LUND UNIVERSITY

APPLIED ARTIFICIAL INTELLIGENCE

PROJECT 2
ROBOT LOCALISATION WITH HMM-BASED FORWARD-FILTERING

Erwan BLAIN
Abdeljalil MOUSSA

22nd February 2019

Introduction

In this project, we aim at solving a robot localization problem with HMM-based forward-filtering.

We consider a moving robot, whose location is not known and the only evidence available is a direct but vague sensor that gives an approximation of the robot's position. We need to compute some predictions using this sensor and compare it to the ground truth: the actual position. Every loop, our Hidden Markov Model (HMM) computes these predictions and we implement a model that can be as accurate as possible.

This model will be implemented in Python, using different classes that represent the core components of our problem: the robot (with its movement behaviour), the grid (the space where the robot moves), the HMM (which computes all the predictions), and the sensor (which provides inaccurate measures).

1 Method

First, we implemented the sensor model in the `robot` class according to the lecture's sensor model. The sensor reports the location of the robot as pair of coordinates. Hence, it detects current coordinates by returning tuples `(x, y)`. The sensor can report 4 different types of readings : p, p_s, p_{s2} and *nothing*. These readings denote respectively : the current location, any of the 8 surroundings of the true location (provided it is within bounds), any of the 16 surrounding fields (provided the same condition), and *nothing*, that is to say the sensor could not read any value.

Then, the HMM model we implemented predicts the current position of the robot, based on its past locations, and the measures made by the sensor, albeit inaccurate. In the Hidden Markov Model, a possible state is given by the triplet $(x, y, direction)$. These triplets are encoded in a transition matrix, expressed as follows:

$$T_{ij} = P(X_t = j | X_{t-1} = i), (i, j) \in [\![1, 4wh]\!]^2$$

where h and w are respectively the height and the width of the grid, and the state i is related to the triplet (x, y, d) through the following bijection:

$$i = 4 \cdot h \cdot x + 4 \cdot y + d$$

For a 2×2 grid, it gives:

$$T = \begin{pmatrix}
0 & 0 & 0 & 0 & 0.7 & 0 & 0 & 0 & 0 & 0.3 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0.3 & 0 & 0 & 0 & 0 & 0.7 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0 \\
0 & 0 & 0.3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.7 & 0 & 0 \\
0 & 0 & 0.7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.3 & 0 & 0 \\
0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0 \\
0 & 0 & 0 & 0.3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.7 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0.7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.3 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.3 & 0 & 0 & 0.7 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.7 & 0 & 0 & 0.3 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

Moreover, one assumption that we made in the HMM Model, is the `none_matrix` method. In fact, when nothing can be observed or *sensed*, we just create a matrix (the same size as the transition matrix) and we fill each cell by using an average of the transition function on the current position and number of possible adjacent cells (for 8 and 16 surroundings in total). In this way, the closer the cell to an edge, the higher its probability, because the *nothing* measure is more likely to be observed when the robot is near a wall.

On the other hand, the `create_sensor_matrix` method is computed by assigning probabilities to the *sensed* coordinates and *sensed* adjacent cells.

2 Results

In the following we consider a 8×8 grid, in which we performed our tests.

The robot's path can be analyzed using an accuracy measure. This corresponds to the number of times the position was correctly predicted, out of the total number of moves. With our implementation, we get an accuracy varying between 0.25 and 0.5 after 100 moves, and around 0.4 after 1000 moves.

However, this does not precisely account for the quality of the estimation. Considering the average distance between the predicted position and the actual position is therefore valuable. Our estimations prove to be on average between 1.4 and 2 units of distance away from the real position of the robot, using Manhattan distance, with 100 moves. With 1000 moves, the distance revolves around 1.6.

Example of a final output after 100 moves:

```
Robot position:  (2, 4)
Sensor thinks:  (1, 5)
Robot predicts it is in:  (2, 4)  with probability:  0.34445817723530436
Manhattan distance:  0
Average Manhattan distance 1.61
Robot has an accuracy of: 0.32 during this time.
```

3 Discussion

The vague definition presented in the lecture is equivalent to the locations p, p_s, and p_{s2} (explained in the method) and are used to report the true location, the 8 and 16 surroundings of the robot. However, the *nothing* reading corresponds to a step forward without update, it happens more often when the robot is close to wall or a corner.

Regarding the model accuracy, it seems to be performing quite well. The Manhattan distance is smaller than 2, which means that the model manages to roughly follow the robot, but fails to determine its direction when moving. Indeed, the robot directions are random, and it is reasonable to have errors in determining the next square the robot will move to.

4 Implementation

The program can be tested by running the file at the following path:

```
/h/d1/b/er2283bl-s/Documents/LU/EDAF70/Assignment 2/main.py
```

The program runs in Python 3, and requires two arguments: `--height` and `--width`.

Example of use:

```
python3 main.py --height 8 --width 8
```

It simulates 100 moves of a robot, placed randomly at the beginning, with no assumption on its initial position. Hence, the probability of all possible initial states are equal.

The program displays for each move:

- The robot actual position
- The position retrieved by the sensor
- The position predicted by our model, and the corresponding probability
- The Manhattan distance between the actual position and the predicted one
- The average distance so far
- The accuracy of the model so far

The program is divided in four files, managing different parts of the problem. The `robot.py` file handles the robot and the sensor, the `hmm.py` file holds the model and its implementation, the `grid.py` file implements the grid and movement mecanisms, and the `main.py` file runs the experiments and displays the results.

5 Sources

For this project, all the necessary information has been collected from these sources:

- Stuart J. RUSSELL, Peter NORVIG. *Artificial Intelligence: A Modern Approach, Third Edition.* Section 15.3.1
- Stuart J. RUSSELL, Peter NORVIG. *Artificial Intelligence: A Modern Approach, Third Edition.* Section 15.9
- GitHub project by AlexTLarsson (`https://github.com/AxelTLarsson/robot-localisation`)