Robot Localization
Javascript Implementation
CIS 479
Project 2 Report

Team: Samuel Hale

Date: 8/8/23

```
Initial Probabilities
####  5.26  ####  ####  ####  ####  ####
####  5.26  5.26  5.26  ####  5.26  ####
####  ####  ####  5.26  ####  5.26  ####
####  5.26  5.26  5.26  5.26  5.26  ####
####  5.26  ####  5.26  ####  ####  ####
####  5.26  ####  5.26  5.26  5.26  ####
####  ####  ####  ####  ####  5.26  ####

Filtering: [1, 0, 1, 0]
####  2.36  ####  ####  ####  ####  ####
####  0.13  0.00  0.13  ####  2.36  ####
####  ####  ####  22.42 ####  22.42 ####
####  0.13  0.00  0.07  0.00  0.13  ####
####  22.42 ####  22.42 ####  ####  ####
####  2.36  ####  0.13  0.00  0.13  ####
####  ####  ####  ####  ####  2.36  ####

Prediction after action: N
####  2.46  ####  ####  ####  ####  ####
####  0.02  0.03  16.93 ####  19.18 ####
####  ####  ####  5.66  ####  5.7  ####
####  16.93 0.02  16.82 0.03  0.01  ####
####  7.38  ####  5.7  ####  ####  ####
####  0.59  ####  0.02  0.03  1.88  ####
####  ####  ####  ####  ####  0.59  ####
```

```
Filtering: [0, 0, 0, 0]
####  0.02  ####  ####  ####  ####  ####
####  0.00  0.00  1.07  ####  0.13  ####
####  ####  ####  0.36  ####  0.36  ####
####  1.07  0.00  96.03 0.00  0.00  ####
####  0.47  ####  0.36  ####  ####  ####
####  0.00  ####  0.00  0.00  0.12  ####
####  ####  ####  ####  ####  0.00  ####

Prediction after action: E
####  0.02  ####  ####  ####  ####  ####
####  0.00  0.00  1.02  ####  0.17  ####
####  ####  ####  14.78 ####  0.28  ####
####  0.23  0.8   0.09  72.02 0.04  ####
####  0.46  ####  9.87  ####  ####  ####
####  0.05  ####  0.04  0.00  0.11  ####
####  ####  ####  ####  ####  0.01  ####

Filtering: [0, 1, 0, 1]
####  0.00  ####  ####  ####  ####  ####
####  0.00  0.00  0.01  ####  0.00  ####
####  ####  ####  0.00  ####  0.00  ####
####  0.00  1.1   0.00  98.89 0.00  ####
####  0.00  ####  0.00  ####  ####  ####
####  0.00  ####  0.00  0.00  0.00  ####
####  ####  ####  ####  ####  0.00  ####
```

```
Prediction after action: E
####  0.00  ####  ####  ####  ####  ####
####  0.00  0.00  0.01  ####  0.00  ####
####  ####  ####  0.00  ####  0.00  ####
####  0.00  0.17  0.83  14.83 74.17 ####
####  0.00  ####  0.00  ####  ####  ####
####  0.00  ####  0.00  0.00  0.00  ####
####  ####  ####  ####  ####  0.00  ####

Filtering: [0, 0, 1, 1]
####  0.00  ####  ####  ####  ####  ####
####  0.00  0.00  0.00  ####  0.00  ####
####  ####  ####  0.00  ####  0.00  ####
####  0.00  0.00  0.00  0.12  99.88 ####
####  0.00  ####  0.00  ####  ####  ####
####  0.00  ####  0.00  0.00  0.00  ####
####  ####  ####  ####  ####  0.00  ####

Prediction after action: N
####  0.00  ####  ####  ####  ####  ####
####  0.00  0.00  0.00  ####  0.00  ####
####  ####  ####  0.00  ####  74.91 ####
####  0.00  0.00  0.02  15.07 10  ####
####  0.00  ####  0.00  ####  ####  ####
####  0.00  ####  0.00  0.00  0.00  ####
####  ####  ####  ####  ####  0.00  ####
```

```
Filtering: [1, 0, 1, 0]
####  0.00  ####  ####  ####  ####  ####
####  0.00  0.00  0.00  ####  0.00  ####
####  ####  ####  0.00  ####  99.92 ####
####  0.00  0.00  0.00  0.00  0.08  ####
####  0.00  ####  0.00  ####  ####  ####
####  0.00  ####  0.00  0.00  0.00  ####
####  ####  ####  ####  ####  0.00  ####

Programmed By Yours Truly, Samuel Hale
Current Time: 1:52:51 PM
```

SOURCE CODE

```
// import fs module
var fs = require('fs');

// Initialize a 2D array representing a maze from maze.txt. '####' are
obstacles and '????' are open spaces.
function initializeMaze(maze) {
    var mazeString = fs.readFileSync('maze.txt', 'utf8');
    var mazeArray = mazeString.split('\n');
    for (var i = 0; i < mazeArray.length; i++) {
        maze[i] = mazeArray[i].replace(/\r/g, '').split(' ');
    }
    return maze;
}

// Write a function that prints the maze as a string to the console and is
evenly spaced
function printMaze(maze) {
    var mazeString = '';
    for (var i = 0; i < maze.length; i++) {
        for (var j = 0; j < maze[0].length; j++) {
            // print 0's as 0.00
            if (maze[i][j] == 0) {
                maze[i][j] = '0.00';
```

```
            }
            if (maze[i][j].toString().length > 4) {
                mazeString += maze[i][j] + ' ';
            }
            else mazeString += maze[i][j] + '  ';
        }
        mazeString += '\n';
    }
    console.log(mazeString);
}

// overload print maze to take in a string to print before maze printing
function printMaze(maze, string) {
    var mazeString = string + '\n';
    for (var i = 0; i < maze.length; i++) {
        for (var j = 0; j < maze[0].length; j++) {
            // print 0's as 0.00
            if (maze[i][j] == 0) {
                maze[i][j] = '0.00';
            }
            if (maze[i][j].toString().length > 4) {
                mazeString += maze[i][j] + ' ';
            }
            else mazeString += maze[i][j] + '  ';
        }
        mazeString += '\n';
    }
    console.log(mazeString);
}

// Initialize all '????' parts of the maze to be 100 / # of '????' squares
function initializeProbabilities(maze) {
    var totalSpaces = 0;
    for (var i = 0; i < maze.length; i++) {
        for (var j = 0; j < maze[0].length; j++) {
            if (maze[i][j] == '????') {
                totalSpaces++;
            }
        }
    }
```

```
    var probability = 100 / totalSpaces;
    // limit probability to 2 decimal places
    probability = Math.round(probability * 100) / 100;
    for (var i = 0; i < maze.length; i++) {
        for (var j = 0; j < maze[0].length; j++) {
            if (maze[i][j] == '????') {
                maze[i][j] = probability;
            }
        }
    }
    return maze;
}

// Add in Filtering Algorithm
function calculateProbability(maze, e, x, y) {
    // e is the surrounding evidence (i.e. [1, 0, 1, 0])
    // e[0] is west, e[1] is north, e[2] is east, e[3] is south
    // 1 is an obstacle, or an out of bounds index. 0 is an open space.
    // Correctly identifying an obstacle = 0.9, correctly identifying an
open space = 0.95
    // Incorrectly identifying an obstacle = 0.05, incorrectly identifying
an open space = 0.1
    var correctObstacle = 0.9;
    var correctOpenSpace = 0.95;
    var incorrectObstacle = 0.05;
    var incorrectOpenSpace = 0.1;
    // Probability = P(e[0] | x-1) * P(e[1] | x+1) * P(e[2] | y-1) *
P(e[3] | y+1)
    var probability = 1;
    for (var i = 0; i < e.length; i++) {
        if (e[i] == 1) {
            if (i == 0) {
                // west
                if (y > 0 && maze[x][y-1] == '####') {
                    probability *= correctObstacle;
                } else {
                    probability *= incorrectObstacle;
                }
            } else if (i == 1) {
                // north
```

```
            if (x > 0 && maze[x-1][y] == '####') {
                probability *= correctObstacle;
            } else {
                probability *= incorrectObstacle;
            }
        } else if (i == 2) {
            // east
            if (y < maze.length - 1 && maze[x][y+1] == '####') {
                probability *= correctObstacle;
            } else {
                probability *= incorrectObstacle;
            }
        } else if (i == 3) {
            // south
            if (x < maze[0].length - 1 && maze[x+1][y] == '####') {
                probability *= correctObstacle;
            } else {
                probability *= incorrectObstacle;
            }
        }
    } else {
        if (i == 0) {
            // west
            if (y > 0 && maze[x][y-1] != '####') {
                probability *= correctOpenSpace;
            } else {
                probability *= incorrectOpenSpace;
            }
        } else if (i == 1) {
            // north
            if (x > 0 && maze[x-1][y] != '####') {
                probability *= correctOpenSpace;
            } else {
                probability *= incorrectOpenSpace;
            }
        } else if (i == 2) {
            // east
            if (y < maze.length - 1 && maze[x][y+1] != '####') {
                probability *= correctOpenSpace;
            } else {
```

```
                    probability *= incorrectOpenSpace;
                }
            } else if (i == 3) {
                // south
                if (x < maze[0].length - 1 && maze[x+1][y] != '####') {
                    probability *= correctOpenSpace;
                } else {
                    probability *= incorrectOpenSpace;
                }
            }
        }
    }

    return probability;
}

function filtering(maze, e) {
    // e is the surrounding evidence (i.e. [1, 0, 1, 0])
    // var probabilityObject = {x: i, y: j, probability: 0};
    var probabilities = [];
    var probabilitySum = 0;
    // Loop through maze and sense at every open spot/ calculate that
probability
    for (var i = 0; i < maze.length; i++) {
        for (var j = 0; j < maze[0].length; j++) {
            if (maze[i][j] != '####') {
                let tempProbOb = {x: i, y: j, probability: maze[i][j] *=
calculateProbability(maze, e, i, j)};
                probabilities.push(tempProbOb);
                probabilitySum += tempProbOb.probability;
            }
        }
    }
    // Normalize the probabilities by dividing by the sum of all
probabilities
    for (var i = 0; i < probabilities.length; i++) {
        probabilities[i].probability /= probabilitySum;
    }
    // Update the maze with the new probabilities
    for (var i = 0; i < probabilities.length; i++) {
```

```
        // convert probability to percentages
        probabilities[i].probability *= 100;
        // round probability to 2 decimal places
        probabilities[i].probability =
Math.round(probabilities[i].probability * 100) / 100;
        maze[probabilities[i].x][probabilities[i].y] =
probabilities[i].probability;
    }
    return maze;
}

function calculateDrift(maze, d, x, y) {
    // d is the direction to move (i.e. W, N, E)
    // There is a 0.75 chance of moving in the correct
direction(straight), a 0.15 chance he blows left
    // and a 0.15 chance he blows right. If he blows left or right into an
obstacle he stays in the same spot.
    // If he blows left or right into an open space he moves into that
space.
    var relativeStraight = 0.75;
    var leftDirection = 0.15;
    var rightDirection = 0.1;

    // Outcomes will be an array of objects with the following properties:
    // x, y, probability
    // x and y are the coordinates of the outcome, probability is the
probability of that outcome
    var outcomes = [];

    // If the direction is west
    if (d == 'W') {
        if (y>0 && maze[x][y-1] != '####') {
            // If the spot to the left is an open space
            // Add the spot to the left to the outcomes array
            outcomes.push({x: x, y: y-1, probability:
maze[x][y]*relativeStraight});
        }
        else {
            // Add the current spot to the outcomes array
```

```
            outcomes.push({x: x, y: y, probability:
maze[x][y]*relativeStraight});
        }
        // Calculate relative drift direction probabilities
        // Calculate left drift direction from robot facing west
        if (x>0 && maze[x+1][y] != '####') {
            outcomes.push({x: x+1, y: y, probability:
maze[x][y]*leftDirection});
        }
        else {
            outcomes.push({x: x, y: y, probability:
maze[x][y]*leftDirection});
        }
        // Calculate right drift direction from robot facing west
        if (x>0 && maze[x-1][y] != '####') {
            outcomes.push({x: x-1, y: y, probability:
maze[x][y]*rightDirection});
        }
        else {
            outcomes.push({x: x, y: y, probability:
maze[x][y]*rightDirection});
        }
    }
    // If the direction is north
    else if (d == 'N') {
        if (x>0 && maze[x-1][y] != '####') {
            // If the spot to the left is an open space
            // Add the spot to the left to the outcomes array
            outcomes.push({x: x-1, y: y, probability:
maze[x][y]*relativeStraight});
        }
        else {
            // Add the current spot to the outcomes array
            outcomes.push({x: x, y: y, probability:
maze[x][y]*relativeStraight});
        }
        // Calculate relative drift direction probabilities
        // Calculate left drift direction from robot facing north
        if (y>0 && maze[x][y-1] != '####') {
```

```
            outcomes.push({x: x, y: y-1, probability:
maze[x][y]*leftDirection});
        }
        else {
            outcomes.push({x: x, y: y, probability:
maze[x][y]*leftDirection});
        }
        // Calculate right drift direction from robot facing north
        if (y<maze[0].length-1 && maze[x][y+1] != '####') {
            outcomes.push({x: x, y: y+1, probability:
maze[x][y]*rightDirection});
        }
        else {
            outcomes.push({x: x, y: y, probability:
maze[x][y]*rightDirection});
        }
    }
    // If the direction is east
    else if (d == 'E') {
        if (y<maze[0].length-1 && maze[x][y+1] != '####') {
            // If the spot to the left is an open space
            // Add the spot to the left to the outcomes array
            outcomes.push({x: x, y: y+1, probability:
maze[x][y]*relativeStraight});
        }
        else {
            // Add the current spot to the outcomes array
            outcomes.push({x: x, y: y, probability:
maze[x][y]*relativeStraight});
        }
        // Calculate relative drift direction probabilities
        // Calculate left drift direction from robot facing east
        if (x>0 && maze[x-1][y] != '####') {
            outcomes.push({x: x-1, y: y, probability:
maze[x][y]*leftDirection});
        }
        else {
            outcomes.push({x: x, y: y, probability:
maze[x][y]*leftDirection});
        }
```

```
        // Calculate right drift direction from robot facing east
        if (x<maze.length-1 && maze[x+1][y] != '####') {
            outcomes.push({x: x+1, y: y, probability:
maze[x][y]*rightDirection});
        }
    }
    return outcomes;
}

// Add in Prediction Algorithm
function move(maze, d) {
    var outcomes = [];

    // Loop through maze
    for (var i = 0; i < maze.length; i++) {
        for (var j = 0; j < maze[0].length; j++) {
            // If the current spot is not an obstacle
            if (maze[i][j] != '####') {
                // Append the entries of the returned array from
calculateDrift to outcomes
                let temp = calculateDrift(maze, d, i, j);
                outcomes.push.apply(outcomes, temp);
            }
        }
    }

    // Loop through the maze and everytime you find an open space
    // Add all of the outcome probabilities with the same i,j coordinates
    // And initialize that to maze at i,j
    for (var i = 0; i < maze.length; i++) {
        for (var j = 0; j < maze[0].length; j++) {
            if (maze[i][j] != '####') {
                var sum = 0;
                for (var k = 0; k < outcomes.length; k++) {
                    if (outcomes[k].x == i && outcomes[k].y == j) {
                        sum += outcomes[k].probability;
                    }
                }
                // round sum to 2 decimal places
                sum = Math.round(sum*100)/100;
```

```
                maze[i][j] = sum;
            }
        }
    }
    return maze;
}


maze = [];
maze = initializeMaze(maze);
maze = initializeProbabilities(maze);
/* 1. Sensing: [1, 0, 1, 0]
2. Moving: N
3. Sensing: [0, 0, 0, 0]
4. Moving: E
5. Sensing: [0, 1, 0, 1]
6. Moving: E
7. Sensing: [0, 0, 1, 1]
8. Moving: N
9. Sensing: [1, 0, 1, 0] */
printMaze(maze, "Initial Probabilities");
maze = filtering(maze, [1, 0, 1, 0]);
printMaze(maze, `Filtering: [1, 0, 1, 0]`);
maze = move(maze, 'N');
printMaze(maze, `Prediction after action: N`);
maze = filtering(maze, [0, 0, 0, 0]);
printMaze(maze, `Filtering: [0, 0, 0, 0]`);
maze = move(maze, 'E');
printMaze(maze, `Prediction after action: E`);
maze = filtering(maze, [0, 1, 0, 1]);
printMaze(maze, `Filtering: [0, 1, 0, 1]`);
maze = move(maze, 'E');
printMaze(maze, `Prediction after action: E`);
maze = filtering(maze, [0, 0, 1, 1]);
printMaze(maze, `Filtering: [0, 0, 1, 1]`);
maze = move(maze, 'N');
printMaze(maze, `Prediction after action: N`);
maze = filtering(maze, [1, 0, 1, 0]);
printMaze(maze, `Filtering: [1, 0, 1, 0]`);
console.log("Programmed By Yours Truly, Samuel Hale");
```

```
//console.log the current time
console.log("Current Time: " + new Date().toLocaleTimeString());
```

(Look for the corresponding highlight color in the code)

Transitional Probability

For this part I loop the maze. When I find an open space I send the direction and coordinates in the maze array to another function. In this function it will assess the Probability of going straight, drifting left and drifting right. From there it builds objects that contain probability and i,j coordinates (So the probability of the robot ending up there based on drift probability * current state probability). It appends that object to an array and returns it back to the function loop. After the loop in the function we have an array with all of the transitional probabilities.

Evidence CP

For this part I loop through the open spaces. At each open space I build a temp object with the indices and probability that the robot is in that open space. The probability that the robot is in the open space is current state probability * the evidence conditional probability which is calculated in calculateProbability. calculateProbability takes in the evidence and current indicies. It will evaluate each direction and keep a running multiplication of 0.9 (for correct obstacle), 0.95 (for correct open space), 0.1 (for incorrect open space), 0.05 (for incorrect obstacle). It returns that running multiplication.

Filtering

For this part we loop through the array of objects keeping a running sum of all the probabilities. From there we will loop through objects array again and normalize it by doing probability / sum of all probabilities. It will do some value standardization for printing purposes and initialize the maze at the objects indices to the newly normalized probability.

Prediction

For this part I loop through the maze. Once I find an open space I loop through our outcomes array while keeping a running sum of probabilities. If the current indices in the maze match indices from the current outcomes than we append the probability to the running sum. After outcomes has fully been looped through the probability of the robot being at 1 open space in the maze will be calculated and the maze will be updated. It will loop through the rest of the open spaces in the maze and repeat.

All in all this was an amazing project that allowed me to program robot localization in my own way. It helped me to understand exactly how it is that a robot can localize and find itself in a maze. Programatically speaking I could definitely improve the efficiency by keeping track of the open spaces from the first maze initialization so that I don't have to loop through the maze so much. Doing that could make my code more efficient when given larger mazes.