HMM: Robot localization

Applied Artificial Intelligence

Daniel Eliassin and Eva Gala

May 10, 2015

1 Introduction

The goal to this work is to develop a robot localization based on a hidden Markov model. It will also be needed to build the robot that the agent will try to track.
We have chosen to implement this in Java, and following the course *Applied Artificial Intelligence* and the book Artificial Intelligence: A Modern Approach by Stuart Russell and Peter Norvig.
Our work has two main parts: to generate the robot that will move according to the probabilities that had been given, and the algorithm that will try to track the moves of the robot and approximate its position. This last one is the one we are going to call agent for now on.

2 Basic Program

First we needed to lay down the basic design of the program. Some of the important parts are described briefly below:

- *MainWindow:* this class shows the result. Represents a grill with the real position, and the estimation. We'll explain later how does it represents.
- *Robot:* this class contains the robot that we'll track. It generates the aleatory moves, according to some established probabilities, that the robot will be doing while the agent tries to track it. It also has the sensor with the respective error.
- *HmmAgent:* the fundamental part of this assignment is the approximation of the location based on a hidden Markov mode. This is what is done in this class. We generate the most probable states, and also keep the probabilities of each state.
- *The math package:* we used this as auxiliary classes. We need them to managed the matrices and its operations. We borrowed this code from the code repo for the course book: Artificial Intelligence: A Modern Approch.

- *Direction, State, Point* and *Room*: another auxiliary classes useful to manipulate the positions and directions.

3 HmmAgent

For updating the agent, first we need to get he new evidence from the *Robot*. Then we need the sensor model matrix (O) and the transition matrix (T).

3.1 Transition matrix (T)

The transition matrix is used during the update, even though it is generated at the constructor of the function. This is possible since it won't change with the time. It represents the probabilities of the robot to go to one state j at time t assuming it was at state i at time $t-1$.

$$T[i][j] = P(X_t = j | X_{t-1} = i)$$

For calculating this, first we check the directions that lead the robot against a wall ($notDirec$). If the current direction (d_i) is a wall, we set the probabilities as $1/(4-card(posDirec))$ for all the states $\{(x_j, y_j, d) : \ d_j \notin notDirec \ and \ (x_j, y_j) = (x_{d_j}, y_{d_j})\}$.
Being (x_{d_j}, y_{d_j}) the coordinates resulting at moving from the position at the state i in the direction d_j. We call the set that contains this coordinates surroundings (S) and $S_j$the specific ones according to direction d_j.
If d_i is not a wall, then for (x_{d_i}, y_{d_i}, d_i) we set the possibilities to 0.7. And for the rest of the states with d_j not at $notDirec$ with coordinates belonging to S_j, we set $0.3/(4 - card(notDirec) - 1)$.

$$T[i][j] = \begin{cases} 0 & if \ d_j \in notDirec \ or \ (x_j, y_j) \notin S_j \\ 1/(4 - card(posDirec)) & else \ if \ d_i \in notDirec \\ 0.7 & else \ if \ d_j = d_i \\ 0.3/(4 - card(notDirec) - 1) & else \end{cases}$$

3.2 Sensor Model (O)

For generating the sensor model we used the function *sensorModel(Point evidence)*, that gets a point as evidence, and generates a diagonal matrix as big as number of possible states.

$$O[i][j] = \begin{cases} P(e_t | X_t = i) & i = j \\ 0 & i \neq j \end{cases}$$

Being X_t the state, and $e_t$the evidence that we got.
If the evidence is null, we put the same probability to all states. In this case it doesn't matter because at the end we'll normalized the matrix. Otherwise, for each state i, we'll calculate $d_i := \{max\{|x_i - x_e|, |y_i - y_e|\}\}$ with (x_i, y_i)

the position at i and (x_e, y_e) the evidence that we've got. And we assign it according to:

$$O[i][j] = \begin{cases} 0.1/4 & if\ d_i = 0 \\ 0.05/4 & else\ if\ d_i = 1 \\ 0.025/4 & else\ if\ d_i = 2 \\ 0 & else \end{cases}$$

Then we normalize O.

We keep the matrix $F \in \mathcal{M}_{N\times 1}$with N the number of states.
Then, using that $F_{t+1} = O \times T^t \times F_t$, we get the probability of each state. Then, we normalize, and we select the states that have the more possibilities.

4 Robot

The *Robot* has three functions: to update the direction, update the position and update the sensor.

4.1 Direction

To update the direction, first we check that the current one doesn't lead the robot into a wall. In that case, we select between the new directions with the same probabilities. Otherwise, we keep in the same direction with a 70% and we change with 30%. Is very similar to calculate the probabilities at the transition matrix, but we don't consider the position.

4.2 Position

After updating the direction, we update the new position according to that new direction.

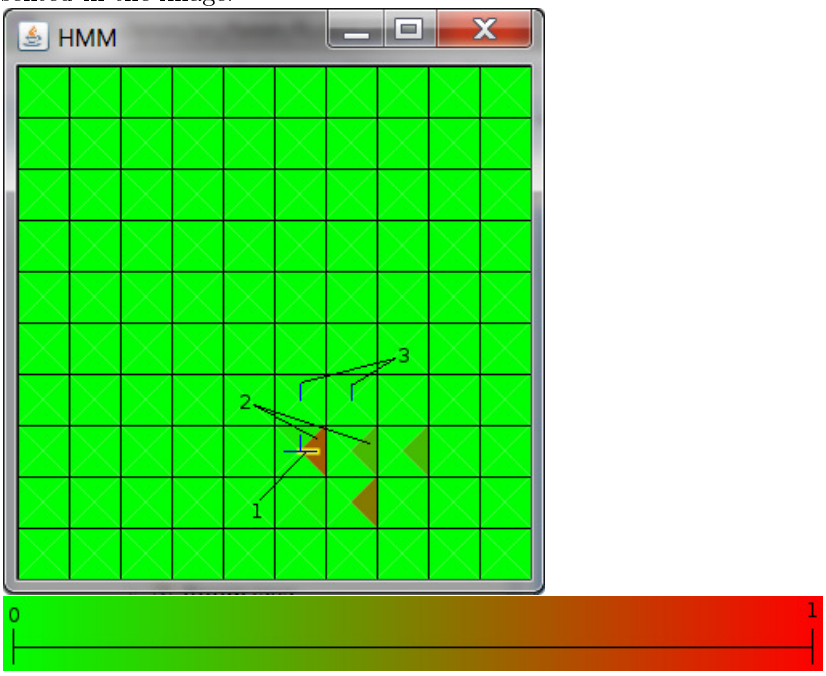
4.3 Sensor

After updating the new position, we update the information the sensor will give. With 10% it will return the real position, with 40%, one of the eight surrounding squares, with also 40%, one of the sixteen other subsurrounding ones and with 10%, noting.

$$P(e|i) = \begin{cases} 0.1 & if\ e = i \\ 0.05 & else\ if\ d_i = 1 \\ 0.025 & else\ if\ d_i = 2 \\ 0 & else \end{cases}$$

5 MainWindow

The MainWindow represents the room in which the robot is in a grid. There we represent the current position as a yellow line looking in the direction [1]. We also represent each triangle of each square that represents the direction with a color from small probabilities, to big probabilities [2]. And, at last, we also represent the states with higher probability for the *HmmAgent* with a blue lines [3]. The triangles representing all the states are colored according to the scale presented in the image.



6 Running the program

To run the program start a terminal window and type the following commands to navigate to and start the program:

```
cd ~ada10del/ai/Hmm/
java -jar hmm.jar
```

References

[1] Russell, Stuart and Norvig, Peter. Artificial Intelligence: A Modern Approach, 2010.