An Analysis of Hidden Markov Models in a Sensory Robotic Enviroment

Israel Nolazco
israelnolazco@lewisu.edu
CPSC-57200-002, Fall 2
Artificial Intelligence 2
Lewis University

I. Introduction

In recent years autonomous vehicles have stride to be fully autonomous and there are several methods in which this can be achieved. However, a great deal of investment goes towards sensory accuracy. As an example, LIDAR is one of the prominent examples of technologies that can fulfill this quest[1]. LIDAR is formidable in object classification and detection which of course come with the cost and unique implementation. However, regardless of the technology and application there will be a small range of error that could lead to the complete loss of the robot. This is of course quite common in unknown environments. In this document we will explore the capabilities of Hidden Markov Model in a simulated 4x4 environment. The aforementioned environment will have obstacles in which the robot will not be available to move based on the sensory response.

II. Methods

The Hidden Markov Model as the name implies shows only the observational data is radially available, but not the information regarding the state of the robot[2]. In essence, we are conduction an estimated guess based on the information gathered from our sensory data. Utilizing the information gathered from Figure 1. The robotic environment is shown only to have a sixteen-potential location, with only twelve of those being available to move. The robot in this environment can only move in four possible directions: North, South, East, and West. Additionally, to fully grasp the capabilities of The Hidden Markov Model the robot will have an equal probability of being located in either twelve possible locations. Figure 2. Shows the transition matrix in which reflects the probability of the robot possible locations based on the number box. Additionally, the image shows an equal probability distribution for its initial state. Lastly, the environment will be simulated with the programming language Python. The analysis was conducted by determining the position of the robot after taking six steps with the following observations: NSW, SE, NW, S, E, and E.

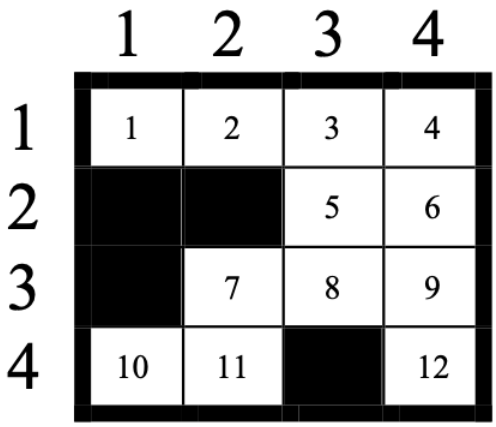

Figure 1. Robot's environment in a 4x4 grid.

```
transition_matrix = np.array([[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                              [1/2, 0, 1/2, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                              [0, 1/3, 0, 1/3, 1/3, 0, 0, 0, 0, 0, 0, 0],
                              [0, 0, 1/2, 0, 0, 1/2, 0, 0, 0, 0, 0, 0],
                              [0, 0, 1/3, 0, 0, 1/3, 0, 1/3, 0, 0, 0, 0],
                              [0, 0, 0, 1/3, 1/3, 0, 0, 0, 1/3, 0, 0, 0],
                              [0, 0, 0, 0, 0, 0, 0, 1/2, 0, 0, 1/2, 0],
                              [0, 0, 0, 0, 1/3, 0, 1/3, 0, 1/3, 0, 0, 0],
                              [0, 0, 0, 0, 0, 1/3, 0, 1/3, 0, 0, 0, 1/3],
                              [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
                              [0, 0, 0, 0, 0, 0, 1/2, 0, 0, 1/2, 0, 0],
                              [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]])

#uniform probability distribution for robot
initial_state=np.array([1/12,1/12,1/12,1/12,1/12,1/12,1/12,1/12,1/12,1/12,1/12,1/12])
```

Figure 2. Transition Matrix and Initial state, showing an uniform probability distribution.

III. Analysis

The aid of the article, “Practical Tutorial, - Robot Localization using Hidden Markov Model” was used as the foundation for the source code required in this analysis. The article additionally offers an example of a real-life experiment, The Groundhog Experiment was deployed in an abandoned coal mine[3]. The environment required a robot to fully map the mine without the guidance of a human. The resolution to this puzzle was the deployment of the Hidden Markov Model. With the background of this application, the integration of this analysis was seamless. Nonetheless, few adjustments were necessary. As an example, figure 3. Shows the error rate to be a value of 0.2 or 20% which is relatively small enough to represent a real-world scenario. Furthermore, for each observation/case mentioned in our methods section, the number of errors was pre-calculated in and entered in our observation matrix. Let us review observation NSW with spot 1. Looking at figure 1. We can quickly realize that spot 1 shows an obstacle in the North, South, and West sides of the spot. Thus, the entry in the 0^{th} element of the array was 0. There is no number of errors being reported. This is of course then repeated for all of the spots in figure 1.

```
observation_matrix_NSW = Model.create_observation_matrix(0.2,[0,1,2,3,2,4,1,2,4,0,3,2])

case_1 = Model.prediction(observation_matrix_NSW)
print()
print(case_1)
```

```
[[0.4096 0.     0.     0.     0.     0.     0.     0.     0.     0.
  0.     0.    ]
 [0.     0.1024 0.     0.     0.     0.     0.     0.     0.     0.
  0.     0.    ]
 [0.     0.     0.0256 0.     0.     0.     0.     0.     0.     0.
  0.     0.    ]
 [0.     0.     0.     0.0064 0.     0.     0.     0.     0.     0.
  0.     0.    ]
 [0.     0.     0.     0.     0.0256 0.     0.     0.     0.     0.
  0.     0.    ]
 [0.     0.     0.     0.     0.     0.0016 0.     0.     0.     0.
  0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.1024 0.     0.     0.
  0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.0256 0.     0.
  0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.0016 0.
  0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.4096
  0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
  0.0064 0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
  0.     0.0256]]

[0.27379679 0.18253119 0.0456328  0.0057041  0.0342246  0.00249554
 0.114082   0.0399287  0.00356506 0.27379679 0.01283422 0.0114082 ]
```

Figure 3. An observation matrix for NSW

The observation matrix is then run through all six steps: NSW, SE, NW, S, E, and E. The final results are shown in Figure 4. The prediction model shows a higher probability for spot number six and nine. However, with the combination of our error, and our transition matrix the actual values of probabilities are shown below the printed matrix. The 12x1 array shows a higher probability in spot number six than spot number nine. Therefore, after all six observations the robot must be located in spot number six. Additionally, to verify the validity of the results one can simply add all elements in the array. The result is equal to one.

```
In [7]: bservation_matrix_E = Model.create_observation_matrix(0.2,[4,3,2,1,2,0,3,2,0,4,1,2])

        case_6 = Model.prediction(observation_matrix_E)

        print()
        print(case_6)
```

```
[[0.0016 0.     0.     0.     0.     0.     0.     0.     0.     0.
  0.     0.    ]
 [0.     0.0064 0.     0.     0.     0.     0.     0.     0.     0.
  0.     0.    ]
 [0.     0.     0.0256 0.     0.     0.     0.     0.     0.     0.
  0.     0.    ]
 [0.     0.     0.     0.1024 0.     0.     0.     0.     0.     0.
  0.     0.    ]
 [0.     0.     0.     0.     0.0256 0.     0.     0.     0.     0.
  0.     0.    ]
 [0.     0.     0.     0.     0.     0.4096 0.     0.     0.     0.
  0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.0064 0.     0.     0.
  0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.0256 0.     0.
  0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.4096 0.
  0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.0016
  0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
  0.1024 0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
  0.     0.0256]]

[1.54268892e-06 1.76514879e-04 1.92691484e-02 8.44699691e-04
 3.74484764e-04 6.73383308e-01 5.45799011e-06 1.42731726e-01
 5.76017259e-04 4.86182780e-08 5.82151526e-02 1.04421899e-01]
```

Figure 4. Final results after all six observations/cases.

IV. Conclusion

The Hidden Markov Model is a great alternative for understanding the environment and finding the location of robot at any point by gathering information on its surroundings. Alternatively, the only potential problems with this method are the possibilities of getting multiple locations with the same probabilities as seen in our observation matrix. However, an alternative to this is to utilize a filtering calculation. In this document the implementation was of filtering was not necessary, but it is something to be consider in a real-life application rather than a theoretical one.

References

[1] S. Ipek and R. Kapusta, "LIDAR for Autonomous System Design: Object Classification or Object Detection?", *Embedded Computing Design*, 2020. [Online]. Available: https://www.embedded-computing.com/home-page/lidar-for-autonomous-system-design-object-classification-or-object-detection. [Accessed: 16- Nov- 2020].

[2] S. Glen, "Hidden Markov Model: Simple Definition & Overview", *Statistics How To*, 2020. [Online]. Available: https://www.statisticshowto.com/hidden-markov-model/. [Accessed: 16- Nov- 2020].

[3] D. Bogunowicz, "Practical tutorial- Robot localization using Hidden Markov Models", *DTRANSPOSED*, 2020. [Online]. Available: https://dtransposed.github.io/blog/Robot-Localization.html. [Accessed: 16- Nov- 2020].

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn
from mpl_toolkits.mplot3d import Axes3D


'''
Student: Israel Nolazco
Date: November 15th, 2020
Course: Artificial Intelligence 2
Semester: Fall 2
Assigment Name: Homework # 2
'''

class HMM(object):

    def __init__(self, transition_matrix_matrix,current_state):
        self.transition_matrix = transition_matrix
        self.current_state = current_state

    def prediction(self,observation_matrix):
        new_state = np.dot(observation_matrix,np.dot(self.transition_matrix.transpose(),self.current_state))
        new_state_normalized = new_state/np.sum(new_state)
        self.current_state = new_state_normalized
        return new_state_normalized

    def create_observation_matrix(self,error_rate, no_discrepancies):
        sensor_list=[]
        for number in no_discrepancies:
            probability=(1-error_rate)**(4-number)*error_rate**number
            sensor_list.append(probability)
            observation_matrix = np.zeros((len(sensor_list),len(sensor_list)))
            np.fill_diagonal(observation_matrix,sensor_list)
        print(observation_matrix)
        return observation_matrix


# vectors need to add to 1
#transition matrix for following cordinates according to 4X4 grid in assigment
#(1,1), (1,2), (1,3), (1,4), (2,3), (2,4), (3,2), (3,3), (3,4), (4,1), (4,2), (4,4)

transition_matrix = np.array([[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                              [1/2, 0, 1/2, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                              [0, 1/3, 0, 1/3, 1/3, 0, 0, 0, 0, 0, 0, 0],
                              [0, 0, 1/2, 0, 0, 1/2, 0, 0, 0, 0, 0, 0],
                              [0, 0, 1/3, 0, 0, 1/3, 0, 1/3, 0, 0, 0, 0],
                              [0, 0, 0, 1/3, 1/3, 0, 0, 0, 1/3, 0, 0, 0],
                              [0, 0, 0, 0, 0, 0, 0, 1/2, 0, 0, 1/2, 0],
                              [0, 0, 0, 0, 1/3, 0, 1/3, 0, 1/3, 0, 0, 0],
                              [0, 0, 0, 0, 0, 1/3, 0, 1/3, 0, 0, 0, 1/3],
                              [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
                              [0, 0, 0, 0, 0, 0, 1/2, 0, 1/2, 0, 0, 0],
                              [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]])

#uniform probability distribution for robot
initial_state=np.array([1/12,1/12,1/12,1/12,1/12,1/12,1/12,1/12,1/12,1/12,1/12,1/12])

#modeling fuction
Model = HMM(transition_matrix,initial_state)
```

In [2]:
```
# create observation matrices
# numbers inside the bracket represent how wrong the observation is based on the location of
#the robot
#results shows probability of location of robot

observation_matrix_NSW = Model.create_observation_matrix(0.2,[0,1,2,3,2,4,1,2,4,0,3,2])

case_1 = Model.prediction(observation_matrix_NSW)
print()
print(case_1)
```

```
[[0.4096 0.     0.     0.     0.     0.     0.     0.     0.     0.
  0.     0.    ]
 [0.     0.1024 0.     0.     0.     0.     0.     0.     0.     0.
  0.     0.    ]
 [0.     0.     0.0256 0.     0.     0.     0.     0.     0.     0.
  0.     0.    ]
 [0.     0.     0.     0.0064 0.     0.     0.     0.     0.     0.
  0.     0.    ]
 [0.     0.     0.     0.     0.0256 0.     0.     0.     0.     0.
  0.     0.    ]
 [0.     0.     0.     0.     0.     0.0016 0.     0.     0.     0.
  0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.1024 0.     0.     0.
  0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.0256 0.     0.
  0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.0016 0.
  0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.4096
  0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
  0.0064 0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
  0.     0.0256]]

[0.27379679 0.18253119 0.0456328  0.0057041  0.0342246  0.00249554
 0.114082   0.0399287  0.00356506 0.27379679 0.01283422 0.0114082 ]
```

In [3]:
```
observation_matrix_SE = Model.create_observation_matrix(0.2,[3,2,3,2,3,1,4,1,1,3,0,1])

case_2 = Model.prediction(observation_matrix_SE)

print()
print(case_2)
```

```
[[0.0064 0.     0.     0.     0.     0.     0.     0.     0.     0.
  0.     0.    ]
 [0.     0.0256 0.     0.     0.     0.     0.     0.     0.     0.
  0.     0.    ]
 [0.     0.     0.0064 0.     0.     0.     0.     0.     0.     0.
  0.     0.    ]
 [0.     0.     0.     0.0256 0.     0.     0.     0.     0.     0.
  0.     0.    ]
 [0.     0.     0.     0.     0.0064 0.     0.     0.     0.     0.
  0.     0.    ]
 [0.     0.     0.     0.     0.     0.1024 0.     0.     0.     0.
  0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.0016 0.     0.     0.
  0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.1024 0.     0.
  0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.1024 0.
  0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.0064
  0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
  0.4096 0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
  0.     0.1024]]

[3.73725356e-03 4.73385451e-02 4.32119942e-03 2.62775641e-03
 1.20195525e-03 1.01237284e-02 2.01947946e-04 4.56256372e-02
 1.67397816e-02 2.62775641e-04 8.67042825e-01 7.78594491e-04]
```

In [4]:
```
observation_matrix_NW = Model.create_observation_matrix(0.2,[1,2,1,2,1,3,0,3,3,1,4,3])

case_3 = Model.prediction(observation_matrix_NW)

print()
print(case_3)
```

```
[[0.1024 0.     0.     0.     0.     0.     0.     0.     0.     0.
  0.     0.    ]
 [0.     0.0256 0.     0.     0.     0.     0.     0.     0.     0.
  0.     0.    ]
 [0.     0.     0.1024 0.     0.     0.     0.     0.     0.     0.
  0.     0.    ]
 [0.     0.     0.     0.0256 0.     0.     0.     0.     0.     0.
  0.     0.    ]
 [0.     0.     0.     0.     0.1024 0.     0.     0.     0.     0.
  0.     0.    ]
 [0.     0.     0.     0.     0.     0.0064 0.     0.     0.     0.
  0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.4096 0.     0.     0.
  0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.0064 0.     0.
  0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.0064 0.
  0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.1024
  0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
  0.0016 0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
  0.     0.0064]]

[3.24361408e-05 6.44116179e-03 3.97092069e-04 1.07917659e-04
 4.38020426e-04 3.84870934e-04 1.62305343e-06 7.25325220e-01
 3.14441050e-05 1.42422552e-07 2.66819857e-01 2.02150050e-05]
```

In [6]:
```
observation_matrix_E = Model.create_observation_matrix(0.3,[4,3,2,1,2,0,3,2,0,4,1,2])

case_5 = Model.prediction(observation_matrix_E)

print()
print(case_5)
```

```
[[0.0081 0.     0.     0.     0.     0.     0.     0.     0.     0.
  0.     0.    ]
 [0.     0.0189 0.     0.     0.     0.     0.     0.     0.     0.
  0.     0.    ]
 [0.     0.     0.0441 0.     0.     0.     0.     0.     0.     0.
  0.     0.    ]
 [0.     0.     0.     0.1029 0.     0.     0.     0.     0.     0.
  0.     0.    ]
 [0.     0.     0.     0.     0.0441 0.     0.     0.     0.     0.
  0.     0.    ]
 [0.     0.     0.     0.     0.     0.2401 0.     0.     0.     0.
  0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.0189 0.     0.     0.
  0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.0441 0.     0.
  0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.2401 0.
  0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.0081
  0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
  0.1029 0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
  0.     0.0441]]

[3.37927594e-04 4.03481724e-05 1.95406280e-03 3.47443948e-04
 1.38268258e-01 6.54544740e-04 9.18568816e-02 8.98609321e-05
 7.52445039e-01 1.39983741e-02 1.27158407e-06 5.98771330e-06]
```

In [7]:
```
bservation_matrix_E = Model.create_observation_matrix(0.2,[4,3,2,1,2,0,3,2,0,4,1,2])

case_6 = Model.prediction(observation_matrix_E)

print()
print(case_6)
```

```
[[0.0016 0.     0.     0.     0.     0.     0.     0.     0.     0.
  0.     0.    ]
 [0.     0.0064 0.     0.     0.     0.     0.     0.     0.     0.
  0.     0.    ]
 [0.     0.     0.0256 0.     0.     0.     0.     0.     0.     0.
  0.     0.    ]
 [0.     0.     0.     0.1024 0.     0.     0.     0.     0.     0.
  0.     0.    ]
 [0.     0.     0.     0.     0.0256 0.     0.     0.     0.     0.
  0.     0.    ]
 [0.     0.     0.     0.     0.     0.4096 0.     0.     0.     0.
  0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.0064 0.     0.     0.
  0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.0256 0.     0.
  0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.4096 0.
  0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.0016
  0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
  0.1024 0.    ]
 [0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
  0.     0.0256]]

[1.54268892e-06 1.76514879e-04 1.92691484e-02 8.44699691e-04
 3.74488764e-04 6.73383308e-01 5.45799011e-06 1.42731726e-01
 5.76017259e-04 4.86182780e-08 5.82151526e-02 1.04421899e-01]
```

In [ ]:
```

```

In [ ]:
```

```