CS 470 Programming Assignment: Bayes Filter

The purpose of this lab is to give you experiencing implementing an algorithm discussed in class which allows our agents to reason in uncertain environments. 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.

---

1 Software

To perform this project, you will add to the supplied Java program. The supplied software includes two programs.

- The Server (in the `Server` folder) - The Server creates a grid world that a robot moves about in. This grid world serves as the environment for your agent. It will receive the actions from the agent (your robot), determine how the environment changes as a result of those actions (ie where did the robot actually go, which is determined probabilistically based on the map and the robot's action. For example, the robot can't drive into a wall, even if the action tried to. Also, sometimes actions don't work out the way they should). Then the environment will send observations (sensor readings) to your robot, which are determined from the sensor model, given the true state of the environment (robot location and map). You shouldn't have to modify this code at all, but you can look at it to see how everything works.
- The Client (in the `Robot` folder) - This is the client, which is your agent/robot, which will receive observations from the Server (environment), and then send actions to the environment. This is where your new code will go. There is a fair amount of starter code provided for you. Your task is to utilize the sensor readings from the environment to update the beliefs of where the robot is on the map. You will do this in this assignment using a Discrete Bayes Filter.

All of the provided software is written in Java, and you will be modifying the client program to implement the Bayes Filter.

1.1 Server Operation

When the server is started, the robot is randomly placed in the world. The true 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).

1.2 Robot Transition Model

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 $\frac{(1-p_m)}{4}$, 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.

1.3 Robot Sonar Sensor Model

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 also specified as a command-line argument when you start up the server.

1.4 Running the Code

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
- `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,
- `[decisionDelay]` is a time in milliseconds used to slow down the robot's movements when it chooses automatically (so you can see it move).

Running the robot program will open up another GUI, which displays the beliefs of the robot on the map. This allows you to see what your filter is doing, and is very helpful for debugging, as well as verifying that things are working properly.

1.5 Driving the Robot

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, you can just specify 0 for `decisionDelay` for this lab. Thus, you can simply type:

```
java theRobot manual 0
```

to run the program.

2 The Assignment

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, which we have covered in class. Psuedocode for a single update for the discrete Bayes Filter is given in Algorithm 1 below:

Algorithm 1: Discrete Bayes Filter ($Bel(X_{t-1}), a_t, z_t$))

Result: Updated beliefs $Bel(X_t)$ given action a_t and sensor reading z_t
for *all* x_t *in* X_t **do**
  $Bel'(x_t) = \sum_{x_{t-1}} P(x_t|a_t, x_{t-1})Bel(x_{t-1})$;
  $Bel(x_t) = \eta P(z_t|x_t)Bel'(x_t)$;
end
return $Bel(X_t)$

Here, $Bel(X_{t-1})$ is the robot's beliefs about where it is in the world (a probability distribution 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 initially define a legal probability distribution, after only multiplying by the probabilities from the sensor model.

2.1 Specifics of Your Task

You should implement the Bayes Filter from the method `updateProbabilities()` in `theRobot.java`. To accomplish this you are welcome (and even encouraged) to implement any helper functions that you need to complete the task and keep your code organized.

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.
- You might 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. (make sure you renormalize after doing this)
- For debugging, it might be helpful to run with $p_s = 1$ and/or with $p_m = 1$. This will enable you to make sure that the basics of your code are functioning correctly. Only after this is working should you move on to testing with p_m and p_s values less than 1.

2.2 Testing your Filter

To test your code, you should experiment with how well you can determine where the robot is in the world (without viewing the server GUI). See if your beliefs are accurate enough to get the robot to the goal You should make sure your code works in different worlds, and with different values of p_m and p_s.

3 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 a `.pdf` file). If it looks like your Bayes Filter works, you'll get full credit.
- A `.pdf` including the code you modified/wrote for this assignment and a paragraph giving advice to the younger you (before you started the assignment) of what you think would have helped you make progress on the lab faster. Or if everything was clear, you can just say that.