Demetrius Johnson

UM-Dearborn CIS-479

Program 2 Report: Hidden Markov Model – Robot Localization

With Prof. Dr. Shengquan Wang

Summer II 2022

8-09-22

Contents

Transitional Probability

```
//moving transition probabilities:
const float forward_probability = (float)0.8;     //move forward in desired direction = 
80%
const float left_probability = (float)0.1;    //drift left in undesired direction = 10%
const float right_probability = (float)0.1;   //drift right in undesired direction = 10%
```

Evidence Conditional Probability

```
//obstacle sensor transition probabilities:
const float sensing_OBJ_correctly_probability = (float)0.85; //correctly sense square as 
an obstacle = 85%
const float sensing_OBJ_incorrectly_probability = (float)0.15; //false negative(no 
obstacle) - incorrectly sense square as NOT an obstacle (open square) = 15%

//open-square (non-obstacle) sensor transition probabilities:
const float sensing_no_OBJ_correctly_probability = (float)0.95; //correctly sense square 
as NOT an obstacle = 95%
const float sensing_no_OBJ_incorrectly_probability = (float)0.05; //false positive(is an 
obstacle) - incorrectly sense square as an obstacle = 5%
```

Filtering (from sensing)

```
void updateProb_sensing_filtering(table& table_struct, Location* 
sensor_evidence_bit_WNES) {


    //Filtering: P(St|Z1=z1, …, Zt=zt) ∝ P(Zt=zt|St) P(St|Z1=z1, …, Zt-1=zt-1)

    /* note that Z_t represents an evidence variable --> evidence from the neighbor 
sensor
        Zt is composed of 4 random variables for four directions:
        Zt = (ZW,t,  ZN,t,  ZE,t, ZS,t)

        For each state St, conditionally independent:
        P(Zt|St) = P(ZW,t|St) P(ZN,t|St) P(ZE,t|St) P(ZS,t|St)


        note that:

        P(Zt=zt|St)==  likelihood
        P(St|Z1=z1, …, Zt-1=zt-1) ==  prior
        P(St|Z1=z1, …, Zt=zt) == posterior

    */

    float sum_of_proportions = 0; //sum of proportions == SUM for all States (S_t) of [ 
P(Zt=zt|St) P(St|Z1=z1, …, Zt-1=zt-1)  ].
                                  //add all proportions to get total probability of 
evidence Z_t given S_t,
                                  //then we can use this sum to get  P(St|Z1=z1, …, 
Zt=zt) == [  P(Zt=zt|St) P(St|Z1=z1, …, Zt-1=zt-1) / sum_of_proportions  ].
                                  //This is the same thing as saying: [1 of the 
proportions / sum_of_propotions]...
```

```
                              //--> and we can do this for all states to find the
probabiility the robot is at any given location given evidence(s) Z_t.

    Location neighbors_WNES[4]; //use this to get the true neighbor based om the true map
the robot has access to to compare it with the sensor evidence

    for (int row = 0; row < table_struct.numRows; row++) {
        for (int col = 0; col < table_struct.numCols; col++) {

            if (is_obstacleLocation(row, col)) //no calculation necessary for obstacle
case --> robot knows it cannot be there
                continue;

            //check neighboring squares for obstacles so we can determine which sensor
error transistion probability to use at each possible state S_t
            getNeighbors_WNES(row, col, neighbors_WNES);

            //calculate likelihood --> P(Z_t | S_t) --> obstacle or open square
probability at a given state S_t = s_t, based on evidence from sensor:
            float sensorProb_WNES[4];
            for (int i = 0; i < 4; i++) {

                //case: correctly sense an obstacle ==85%
                if (sensor_evidence_bit_WNES[i] == OBSTACLE && neighbors_WNES[i] ==
OBSTACLE)

                    sensorProb_WNES[i] = sensing_OBJ_correctly_probability;

                //case: incorrectly sense an obstacle as open square (false negative)
==15%
                else if (sensor_evidence_bit_WNES[i] == OPEN_SQUARE && neighbors_WNES[i]
== OBSTACLE)
                    sensorProb_WNES[i] = sensing_OBJ_incorrectly_probability;

                //case: correctly sense an open sqaure ==95%
                else if (sensor_evidence_bit_WNES[i] == OPEN_SQUARE && neighbors_WNES[i]
== OPEN_SQUARE)
                    sensorProb_WNES[i] = sensing_no_OBJ_correctly_probability;

                //case: incorrectly sense an open sqaure as an obstacle (false positive)
==5%
                else if (sensor_evidence_bit_WNES[i] == OBSTACLE && neighbors_WNES[i] ==
OPEN_SQUARE)
                    sensorProb_WNES[i] = sensing_no_OBJ_incorrectly_probability;
            }

            //multiply all conditionall indepdent likelihoods to get total likelihood of
a state S_t given given sensor evidence Z_t
            float state_totalLikelihood = 1;
            for (int i = 0; i < 4; i++)
                state_totalLikelihood *= sensorProb_WNES[i];

            //Now, multiply total likelihood by the prior and add that to the sum
            sum_of_proportions += (state_totalLikelihood *
table_struct.tablePos_locationProb_prior[row][col]);
        }
    }
```

Here, I demonstrate how to get the sum of all proportions of probabilities given an evidence:

```
    //now, sum of proportions has been acquired, we now can normalize and find the posterior probabilitl for all states:
    // P(St|Z1=z1, …, Zt=zt) == [  P(Zt=zt|St) P(St|Z1=z1, …, Zt-1=zt-1) / sum_of_proportions  ]

    for (int row = 0; row < table_struct.numRows; row++) {
        for (int col = 0; col < table_struct.numCols; col++) {

            if (is_obstacleLocation(row, col)) //no calculation necessary for obstacle case --> robot knows it cannot be there
                continue;

            //check neighboring squares for obstacles so we can determine which sensor error transistion probability to use at each possible state S_t
            getNeighbors_WNES(row, col, neighbors_WNES);

            //calculate likelihood --> P(Z_t | S_t) --> obstacle or open square probability at a given state S_t = s_t, based on evidence from sensor:
            float sensorProb_WNES[4];
            for (int i = 0; i < 4; i++) {

                //case: correctly sense an obstacle ==85%
                if (sensor_evidence_bit_WNES[i] == OBSTACLE && neighbors_WNES[i] == OBSTACLE)
                    sensorProb_WNES[i] = sensing_OBJ_correctly_probability;

                //case: incorrectly sense an obstacle as open square (false negative) ==15%
                else if (sensor_evidence_bit_WNES[i] == OPEN_SQUARE && neighbors_WNES[i] == OBSTACLE)
                    sensorProb_WNES[i] = sensing_OBJ_incorrectly_probability;

                //case: correctly sense an open sqaure ==95%
                else if (sensor_evidence_bit_WNES[i] == OPEN_SQUARE && neighbors_WNES[i] == OPEN_SQUARE)
                    sensorProb_WNES[i] = sensing_no_OBJ_correctly_probability;

                //case: incorrectly sense an open sqaure as an obstacle (false positive) ==5%
                else if (sensor_evidence_bit_WNES[i] == OBSTACLE && neighbors_WNES[i] == OPEN_SQUARE)
                    sensorProb_WNES[i] = sensing_no_OBJ_incorrectly_probability;
            }

            //multiply all conditionall indepdent likelihoods to get total likelihood of a state S_t given given sensor evidence Z_t
            float state_totalLikelihood = 1;
            for (int i = 0; i < 4; i++)
                state_totalLikelihood *= sensorProb_WNES[i];
```

Here, I show a repeated process, but only this time we use our sum of proportions to get the probability for all locations:

```
            //Now, multiply total likelihood by the prior
           //Now, at this moment we have calculated one proportion; now,
```

```
            //we normalize since we already have sum of all proportions, to get posterior for the current state at [row][col]
            table_struct.tablePos_locationProb_posterior[row][col] = (state_totalLikelihood * table_struct.tablePos_locationProb_prior[row][col]) / sum_of_proportions;


        }
    }

}
```

Prediction (from movement)

```
void updatProb_moving_prediction(table& table_struct, Direction_Cardinal move_direction)
{


    //Prediction: P(St+1|Z1=z1, …, Zt=zt) = ∑sP(St+1|St=s) P(St|Z1=z1, …, Zt=zt)
    //The prediction is essentially just the sum of all probabilities that you can get to state S_t+1, given all possible prior states S_t.

    //first, reinitialize the posterior table, so that we can incrementally sweep across it and add the probability of getting to one state from one of the other possible states
    for (int i = 0; i < table_struct.numRows; i++) {

        for (int j = 0; j < table_struct.numCols; j++) {

            if (is_obstacleLocation(i, j))
                continue;
            else
                table_struct.tablePos_locationProb_posterior[i][j] = 0;
        }
    }
```

Here, I demonstrate how I don't add all probabilities for a location at one time, rather I sweep across the entire matrix and update probabilities as a location should absorb it given do a move from one location to another:

```
    //now sweep across all locations adding any probability from neighbor states that it can be reached from - when done, then all probability for reaching a state S_t+1 from all other possible states S_t will be complete
    for (int row = 0; row < table_struct.numRows; row++) {
        for (int col = 0; col < table_struct.numCols; col++) {

            if (is_obstacleLocation(row, col)) //no need to check impossible cases since robot cannot be at an obstacle location
                continue;

            switch (move_direction) {



            case Direction_Cardinal::WEST:

                prob_moving_WEST(row, col, table_struct);
                break;
            case Direction_Cardinal::NORTH:
```

```
                prob_moving_NORTH(row, col, table_struct);
                break;
            case Direction_Cardinal::EAST:

                prob_moving_EAST(row, col, table_struct);
                break;
            case Direction_Cardinal::SOUTH:

                prob_moving_SOUTH(row, col, table_struct);
                break;
            }
        }

    }
}
```

And here, I demonstrate how I create a transition function that sweeps across the matrix allowing the proper location to absorb probability for all directions as shown in the function above, below I will show you just one of the functions for the sake of space and unnecessary redundancy:

```
void prob_moving_WEST(int row, int col, table& table_struct) {


    //WEST == FORWARD-->(col - 1), LEFT-->(row + 1), RIGHT-->(row - 1):

    //forward
        //check wall bounce back case, otherwise use (col - 1)
    if (is_obstacleLocation(row, col - 1))
        //current [row][col] location absorbs the probability
        table_struct.tablePos_locationProb_posterior[row][col] += (forward_probability *
table_struct.tablePos_locationProb_prior[row][col]);
    else
        //otherwise, forward location absorbs the probability
        table_struct.tablePos_locationProb_posterior[row][col - 1] +=
(forward_probability * table_struct.tablePos_locationProb_prior[row][col]);
    //left
        //check bounce back case, otherwise use (row + 1)
    if (is_obstacleLocation(row + 1, col))
        //current [row][col] location absorbs the probability
        table_struct.tablePos_locationProb_posterior[row][col] += (left_probability *
table_struct.tablePos_locationProb_prior[row][col]);
    else
        //otherwise, left location absorbs the probability
        table_struct.tablePos_locationProb_posterior[row + 1][col] += (left_probability *
table_struct.tablePos_locationProb_prior[row][col]);
    //right
       //check bounce back case, otherwise use (row - 1)
    if (is_obstacleLocation(row - 1, col))
        table_struct.tablePos_locationProb_posterior[row][col] += (right_probability *
table_struct.tablePos_locationProb_prior[row][col]);
    else
        //otherwise, right location absorbs the probability
        table_struct.tablePos_locationProb_posterior[row - 1][col] += (right_probability
* table_struct.tablePos_locationProb_prior[row][col]);
```

```
}
```

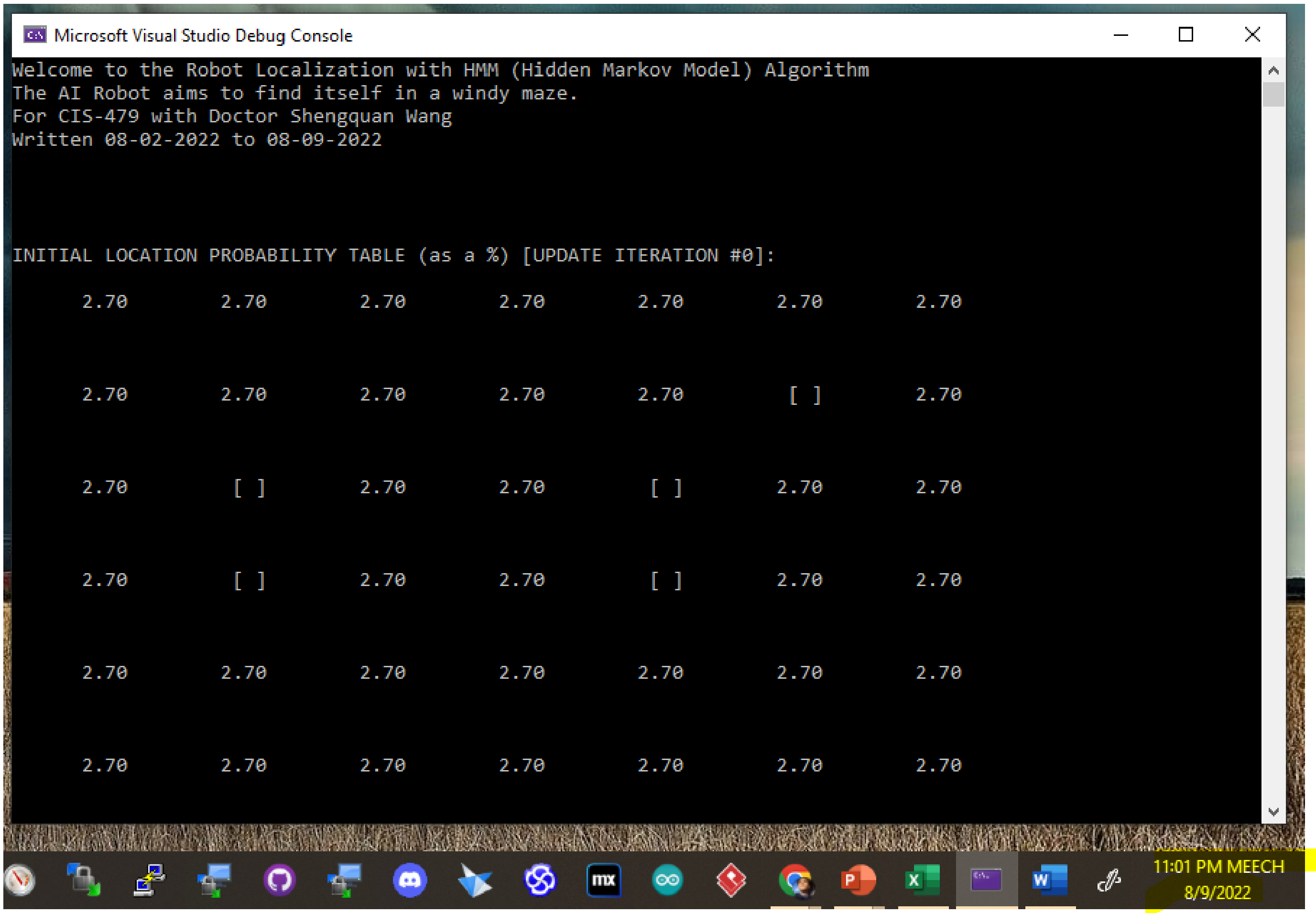
Screenshots of solution output. Notice my last name and the time on the bottom right of each screenshot:

**Notice how you are never 100% certain where the robot is, and also whenever the robot moves once the values converge, it is pretty easy to tell where it is and what square it moves too (with much greater certainty). This is especially demonstrated in my bonus test cases after the standard test cases show a convergence of where the robot most likely is. Also, if we were to just keep sensing and filtering, the result would be that we would become more and more certain – even with the error of the robot adding up. Over time, however, eventually all of the error over adds up (over many, many iterations) and will cause the probabilities to converge to a uniform distribution in this stochastic setup (especially if you moved the robot to every location over and over).

Demetrius Johnson – Program 2 – HMM Robot Localization Probability Algorithm

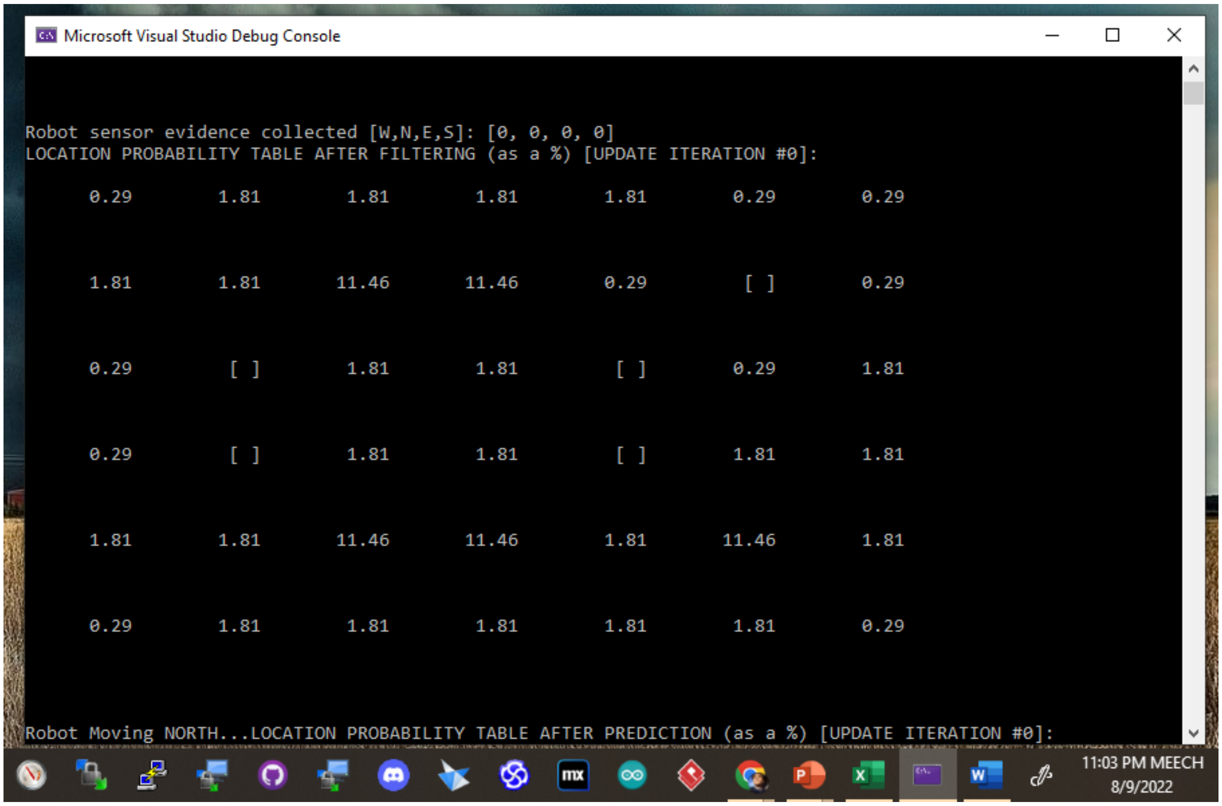
Standard Test Case Screenshots:

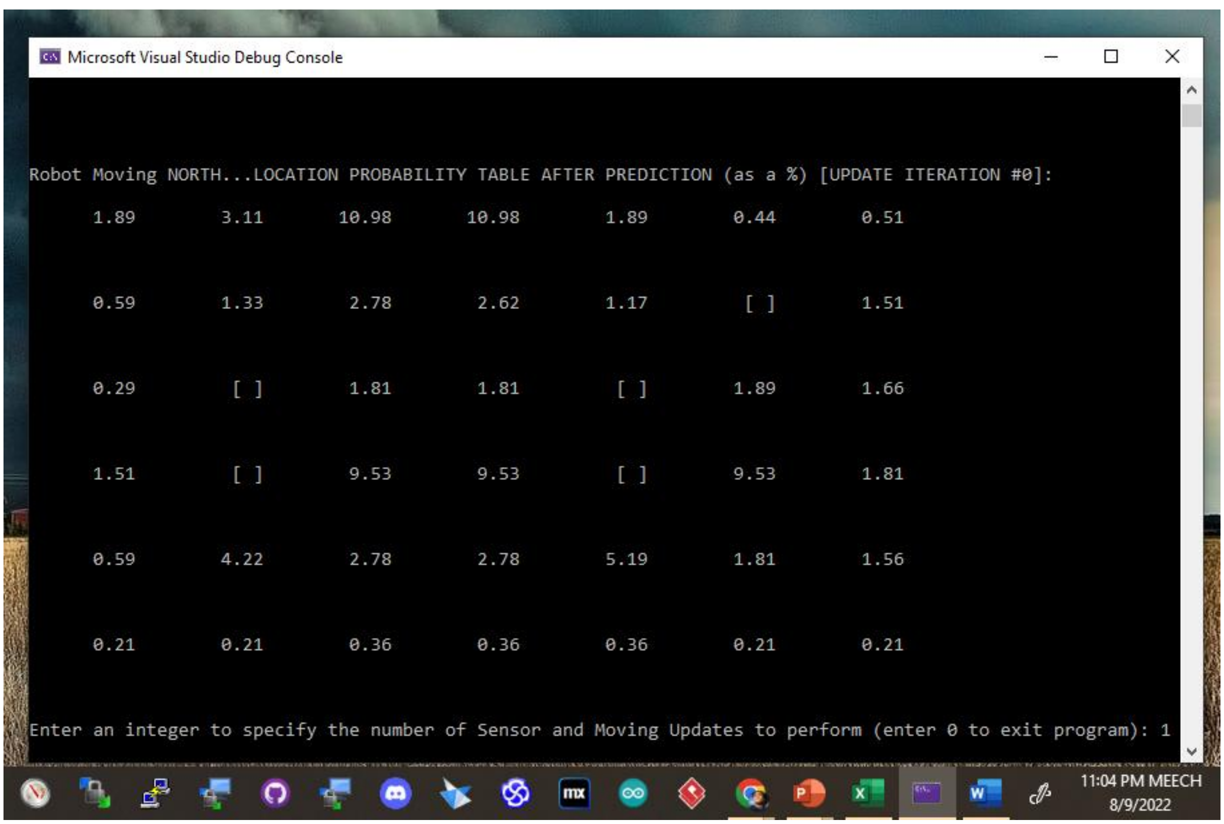
Initial Probabilities == uniform



Demetrius Johnson – Program 2 – HMM Robot Localization Probability Algorithm

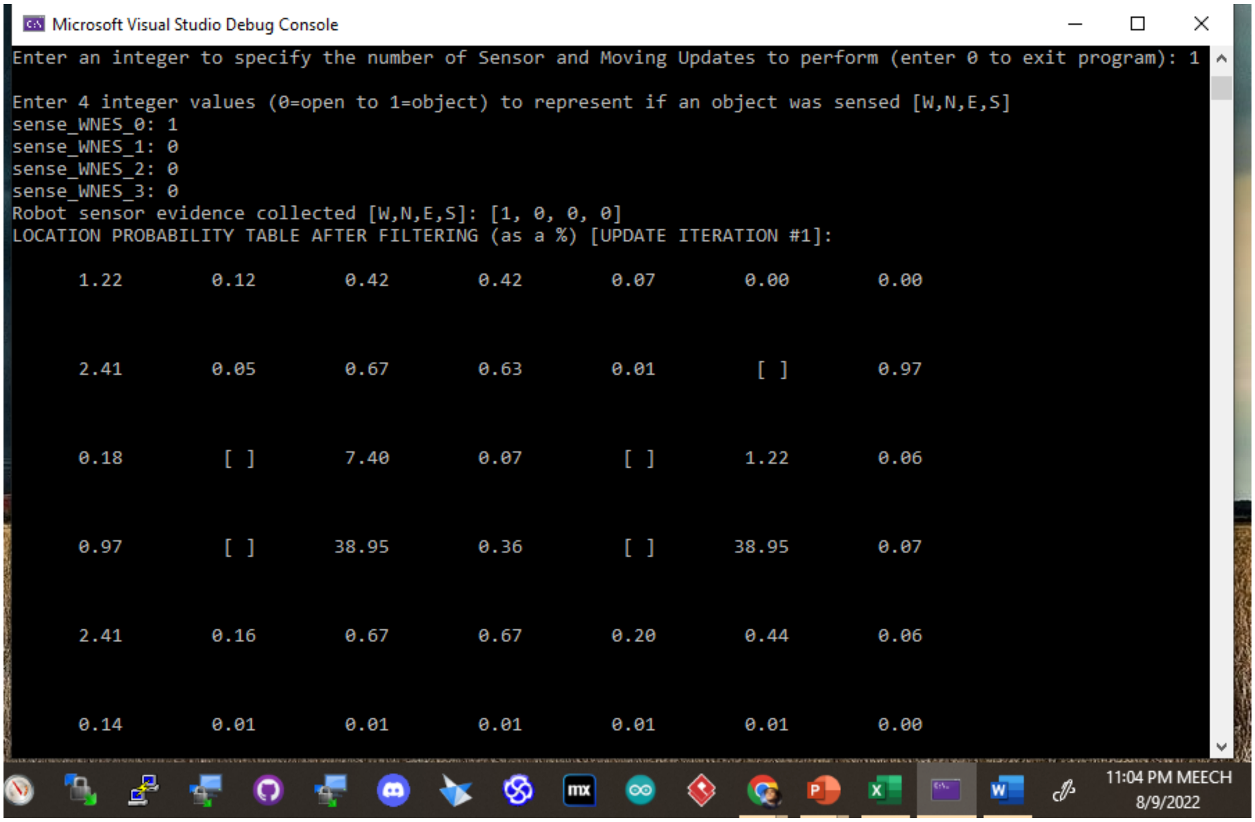
SENSE [0,0,0,0], MOVE N



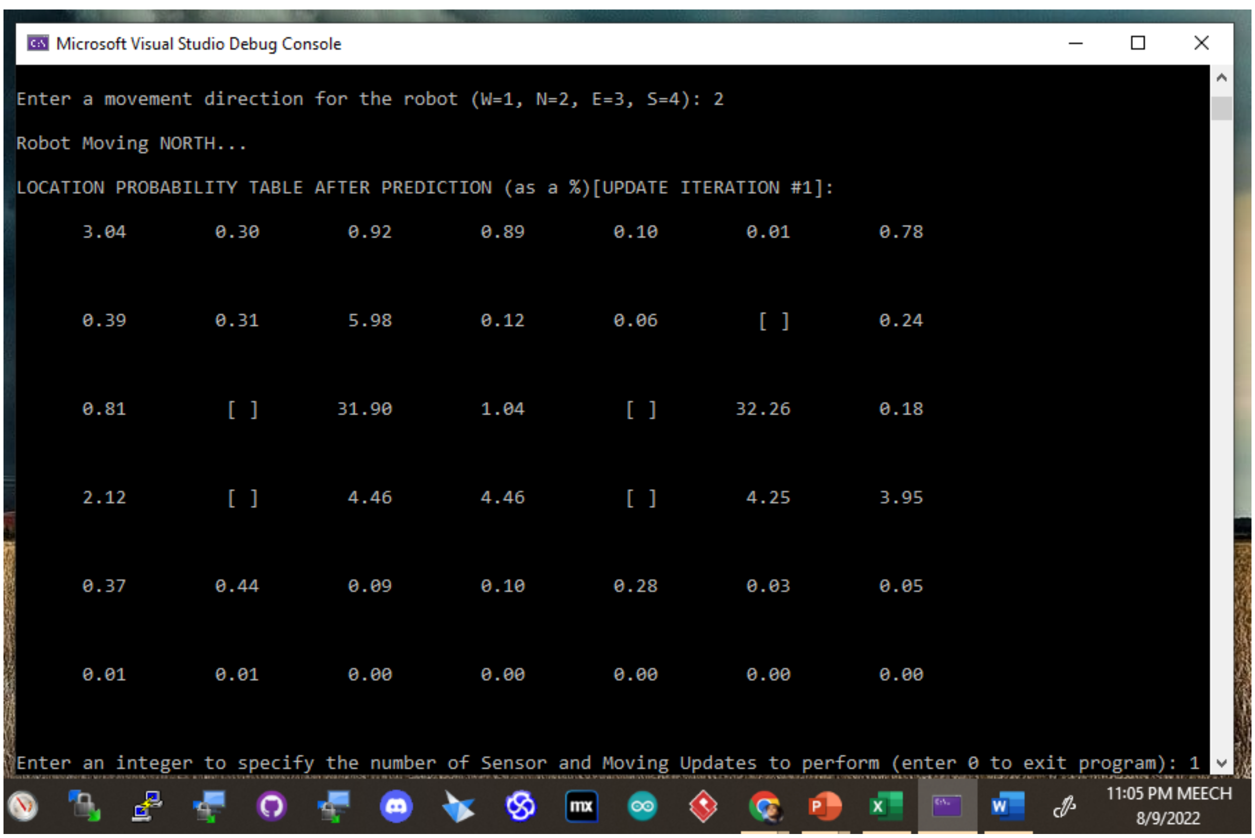


Demetrius Johnson – Program 2 – HMM Robot Localization Probability Algorithm

SENSE [1,0,0,0], MOVE N



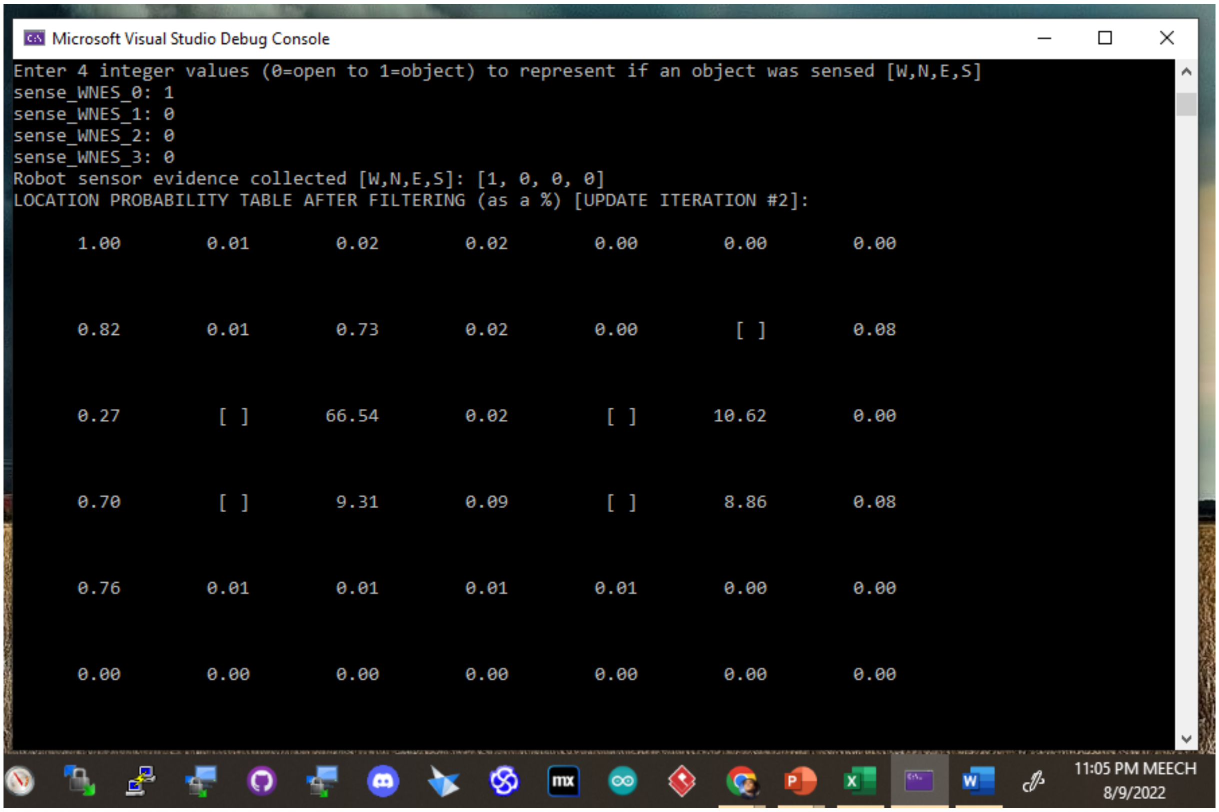

```
Enter an integer to specify the number of Sensor and Moving Updates to perform (enter 0 to exit program): 1

Enter 4 integer values (0=open to 1=object) to represent if an object was sensed [W,N,E,S]
sense_WNES_0: 1
sense_WNES_1: 0
sense_WNES_2: 0
sense_WNES_3: 0
Robot sensor evidence collected [W,N,E,S]: [1, 0, 0, 0]
LOCATION PROBABILITY TABLE AFTER FILTERING (as a %) [UPDATE ITERATION #1]:

      1.22      0.12      0.42      0.42      0.07      0.00      0.00

      2.41      0.05      0.67      0.63      0.01       [ ]      0.97

      0.18       [ ]      7.40      0.07       [ ]      1.22      0.06

      0.97       [ ]     38.95      0.36       [ ]     38.95      0.07

      2.41      0.16      0.67      0.67      0.20      0.44      0.06

      0.14      0.01      0.01      0.01      0.01      0.01      0.00
```




```
Enter a movement direction for the robot (W=1, N=2, E=3, S=4): 2

Robot Moving NORTH...

LOCATION PROBABILITY TABLE AFTER PREDICTION (as a %)[UPDATE ITERATION #1]:

      3.04      0.30      0.92      0.89      0.10      0.01      0.78

      0.39      0.31      5.98      0.12      0.06       [ ]      0.24

      0.81       [ ]     31.90      1.04       [ ]     32.26      0.18

      2.12       [ ]      4.46      4.46       [ ]      4.25      3.95

      0.37      0.44      0.09      0.10      0.28      0.03      0.05

      0.01      0.01      0.00      0.00      0.00      0.00      0.00

Enter an integer to specify the number of Sensor and Moving Updates to perform (enter 0 to exit program): 1
```

Demetrius Johnson – Program 2 – HMM Robot Localization Probability Algorithm

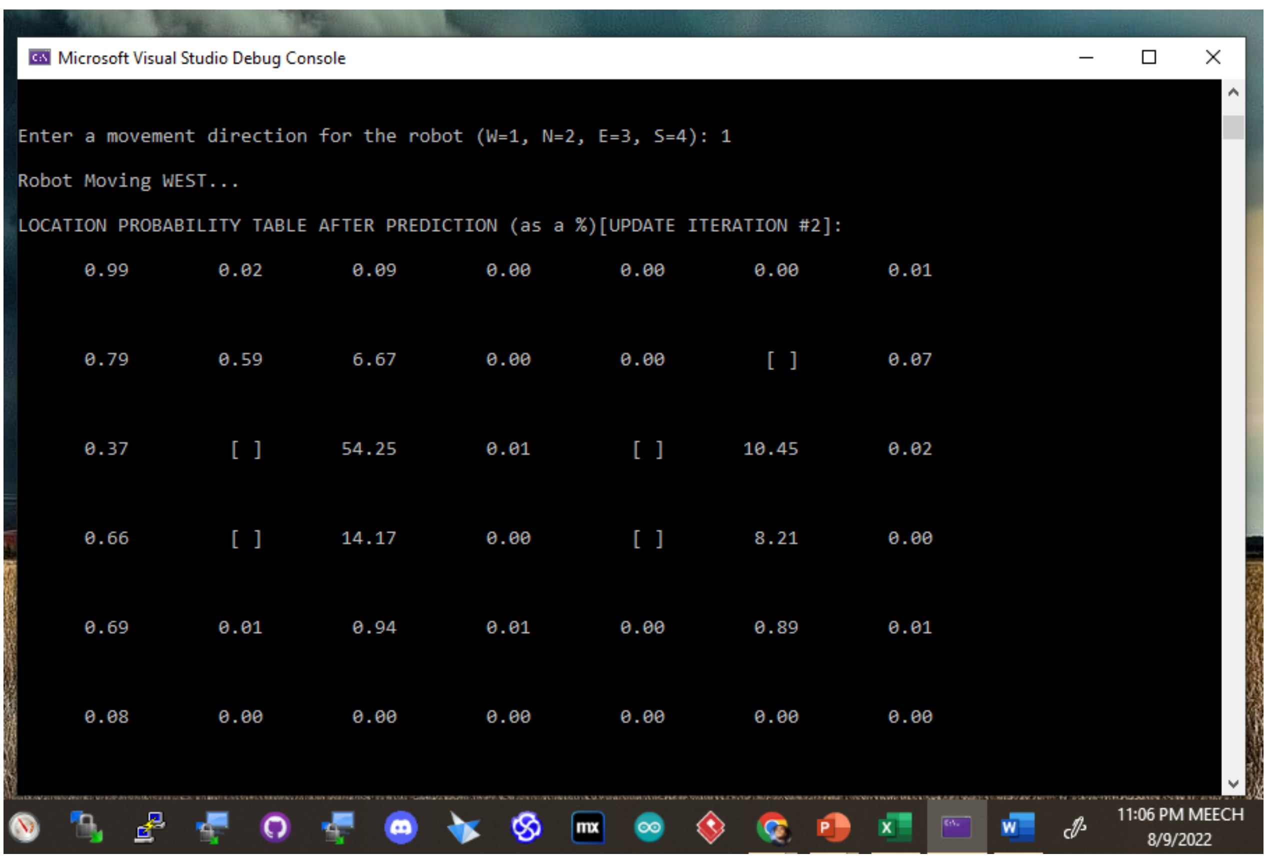
SENSE [1,0,0,0], MOVE W



Microsoft Visual Studio Debug Console

```
Enter 4 integer values (0=open to 1=object) to represent if an object was sensed [W,N,E,S]
sense_WNES_0: 1
sense_WNES_1: 0
sense_WNES_2: 0
sense_WNES_3: 0
Robot sensor evidence collected [W,N,E,S]: [1, 0, 0, 0]
LOCATION PROBABILITY TABLE AFTER FILTERING (as a %) [UPDATE ITERATION #2]:

     1.00       0.01       0.02       0.02       0.00       0.00       0.00


     0.82       0.01       0.73       0.02       0.00       [ ]        0.08


     0.27       [ ]        66.54      0.02       [ ]        10.62      0.00


     0.70       [ ]        9.31       0.09       [ ]        8.86       0.08


     0.76       0.01       0.01       0.01       0.01       0.00       0.00


     0.00       0.00       0.00       0.00       0.00       0.00       0.00
```

11:05 PM MEECH
8/9/2022



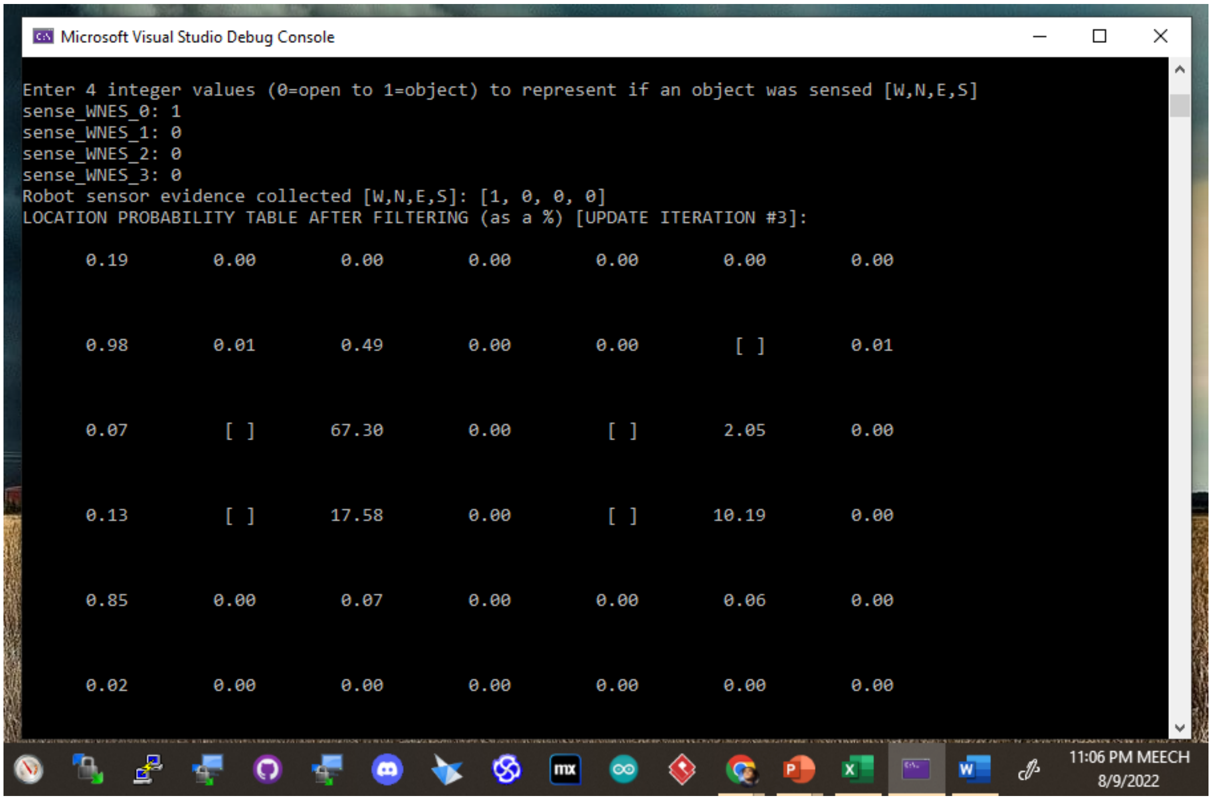
Microsoft Visual Studio Debug Console

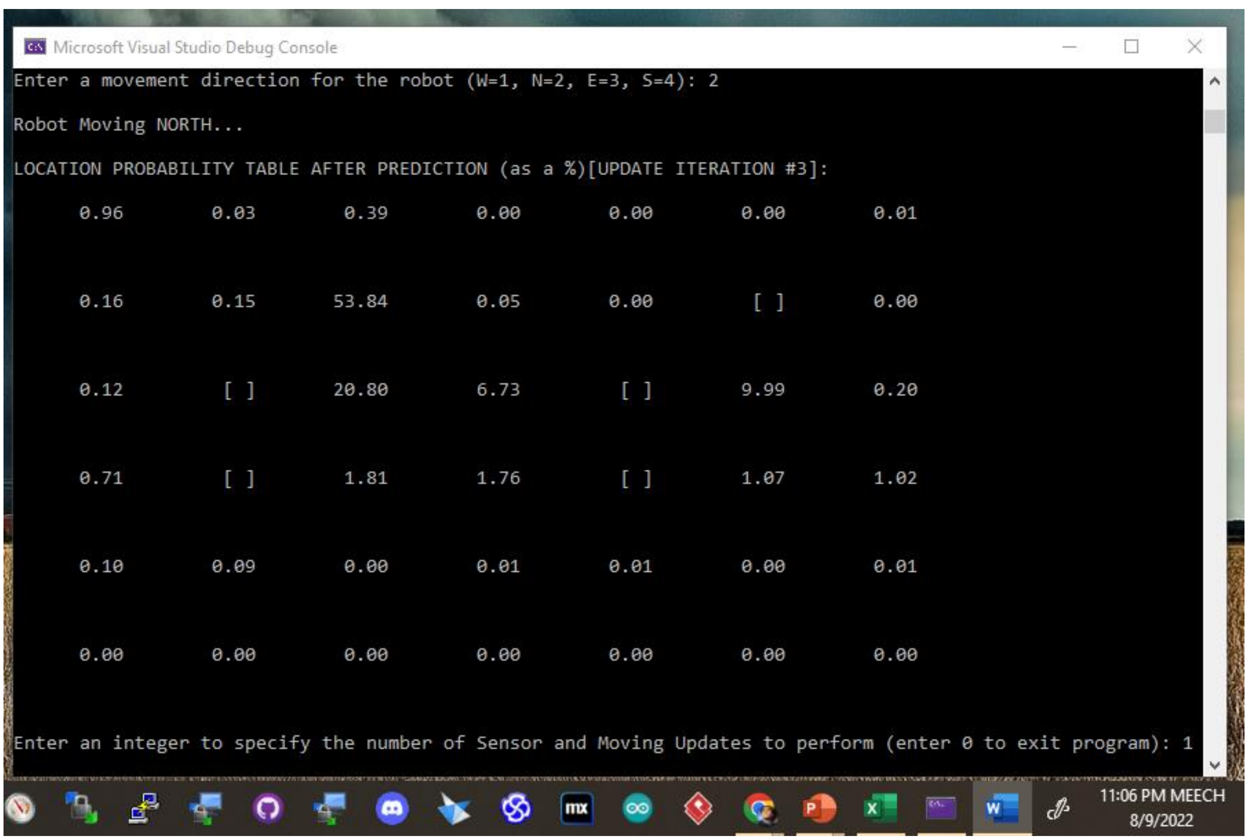
```
Enter a movement direction for the robot (W=1, N=2, E=3, S=4): 1

Robot Moving WEST...

LOCATION PROBABILITY TABLE AFTER PREDICTION (as a %)[UPDATE ITERATION #2]:

     0.99       0.02       0.09       0.00       0.00       0.00       0.01


     0.79       0.59       6.67       0.00       0.00       [ ]        0.07


     0.37       [ ]        54.25      0.01       [ ]        10.45      0.02


     0.66       [ ]        14.17      0.00       [ ]        8.21       0.00


     0.69       0.01       0.94       0.01       0.00       0.89       0.01


     0.08       0.00       0.00       0.00       0.00       0.00       0.00
```

11:06 PM MEECH
8/9/2022

Demetrius Johnson – Program 2 – HMM Robot Localization Probability Algorithm

SENSE [1,0,0,0], MOVE N

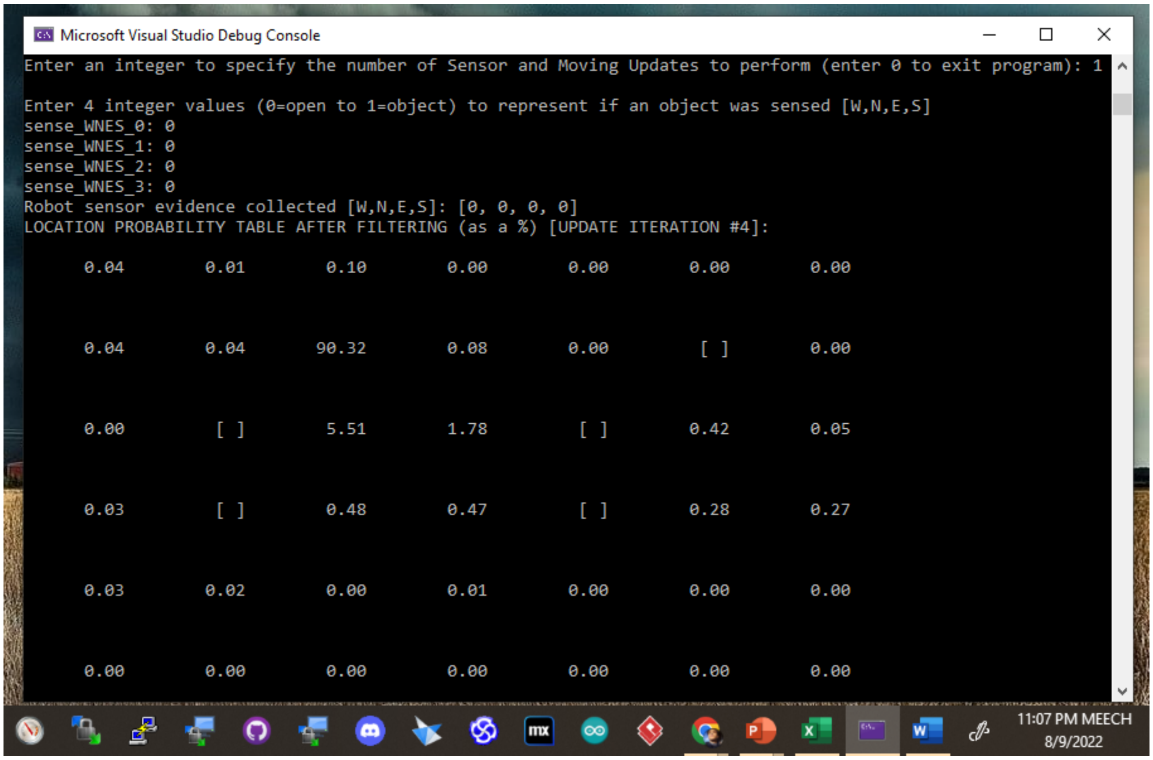




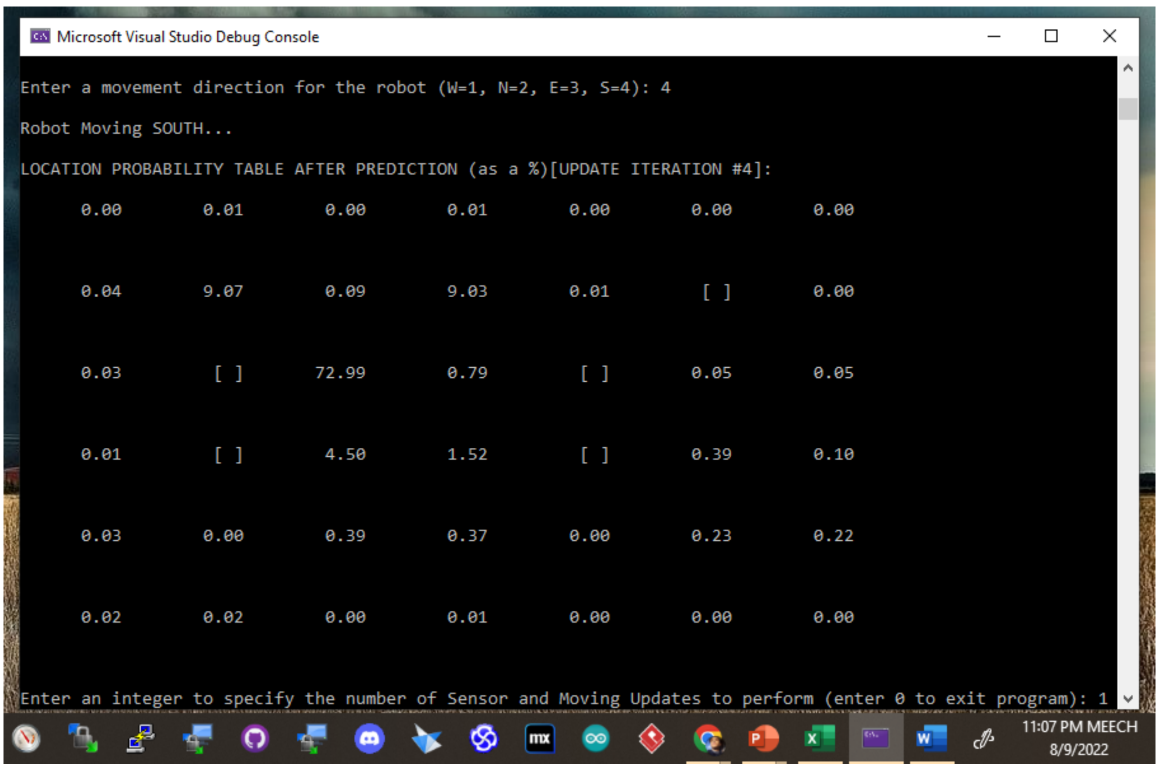
Demetrius Johnson – Program 2 – HMM Robot Localization Probability Algorithm

BONUS moving EAST and SOUTH screenshots:

SENSE [0,0,0,0], MOVE S



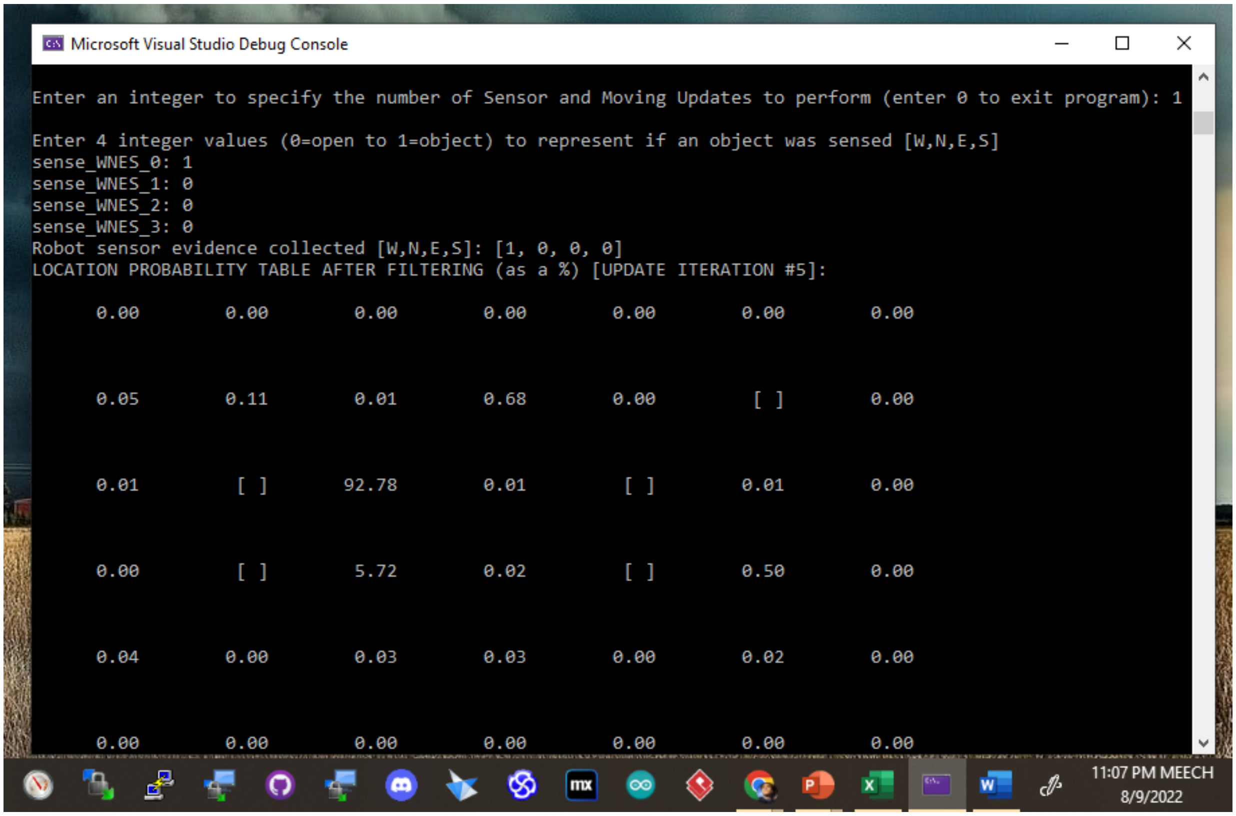
```
Microsoft Visual Studio Debug Console
Enter an integer to specify the number of Sensor and Moving Updates to perform (enter 0 to exit program): 1

Enter 4 integer values (0=open to 1=object) to represent if an object was sensed [W,N,E,S]
sense_WNES_0: 0
sense_WNES_1: 0
sense_WNES_2: 0
sense_WNES_3: 0
Robot sensor evidence collected [W,N,E,S]: [0, 0, 0, 0]
LOCATION PROBABILITY TABLE AFTER FILTERING (as a %) [UPDATE ITERATION #4]:

     0.04     0.01     0.10     0.00     0.00     0.00     0.00

     0.04     0.04    90.32     0.08     0.00     [ ]      0.00

     0.00     [ ]      5.51     1.78     [ ]      0.42     0.05

     0.03     [ ]      0.48     0.47     [ ]      0.28     0.27

     0.03     0.02     0.00     0.01     0.00     0.00     0.00

     0.00     0.00     0.00     0.00     0.00     0.00     0.00
```

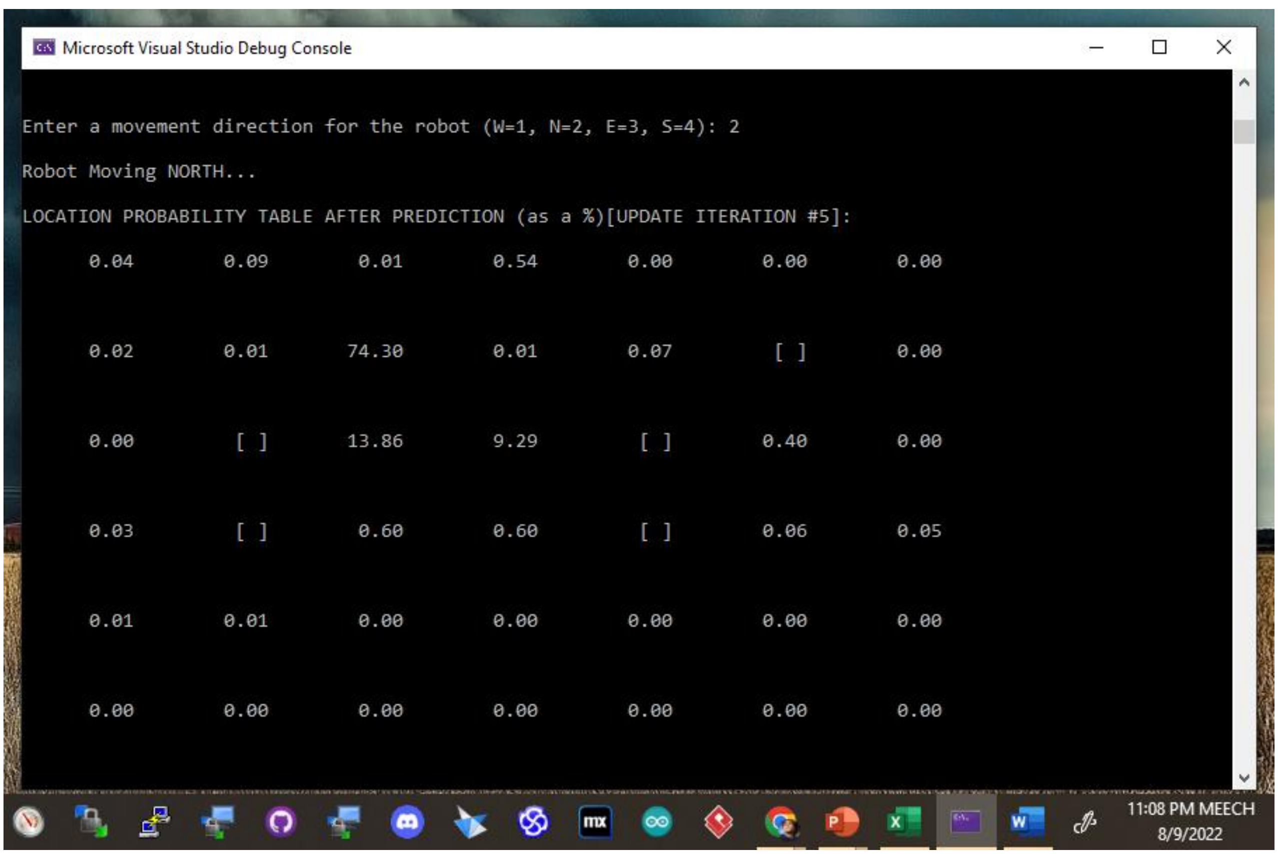



```
Microsoft Visual Studio Debug Console

Enter a movement direction for the robot (W=1, N=2, E=3, S=4): 4

Robot Moving SOUTH...

LOCATION PROBABILITY TABLE AFTER PREDICTION (as a %)[UPDATE ITERATION #4]:

     0.00     0.01     0.00     0.01     0.00     0.00     0.00

     0.04     9.07     0.09     9.03     0.01     [ ]      0.00

     0.03     [ ]     72.99     0.79     [ ]      0.05     0.05

     0.01     [ ]      4.50     1.52     [ ]      0.39     0.10

     0.03     0.00     0.39     0.37     0.00     0.23     0.22

     0.02     0.02     0.00     0.01     0.00     0.00     0.00

Enter an integer to specify the number of Sensor and Moving Updates to perform (enter 0 to exit program): 1
```

Demetrius Johnson – Program 2 – HMM Robot Localization Probability Algorithm

SENSE [1,0,0,0], MOVE N





Demetrius Johnson – Program 2 – HMM Robot Localization Probability Algorithm

SENSE [0,0,0,0], MOVE E