Bayes Filter Lab

Purpose

The purpose of this lab is to give you experiencing creating AI algorithms designed to reason in uncertain environments that can be modeled as Markov processes. In particular, you will implement the discrete Bayes Filter to predict which state the robot is in given (a) knowledge of the state transition function and sensor models and (b) knowledge of the actual actions taken and sensor readings.

You can do this lab in teams of 1-3 people. Only one submission is required for the group. Make sure all group members are specified.

Software

To perform this project, you will add to the supplied Java program. The supplied software includes two programs. The Server and your client program (theRobot). The Server creates a grid world that a robot moves about in. The client program connects to the server and tells the robot what to do. It also receives sonar readings (from the server) to help it reason about its environment. The software is written in Java, and you will be moving the client program to implement the Bayes Filter.

When the server is started, the robot is randomly placed in the world. The position of the robot is shown as a blue circle in the Server's GUI. The world consists of open spaces (white squares), walls (black squares), stair wells (red squares), and a goal (green square). At each time step, the robot can try to move either up, down, left, or right, or try to stay in the same position (if the robot moves into a wall, it remains in the same square). However, the robot's controls are imperfect, as it sometimes moves in a different direction than it intended. Let p_m be the probability that it moves in the direction it intends. Then, with probability $(1-p_m)/4$, it moves in each of the 4 other directions. The probability p_m is specified as a command-line argument when you start up the server.

The robot is also equipped with four sonar, which point up, down, left, and right. At each time step (after moving), the robot takes a single sonar reading in each direction. The sonars sense whether there is a wall directly next to the robot in the specified direction. However, the sonar are also noisy. With probability p_s, a sonar returns the correct reading. With probability $(1-p_s)$, a sonar returns an incorrect reading. The probability p_s is specified as a command-line argument when you start up the server.

To run the software, you must first compile the server and client programs (run `javac *.java` in each folder). Then, to run the server, go to the Server world in a terminal, and type:

```
java BayesWorld [world] [motor_probability] [sensor_probability] [known/unknown]
```

where `world` can be any of the worlds specified in the "Mundo" directory, `[motor_probably]` is a value between 0 and 1 specifying p_m, `[sensor_probability]` is a value between 0 and 1 specifying p_s, and "known" specifies that the robot's initial position is given to the robot at the start of the simulation, and "unknown" is specified to say that the robot's initial position is not given to the robot at the start of the simulation. For example:

```
java BayesWorld mundo_maze.txt 0.9 0.8 unknown
```

starts the server in the world mundo_maze.txt, with $p_m = 0.9$, $p_s = 0.8$, and the robot's initial position is unknown. Several worlds are already provided (you can create your own if you would like).

Note: in this lab you should always set the last parameter to "unknown."

Once the server is running, you can connect the robot (client) to it. In a separate terminal, go to the "Robot" folder and type:

```
java theRobot [manual/automatic] [decisionDelay]
```

where "manual" specifies that the user (you) will specify the robot's actions, "automatic" specifies that the robot will control it's own actions, and `[decisionDelay]` is a time in milliseconds used to slow down the robot's movements when it chooses automatically (so you can see it move). In manual mode, you press keys to have the robot move when the client GUI window is active. 'i' is up, ',' is down, 'j' is left, 'l' is right, and 'k' is stay. Note that the client GUI must be the active window in order for the key commands to work.

Note: the "automatic" mode is not needed for this lab. Likewise, for this lab you can just specify 0 for "decisionDelay" for this lab. Thus, you can simply type:

```
java theRobot manual 0
```

to run the program.

What You Should Do

Your job in this lab is to modify theRobot.java so that it identifies the robot's location in the world as it moves about it using a discrete Bayes Filter. You will use knowledge about the robot's transition model (how it moves in the world given the actions selected) and its sensor model (how the robot's sensors return information) to figure out where it is in the world. To do this, you will implement a discrete Bayes Filter. A single update for the discrete Bayes Filter is given below:

Algorithm Discrete_Bayes_Filter(Bel(X_{t-1}), a_t, z_t)
  for all x_t in X_t do
    Bel'(x_t) = Σ_{xt-1} p(x_t | a_t, x_{t-1}) Bel(x_{t-1})
    Bel(x_t) = η p(z_t | x_t) Bel'(x_t)
  endfor
  return Bel(X_t)

Here, $Bel(X_{t-1})$ is the robot's beliefs about where it is in the world (over it's possible set of states X_{t-1}), a_t is the action taken at time t, and z_t is the sonar reading taken after moving in time t. Also, the variable η, is a normalization factor, as Bel(x_t) will not define a legal probability distribution.

You should implement the Bayes Filter from the method `updateProbabilities()` in theRobot.java.

To test your code, you'll want to experiment with how well you can determine where the robot is in the world (without viewing the server GUI). Can you get the robot to the goal? Try it out in different worlds, with different values of p_m and p_s.

Important notes: As part of implementing this algorithm, you will need to create a function that specifies the transition model and the sensor model from the variables `probMove` and `sensorAccuracy`. You'll need to figure out how to do this. Note that in the grid world, the position (0, 0) is in the top left corner of the world.

Also, you'll find it useful to modify the probabilities given information about whether or not you reached the goal or fell in the stairwell. Your Bayes Filter algorithm will sometimes put some probability that the robot is in the goal state or in the stairwell — however, if the game isn't over, you can known that this

didn't happen. Thus, you can zero out those probabilities once you have verified that the game has not ended.

What You Should Turn In

Submit the following:

- Several videos showing your robot localizing (using the code you wrote) as you move it about the world. Ideally, the video should simultaneously show both the server GUI which shows the robot's actual position and the GUI showing the output of the Bayes Filter (client GUI). Videos should be provided for multiple worlds and multiple values of probMove and sensorAccuracy. Make sure you mark which parameters were used for each video (either on the video or in .doc or .pdf file). If it looks like your Bayes Filter works, you'll get full credit. **Note: If you don't have much experience generating videos, at least one person has recommended that you can easily use Screencastomatic to do this (this isn't a plug for this software, but you may find it useful).*