczyns ps4 yourfullname_ps4.zip
```

If you are not a CS major, email your project to jkuczyns@cs.uml.edu . If you are a CS major and do not have an account on the cs.uml.edu server, then get one :)

Readings
Read Section 8.2, Grid Localization.

Files Needed
https://github.com/DeepBlue14/uml_hmm


Stage: /home/fredm/Dropbox/uml/teaching/91.451/2012/ros/uml_hmm/share/hmm.w
File
View
Run
Help
12
6
-1
30
-6
-12
12m 07s 900msec [1.0]

Introduction
Your robot lives in a hallway with three doors as shown above. Your robot doesn't know where it is. Your job is to move it around and sense walls or doors, and have the robot learn where it is. You will do this by implementing the histogram/grid localization method (Table 8.1 on page 238).

Map
The map, as shown above, represents a hallway that is approximately 60 meters in length. Each pixel in the source image represents 10cm. The doorways are 2m wide.

Motion Model
You can make the robot move by publishing to the ROS topic */robot/cmd_vel*.

You are only to move horizontally, in the positive or negative x direction. Attempts to move vertically or to rotate will be ignored.

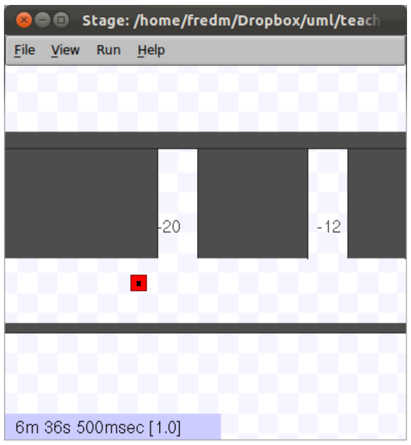
The robot's actual velocity will follow a normal distribution of your commanded velocity. The standard deviation of the normal curve is one-third of the commanded velocity. By the definition of a normal distribution, approximately 68% of the time, the actual velocity will be within one standard deviation of the commanded velocity.

See https://en.wikipedia.org/wiki/Normal_distribution for more.

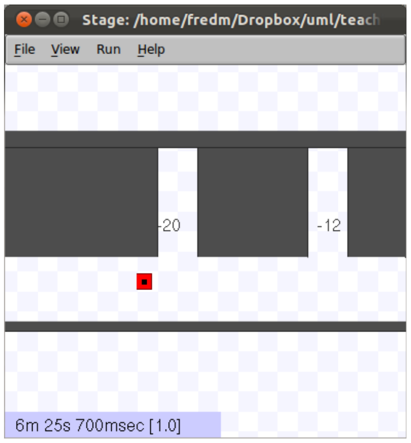
Note: if you attempt to drive beyond the left or right edges of the world, your robot will get stuck, but you won't receive an error message. This will confuse the algorithm, so don't do this.

Sensor Model
Your robot has one sensor. It outputs a string, saying **Wall** or **Door**. If the robot is next to a wall (no door in sight), then 90% of the time the sensor will report Wall. If the robot is directly next to a door, 90% of the time, the sensor will report Door. As the robot crosses over the edge of a door, the probabilities are somewhere between these two endpoints.

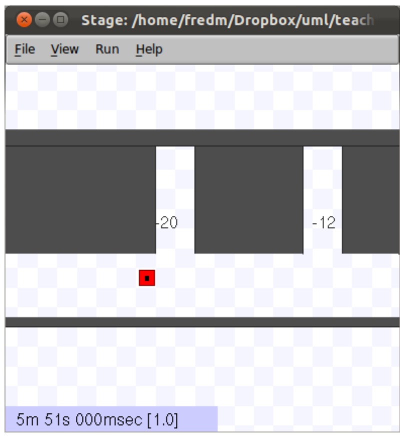
The sensor publishes to the ROS topic */robot/wall_door_sensor*. The images below visually illustrate its performance.



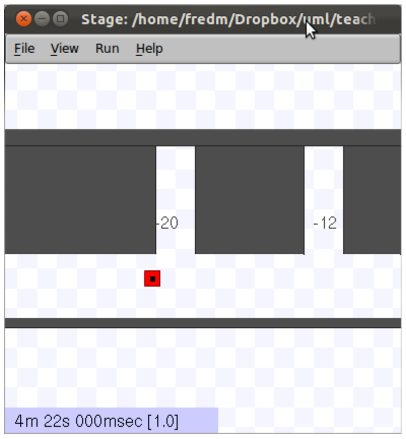
P(Door)=0.1



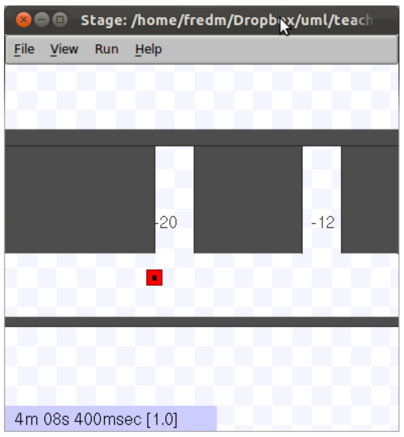
P(Door)=0.12



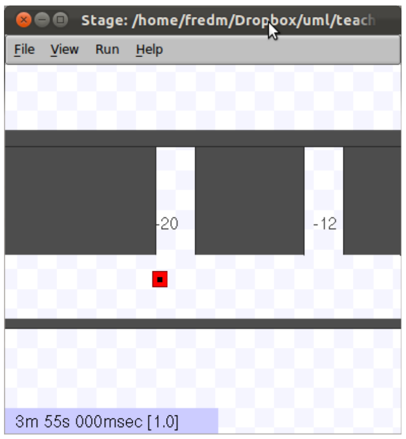
P(Door)=0.21



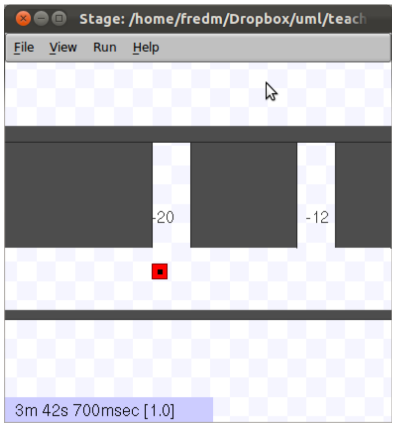
P(Door)=0.37



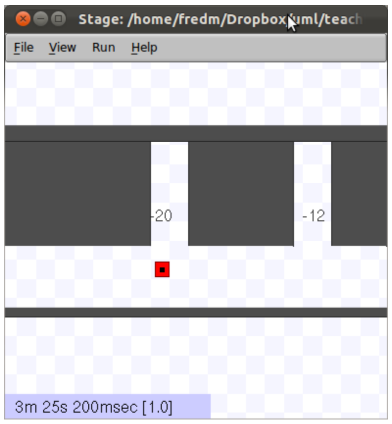
P(Door)=0.47



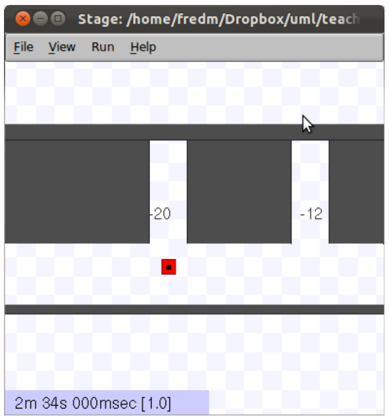
P(Door)=0.65



P(Door)=0.75



P(Door)=0.85



P(Door)=0.9

Your program code will subscribe to the */robot/wall_door_sensor* topic to receive the sensor readings.

You can also display the output of this sensor in a terminal window with the command:

```
$ rostopic echo /robot/wall_door_sensor
```

To view all topics, run

```
$ rostopic list
```

sensor-model-and-map.py

The file sensing-model-and-map.py contains Python code which models the door sensor and the wold map.

The code contains four Python functions:

- *gauss(x, mu, sigma)*: This calculates the value of the normal curve given its center point mu, its standard deviation sigma, and where along the curve you are sampling (x). Note that the height of the curve at its peak varies depending upon the standard deviation, because the area under the curve must sum to one. Therefore, if these is a small standard deviation, and the curve is highly peaked, the maximum value will be greater.
- *door(mu, x)*: This calculates the probability of seeing a doorway, given the center point of the door mu (in meters) and your x position. The result is scaled to a maximum likelihood of 0.8. So if you are standing right in the middle of a door – e.g. door(10, 10) – the output will be 0.8.
- *p_door(x)*: This calculates the probability of seeing a door given your position in the world x. It incorporates the world model by locating doors at 11m, 18.5m, and 41m. It also adds in the error baseline of 0.1, so probabilities will be in the range of 0.1 to 0.9.

  Example output:
  p_door(11) = 0.9 because this is the exact middle of a doorway
  p_door(5) = 0.1 because there is a wall there
  p_door(10) = p_door(12) ~= 0.43 because this is the edge of a doorway.

- *p_wall(x)*: This is 1 – p_door(x).

Before You Write Any Code!!!

Before you write any code, write down your answer and explanations for items 1 through 3 below, under "To Turn In".

What Your Code Should Do

When implemented properly, the grid/histogram algorithm will allow the robot to localize regardless of its initial starting position on the map.

Grid Algorithm Details

You should implement two versions of your code: one in which the robot starts at the left of the world and moves to the right, and one in which the robot starts at the right of the world and moves to the left. The only difference between these two versions should be the x velocity command given to your robot; nothing else should need to change. See the Notes below for information on how to position your robot at the right edge of the world before running your code that moves to the left.

Besides implementing the core algorithm of Table 8.1, you must create some kind of function that outputs your belief histogram each time through the loop. You may print the histogram out as ASCII art to the terminal console, or write your own Python graphics display tool using a ROS graphics dependency (such as rqt).

It is recommended that you use a histogram of 600 bins – i.e., one bin per 10cm of the world (one bin per pixel of the source map).

As discussed on page 241 of *Probabilistic Robotics*, for good performance, it is necessary that the robot's movement update causes probabilities to shift across bins in each time step. Since the default ROS time unit is 100ms (10Hz), it is suggested to use a command velocity of 4 m/s, or approx. 4 bins per time step. It might be necessary to perform a delayed motion update (as described on page 244) for good performance; feel free to experiment.

Remember that you should initialize your histogram with maximum uncertainty as to the robot's initial position (i.e., equal probability in every bin).

Notes/Setup

```
$ cd ~/catkin_ws
$ source devel/setup.bash
$ cd src
$ git clone https://github.com/DeepBlue14/uml_hmm.git
$ cd ..
$ catkin_make
```

Launch the world with the command:

```
$ roslaunch uml_hmm hmm.launch
```

In the terminal that ran the launch command, you will see a continuous display of the output of the wall/door sensor.

You can cause the robot to move around from the terminal with the following command:

```
$ rostopic pub /robot/cmd_vel geometry_msgs/Twist --once [5,0,0] [0,0,0]
```

This will cause the robot to move to the right for one time step. This goes through the probabilistic function to smear the actual velocity. You can also use the "-r hz" option instead, or write a short node to take keyboard input; see http://wiki.ros.org/rostopic#rostopic_pub for more details.
You can use this to initialize your robot to different locations before running your code.

To Turn In

You should zip up **all of the below** when submitting your assignment:

- A ROS package directory containing your solution. Do ***not*** submit a whole ROS workspace.
- A writeup for the questions below.
- A text or markdown file with instructions for how to run the program (so the grader knows how to run your node).

Make sure that all of this is contained in a directory that has your real time as its name.

PS4 Questions

1. If your robot happens to start at the left edge of the world (per the given startup code), and moves to the right, at what point in the world will the robot be when the histogram first reports high confidence in the robot's position? Explain.
2. If your robot happens to start *at the right edge* of the world, and moves to the left, at what point in the world will the robot be when the histogram first reports high confidence in the robot's position? Explain.
3. If your robot happens to start *in the center* of the world, and moves to the right, at what point in the world will the robot be when the histogram first reports high confidence in the robot's position? Explain.
4. Turn in your implementation control program for the grid/histogram localization.
5. Turn in some visual evidence (e.g., screen snaps) showing that your localization code works properly when your robot moves from left to right. This should include a simultaneous view of both the robot-in-its-world (i.e., Stage) and your histogram display. Please include three to five screen snaps, with numerically increasing filenames indicating temporal progress. If you have the software to record a continuous movie of what's on your screen, that would be fine too.
6. Using grid/histogram localization, turn in visual evidence of your code working with the robot moving starting at the left edge of the world (the default world initialization) and moving to the right.
7. Indicate your choice of robot velocity, and how frequently you performed motion model updates (e.g., every time step, or every *n* time steps?).

This assignment was jointly developed by Fred Martin, James Dalphond, and Nat Tuck. Thanks to Eric McCann for updating it for 2015, and to Holly Yanco and James Kuczynski for updating it for 2017.