Applied Artificial Intelligence
EDA132
2016-02-10
Johanna Petersson (adi10jpe@student.lu.se)

Assignment 2 – Probabilistic Reasoning

By: Johanna Petersson (adi10jpe@student.lu.se)

Introduction

In this the second assignment, a two part solution was required to estimates a robot's position on some grid with the help from a noisy sensor. The two parts consists of a simulation of the robot and the sensor along with the algorithm that estimates the robots position.

The program

As said in the assignment this problem required a two-part solution, done in the same program but nothing to do with each other. Each part is explained under the separate headings.

Robot and sensor

The robot and sensor is implemented in the same class, `Robot.java`, in my program.

Robot

The robot is implemented in such a way that the every time a call is made to the method `move()` the robot moves one step in the grid. The grid size of my program is 8x8, but the program can be modified to be any size with one simple change.
The robot's moves are well defined according to a set of rules in the program. The robot has a heading and moves one step in that heading with the probability of 0.7. The probability that the robot moves one step in a new heading is 0.3. Should the robot be facing the end of the grid on the current heading the robot always chooses a new heading. A new heading is always chosen at uniform random, so is the starting position of the robot.
As an implementation detail I have chosen that the robot only move horizontally and vertically, never diagonally over the grid, giving the options of four headings.

Sensor

The premise of the assignment was that the sensor is noisy and must be implemented as such, thus never giving the exact position of the robot as output. To get the output of the sensor, the method `sensorOutput()` is called in the Robot class is called. The method returns a point that represent where the sensor thinks the robot is positioned.
The rules that define the sensor output is as follows:

- With the probability of 0.1 the real location of the robot is returned
- With the probability of 0.4 any of the eight neighboring positions is returned
- With the probability of 0.4 any of the sixteen positions surrounding the eight neighbors
- With the probability of 0.1 that *nothing* is returned

If neighboring positions to the robot is outside the grid and such a position the sensor returns *nothing*. In reality this means two things. First, that the probability of *nothing* being returned is higher if the robot is close to the wall than if it is in the sensor. Secondly, *nothing* can be thought of in two different ways, an unclear reading/faulty sensor or a position outside of the grid.

Estimation algorithm

This is the second part of the assignment, estimating the position of the robot. For this purpose a forward algorithm along with an underlying Markov chain is implemented in the Algorithm.java class.

Model

As a base for the model we define what a state is (implemented in State.java file and class), and it's simply a position and a direction/heading. A state defines one possibility of where the robot is located. Together all possible states total up to $4\times8\times8 = 132$ number of states.
The model is implemented as a matrix in my program. The states and their respective probability is represented in two vectors, stateMapping containing all the 132 states and stateProbability the probability of each state, with the index of the vectors, are being used to map between the two. A third matrix represents the transition model as a two dimensional vector that is 132×132 in size, where each entry [i][j] is the probability of transitioning between the two states i to j in the stateMapping vector.
I choose to represent the observation matrix as a vector as well with the same size as the stateProbability vector since those are the observable states, but it is supposed to be represented as the observation in diagonal.

Forward algorithm

Depending on the implementation the forward algorithm uses a hidden markov model to calculate filtering or prediction. In this assignment it is used as a filtering algorithm to estimate the probable state of the robot, given the evidence. One of the major things with the forward algorithm is that it takes $O(1)$ time and space to calculate, even though it is dependent on the amount of states available. The pseudo code below explains how I have implemented the forward algorithm.

```
State Probability Vector S
Transition model T
Observation vector O
for row = 0 ... S.size do
   for col = 0 … S.size do
      $S(row)_{next} + \alpha O(row) * T(col)(row) * S(col)_{prev}$
   end for
end for
```

A few points should be made as explanation.

- $T(col)(row)$ is the transpose of the regular $T(col)(row)$ and it's the transition probability that it arrived to the current row from the given columns.
- $O(row)$ is the observation was made, ergo the probability that it got that certain sensor reading if it is in the state row.
- $S(col)$ is the current probability that the robot came from this state.

Result and discussion

One of the decisions I made was that the probability of getting *nothing* from the sensor is higher if the robot is closer to the edge of the grid. This was done after to start with forcing the sensor to output a neighboring position, even if it was outside the grid. This in turn meant that I had to make sure the output from the sensor was a position inside the grid, giving a higher probability to those positions in the grid, than those outside. This caused some problems when I were to estimate the robot's position as any one of the positions that had higher probability than normal, given them higher probability than the actual position. In turn this made it hard to estimate the actual position of the robot in the grid. The estimations when this "faulty" implementation gave a correct position about $35 - 40\%$ of the time.
With the chosen implementation I archived a final probability of 41.9% after running 100 iterations of a 1000-step model.

A short introduction to the source code

This is a short short introduction to the nine java files the program contains as an orientation to reading the code.

Direction.java

This file contains the Direction enum, with the four possible directions for the robot as well as a numerical representation. Used for clarity in the code instead of using integers.

Point.java

A simple class representing a position in any xy grid. A trivial but useful class.

State.java

This file represents a state in the program, holding a point and a direction. Again a trivial class that makes life easy. It can determine if another state is a neighbor to the objects state, as well as if it is next to the end of the grid and how many neighbors it ought to have.

Robot.java

This is the class that represents the robot and it's sensor. It can move the robot to a new position, knows what the heading of the robot is, and can return the output of the sensor.

Algorithm.java

This file contains the implementation of the forward algorithm and that estimates the robot's position. A lot of the rows of code in this class is taken up with creating the different vectors that are needed. It only contains one public method besides the constructor and that is getEstimatedLocation, that takes a point and returns the estimated point.

Gui.java, MatrixPanel.java and MsgPanel.java

These three files are what make up the GUI, with Gui.java in the center. They are simply split up to make it a bit more manageable.

MatrixPanel.java shows the grid and where the robot is, what the sensor says it is and where the estimation say the robot is located, all color coded to make it bit easier to see. MsgPanel.java contains the panels at the bottom of the GUI showing how the robot moves and the estimations done.

Program.java

This is where the main method of the program lives. Along with some parameters for the program such as the grid size and how long the delay for each robot move should be. The delay is there simply to make it usable when viewing the GUI. The loop that keeps the program alive and calculate the ration of correct estimates are done here for convenience.

Getting the program

The code and an executable .jar file can be found at https://github.com/JohannaMoose/eda132 in the folder Assignment 2 – Probabilistic Reasoning. To download the code and program onto one's computer go to the URL above and click the button "Download ZIP" that can be found just above the file list. This will download a zip file to the computer with all the files; go to where the file downloaded and into the folder Assignment 2 – Probabilistic Reasoning to see the files pertaining to this assignment.

The file Robot.jar is runnable from the terminal on Linux/Mac computers with the command `java -jar Robot.jar` (tested on Mac) if the terminal is pointed to the folder. This will start the program, and instructions on how the game works will follow in the terminal window.

Of cause, the code in the folder should be buildable as well and should run the program without a hiccup, provided that Java SDK8 is installed on the computer. Nothing else is tested.