INF 552 HW7

Teammates:
He Chang 5670527576
Ziqiao Gao 2157371827
Fanlin Qin 5317973858

Part 1.

Data Structure：

Before performing the HMM Viterbi algorithm, we need to collect a set of matrix information from the hmm-data.txt file and calculate a set of probabilities. For storing the data, we basically use Python arrays and dictionaries.

Function load_file returns:

Grid: 10 x 10 matrix that stores {1,0} values to show if a cell is valid to place the robot on.
Tower: 4 x 2 matrix that store the positions of 4 towers
Noises: 11 x 4 matrix that store values of noises from 4 towers in each step

Function getEmission returns:

Emission_matix: 4 x 87 x 166 matrix.

For each tower with different noises, we will need to calculate the emission probability of emitting to each of the 166 observations with respect to different current states. The steps work in the following logic:

For each of the 4 towers:
Step1: for each state, calculate its distance to the current tower, denoted as **d**, and get a range (0.7d,1.3d). The probability of reaching any point in this range is 1/ ((1.3d-0.7d) *10) using 0.1 as the counting unit.
Step2: for each observation, if its value is between 0.7d and 1.3d, it will have probability 1/ ((1.3d-0.7d) *10), else the probability is 0.

Observations: 1 x 166 array that stores a list of observations that can be made. We get this list by first calculating the max distance from towers on the grid, denoted as **d**, which is the diagonal of the grid. Because of noises, the max distance extends to **1.3*d**. Then we use 0.1 as the unit to intercept 166 points from the distance of **1.3*d**.

Function read_grids returns:

States: 1 x 87 array that stores a list of ids for valid cells which contains value 1 in the grid.
Valid_corrdinates: a dictionary that stores value pairs {state id, actual coordinates in grid}.
Initial_prob: 1 x 87 array that stores a list of probabilities for each state id, and the probability is derived by 1/87. It is just used for the first step.
Transit_matrix:
87 x 87 matrix that stores the transit probability from one cell to another. Transit_matix[X][Y] denotes the probability that a robot at cell with state id X goes to the cell with state id Y in the next step. The probability is calculated in the following order: 1. Check if X and Y and next to each other, because if not it is not possible to transit from X to Y in one step. 2. Examine how many valid direction moves (up,down,right,left) can be made from X denoted as **count**, and the probability will be **1/count** for the transition from X to Y in one step.

HMM:
1. For each step, get the stacked emission probabilities from the 4 emission matrices affected by the 4 towers.
2. Calculate the states probability from previous states, select the maximum probability of each current state and record the last state that leads to this probability.
3. Repeat 11 times, except that at the first iteration, the previous states of probabilities are the initial probabilities.

Optimization:

- In order to ensure the correctness of all input matrices to HMM, we separate the procedures into 3 steps by having 3 functions. Each function is in charge of generating a subset of the whole matrices we need.
- We also put the predicted trajectory of the robot into a graph to visually confirm the results are not violating any rules.

Challenge:

- We spent a long time trying to map the HMM and Viterbi algorithm knowledge to solving this problem because it is hard to perceive relationships of the grid, towers and noises.
- Constructing the transit matrix and emission matrix confused us many times for they involved a lot of dimensions of data and we often have to stop and refresh our minds to avoid mistakes.
- Designing the Viterbi algorithm also consumed a lot of our effort, because we debugged the matrix multiplications for a long time.

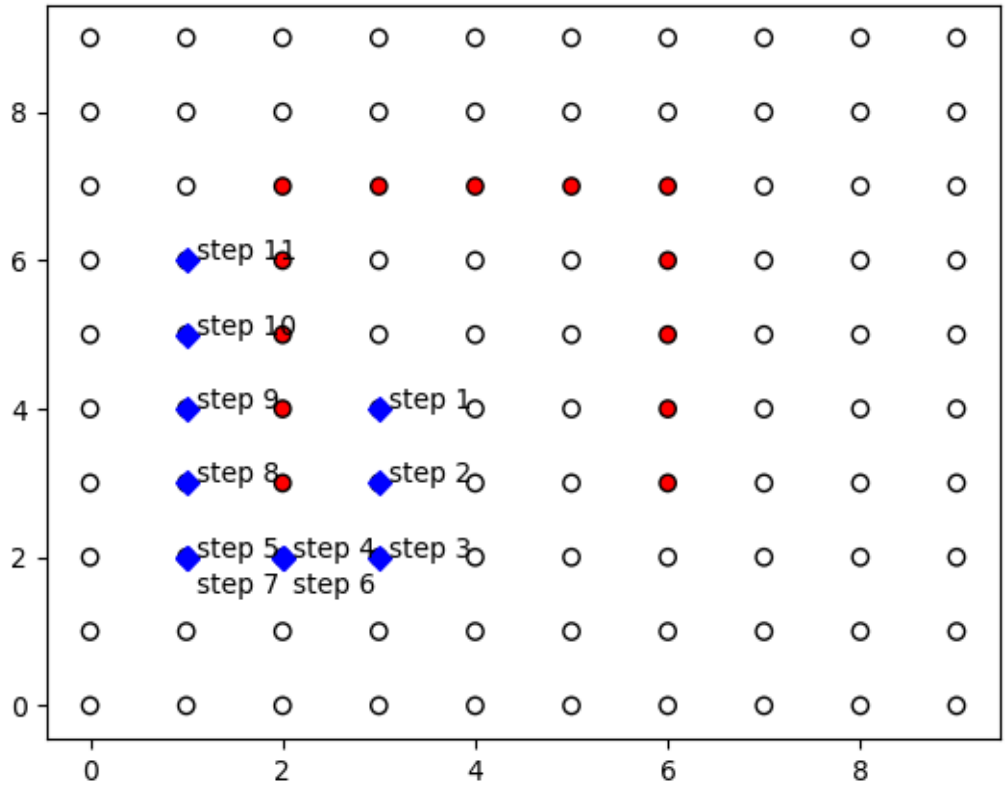
- Because the data file organizes the data points from top left corner, which is different from how plt library works, we have to rotate the points so that the plotted graph matches how the points look in the data file.

Result:

The predicted coordinates of the most likely trajectory of the robot for 11 time-steps:

[[5, 3], [6, 3], [7, 3], [7, 2], [7, 1], [7, 2], [7, 1], [6, 1], [5, 1], [4, 1], [3, 1]]

White dots represent free cells.
Red dots represent obstacles.
Blue annotated rhombuses represent states that the robot transits by.



Part 2.

Library tried and used:

- hmmlearn.MultinomialHMM()
- hidden_markov.hmm()
- hmms.DtHMM()

Summary:

In general, the libraries we tried use similar parameters as the following:

- initial probability matrix with shape (n_components,)
- transition probability matrix with shape (n_components, n_components)
- emission probability matrix with shape (n_components, n_features)

N_components refers to the number of all possible states.
N_featrues refers to the number of all possible observations.

Unfortunately, for this trajectory prediction problem we find it is hard to fit our ways of generating matrices into the parameter requirements. The HMM libraries would like to just take one emission matrix that applies to all possible states. For hw7, we actually have 4 possible emission tables. Our emission probability matrix is actually dynamic, where in each step the emission probabilities are affected by a different set of tower noises. In each of the 11 steps, we have to recalculate the emission probabilities based on the new tower noises by collecting data from our 4 * 87 * 166 large emission probability matrices set. We are not able to provide a static 87 * 166 emission matrix as a parameter for the library modules.

By researching the third-party libraries, we found one interesting fact. Our current problem mainly targets discrete emissions, but the third-party libraries are able to solve continuous types by implementing Gaussian and Gaussian mixture emissions.

Part 3. Application of HMM

The HMM not only does well in speech recognition but also is good at speech synthesis. The speech synthesis based on the Hidden Markov Model (HMM) has recently been demonstrated to be very effective in synthesizing speech. Identities, emotions and styles of speakers can be changed by speech synthesis. Secondly, the Hidden Markov Model (HMM) is used to recognize speech. Then the recognized sequences are used as tags for speech synthesis. So, the purpose of this work is to add the function of many -to-many.

[1] Tokuda, Keiichi, et al. “Speech synthesis based on Hidden Markov models.” Proceedings of the IEEE 101.5 (2013):1234-1252.

Individual contribution:
Ziqiao Gao: Optimize, implement and debug the HMM algorithm for part 1&2

He Chang: Optimize, implement and debug the HMM algorithm for part 1&2

Fanlin Qin: Research online and use the open source python library to implement HMM algorithm applications and complete part 2&3.