Introduction

This report goes over Trilateration and Wall-based Localization which are used to estimate robot position and cell location in each map. The robot uses its onboard cameras and sensors to scan its surroundings and compute its pose and navigate a maze while updating this information for each new cell. The following localization methods are described in the sections as follows:

Task 1 – Trilateration-Based Localization

For this task, the trilateration algorithm works with the calculation from 3 different cylinders. The center x-y coordinates and the distance from the robot to each cylinder are passed as inputs to a function that returns the x, y coordinates of the robot.

As seen by Figure 1, the x, y coordinates of the robot are computed at the intersection of the three circles.

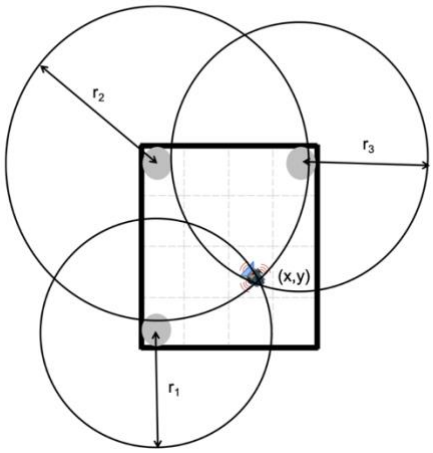

Figure 1: Visual representation of the trilateration algorithm.

The actual mathematical equation for trilateration is presented below:

$$A = 2(x_2 - x_1)$$
$$B = 2(y_2 - y_1)$$
$$C = {r_1}^2 - {r_2}^2 - {x_1}^2 + {x_2}^2 - {y_1}^2 + {y_2}^2$$
$$D = 2(x_3 - x_2)$$
$$E = 2(y_3 - y_2)$$
$$F = {r_2}^2 - {r_3}^2 - {x_2}^2 + {x_3}^2 - {y_2}^2 + {y_3}^2$$

$$x = \frac{(C \times E) - (F \times B)}{(E \times A) - (B \times D)}$$

$$y = \frac{(C \times D) - (A \times F)}{(B \times D) - (A \times E)}$$

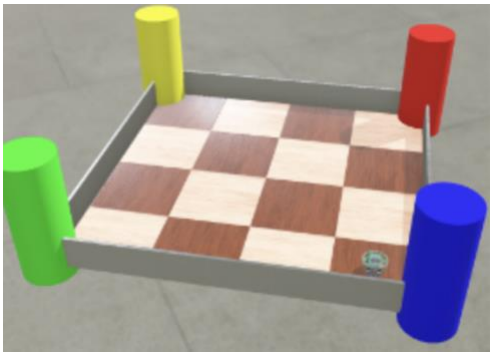

Figure 2: Task 1 map which shows a grid of cells.

To calculate the values that get used in the above function, the robot must rotate to obtain the distance from it to the center of each cylinder.

A) This is accomplished using lm.getPosition()[0]. Please see Figure 2 for reference.

B) To distinguish between different cylinder colors, the function lm.getColors()[] is used. Since trilateration only requires 3 cylinders, I have chosen to ignore the yellow cylinder in my program.

With the calculated x and y coordinates of the robot via trilateration, I was able to determine the exact cell the robot is located in. By passing these x and y coordinates into an array that maps each cell, I'm able to obtain the index (or cell) in the array that corresponds to the cell my robot is located in.

A Boolean variable is used to mark cells which are visited, and another array is used to visually represent this with an 'X'. Additionally, the program finds out the next unvisited cell once the robot finds out it's in a cell that it has already visited. By combining random movement from cell to cell then determining the next empty cell to move to, the robot essentially marks every cell as visited. However, this isn't the most efficient method to navigate the maze as it takes on average 3-4 minutes to complete.

Task 2 – Wall-Based Localization

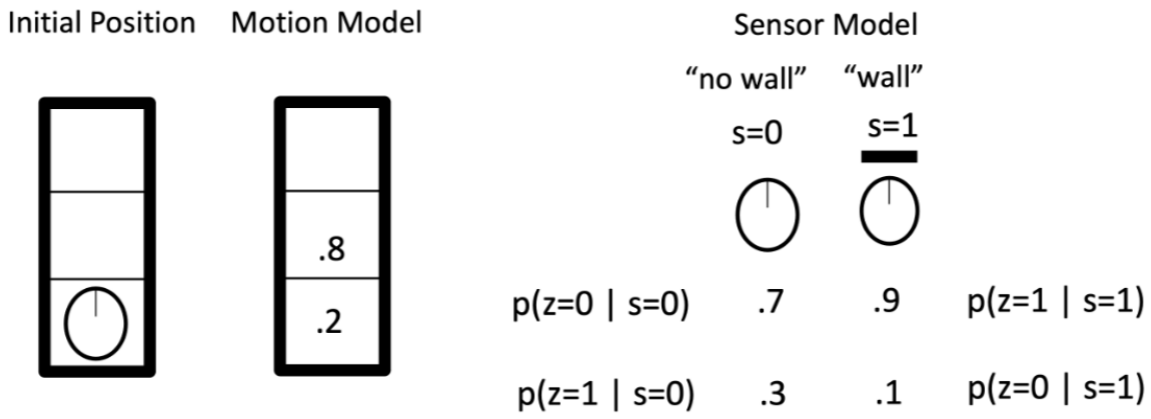

Figure 3: Motion and Sensor Model for Probability Calculations

For task 2, the robot is required to navigate through a series of walls while using them in probability calculations to determine *stay* and *move* probabilities.

Based on the motion model (see Figure 3), 0.8 represents the probability of a forward *move* motion and 0.2 if the robot ended up to *stay* in place. From the sensor model, 0 means "no wall" and 1 means "wall". S is the given value and Z is the actual calculated value from the robot LiDAR sensors.

For example, if the robot calculated (Z) "no wall" given (S) that there was "no wall" → p(z=0 | s=0) this results in a probability of 0.7.

To calculate probability for *move,* it is determined by the next cell's wall configuration.
→ 0.8 * p(S (west) | Z) * p(S (north) | Z) * p(S (east) | Z) * p(S (south) | Z)

To calculate probability for *stay*, it is determined by the current cell wall configuration.
→ 0.2 * p(S (west) | Z) * p(S (north) | Z) * p(S (east) | Z) * p(S (south) | Z)

While printing *move* and *stay* probabilities, the robot must also successfully visit every cell like task 1.

C) The wall configuration is represented via a maze array in the program which initialized with 'W' (wall exists) and 'O' (no wall exists).
D) The addition of sensor noise has minimal effect on the state probabilities as the robot still navigates the maze with high accuracy and detects all walls as normal.

Conclusion

Overall, this lab presented several challenges, most for which I was able to find new solutions and fix. For task 1, after trilateration, my robot gets set to a random N, S, E, W heading which causes it to visit cells in random order. This resulted in a very inefficient way to visit cells due to constant repetition of the same cells and often, failure to visit every cell. This required me to restructure my program to find unvisited cells. Now, my program for task 2 is a lot more efficient by visiting all cells with way less repetition of visiting the same cells. However, it still implements some random elements and can be made more efficient by removing the randomness altogether. For task 2, I had some difficulties understanding the probabilities, however, was able to calculate, implement, and print them with some help from classmates. The robot moves effortlessly through each maze and detects walls and avoids them successfully.