PS5
Particle Filter Localization

Out: Thursday, 12 October
Due: Friday, 20 October at 11:59pm

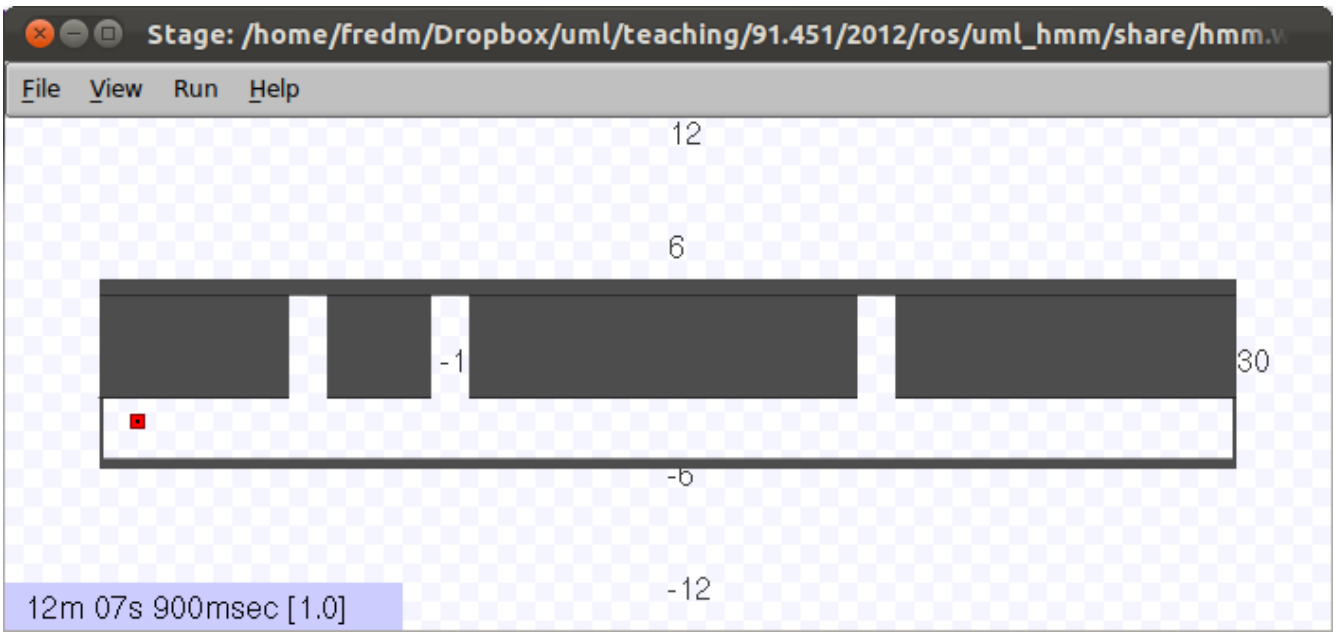
Submission
Copy your assignment from VLabs to the cs.uml.edu server (via the scp command https://www.uml.edu/Sciences/computer-science/CS-Infrastructure/unix-scp-sftp.aspx), then submit material you wish to deliver electronically via:

```
zip -r yourfullname_ps4.zip your_ps5_pkg
submit jkuczyns ps4 yourfullname_ps5.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 8.3, Monte Carlo Localization.

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



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 particle filter/Monte Carlo localization method (Table 8.2 on page 252) from *Probabilistic Robotics*.

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.

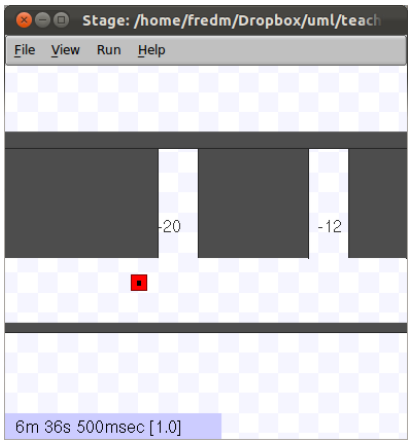
See https://en.wikipedia.org/wiki/Normal_distribution for more.

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.

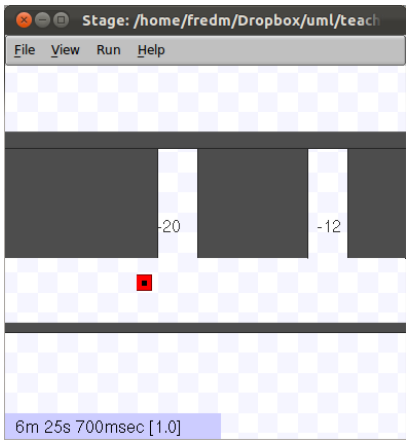
Sensor Model

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.

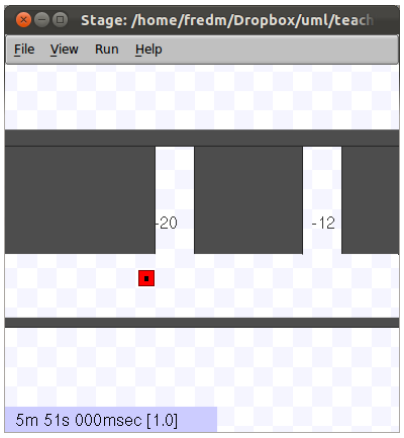
The sensor publishes to the ROS topic */robot/wall_door_sensor*. The images below visually illustrate its performance.



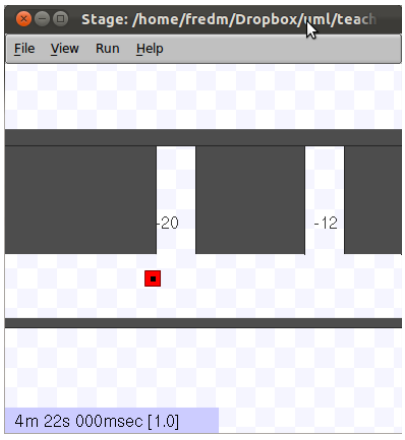
P(Door)=0.1



P(Door)=0.12



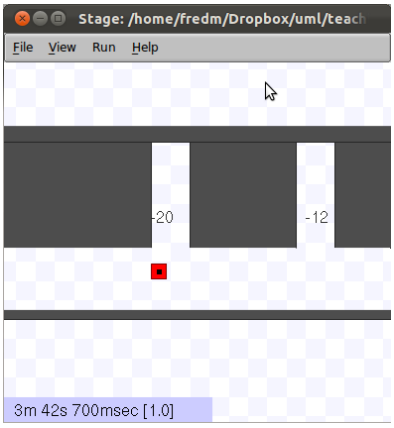
P(Door)=0.21



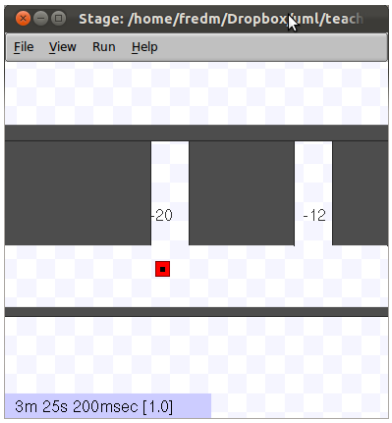
P(Door)=0.37

P(Door)=0.47

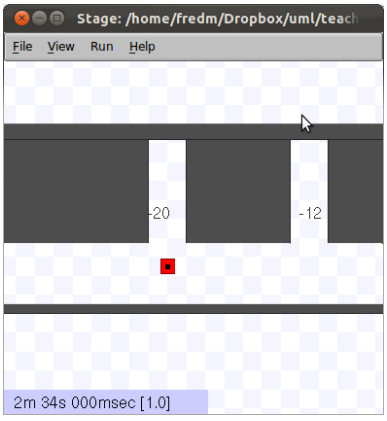
P(Door)=0.65



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 OR sensor_model_and_map.hpp/.cpp

The file sensing_model_and_map.py contains Python code which models the door sensor and the wold map. The file sensor_model_and_map.hpp contains the same functions.

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).

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.

Particle Filter/MCL Algorithm Details

Create the algorithm using the template shown in Table 8.2/page 252 of the text, and using the details provided in lecture, and from the AI class lectures/YouTube videos.

In a similar fashion to the histogram display described in PS4 for the grid algorithm, you must implement some kind of visual display of your particle clustering, which will be shown between each iteration of the particle sampling loop.

You can try difference numbers of particles to determine how many are needed for good performance.

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

1. Code: Turn in your implementation control program for the particle filter/MCL localization, and the code to estimate the robot's position as the algorithm operates.
2. Writeup: (~500 words total)
   - **Include your name and email address so we can send your grade back electronically.**
   - Specify how many particles you used in your MCL implementation, and why this seemed to be a good number.
   - Define what you would mean by work well in terms of the algorithm.
   - How do you determine your confidence on where the robot is?

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