

**ENCS4380—INTERFACING TECHNIQUES.**

**Prepared By:**

**Saleh Khatib – 1200991.**

**Adham dar almasri – 1201199.**

**Section 1.**

**Dr:Wasel Ghanem.**

**Date: 4/6/2024.**

# **Left-Roll**

In the left-roll algorithm for a Micromouse, the mouse starts by facing the initial direction, typically North, with all sensors and encoders calibrated. The mouse moves forward one cell if the left-side sensor detects a wall. If no wall is detected on the left, the mouse turns 90 degrees left and proceeds forward. In situations where forward movement is blocked, the mouse turns right until it finds an open path and then continues moving forward. This process is repeated, enabling the Micromouse to navigate through the maze efficiently by always prioritizing the left-hand path.

**Code link:** <https://drive.google.com/drive/folders/1DU5G27EeFsHMhLqDiVPW7oWwRZnY1cTj?usp=drive_link>

**Video link:**

<https://drive.google.com/file/d/1lH2MW9jF2GXOzS_76XC-hsvV3RN9Xd_3/view?usp=sharing>

# **Flood-Fill**

Envision pouring water into the maze's final destination—the four cells in the center, encircled by seven walls. The cell directly outside the destination cells will receive the first flow of water. and finally, to nearby cells that are easily accessible. Similar to how water flows through the maze, it will eventually arrive at the mouse's starting location.   
  
We design a second 16 x 16 maze, which we'll refer to as the Flood array. We allocated zero to each of the four destination cells, one to the cell that the destination cells can access right away, and so on, taking inspiration from the water pouring example. The cells with the highest number are those to which the water flows last.

If you place your mouse anywhere in the maze, and ask it to travel to the cell with the value 1 less than the value of the cell it is in, the mouse is guaranteed to eventually make it to the destination in the path will minimum number of cells.

**The traversal**

But since your mouse isn’t aware of the walls configuration when it is first kept on the maze, first it assumes that there are no walls at all. (Except the walls at the maze boundaries).

As the mouse continues moves through the maze,

1. Finds the values of its neighboring cells (from the flood array)
2. Travels to the neighboring cell with the least value.
3. Detects the walls to its left, right and the front
4. Updates the newly found walls in the **maze array**
5. Perform the flood fill for the entire **flood array**
6. Back to step 1, and continue until the robot moves to the desired position.

Observe the flood array when the robot is halfway to the destination. This is because the mouse has discovered the walls in some cells, but not in others.

All you need to do is execute the floodfill from the cell you want to send the mouse to, if you want it to go somewhere other than one of the middle four. i.e., begin by setting the cell you want the mouse to go to zero.

**The fast run**

Once you decide that the mouse has discovered enough cells to find an optimum path, you can bring the mouse back to the starting square, and do the fast run. In the process, the mouse:

1. Finds the values of its neighboring cells (from the flood array)
2. Travels to the neighboring cell with the value 1 less than the present cell.
3. Back to step 1, and continue until the robot moves to the desired position.

During the fast run, we don’t need to update the **maze array**or the **flood array** as the mouse will only be moving to the cells that are already discovered.

**Code link:**

<https://drive.google.com/drive/folders/1KDGReR2rf0oGVO1LAsOAC-YrUCQDilwz?usp=sharing>

**Video link:**

<https://drive.google.com/file/d/1lH2MW9jF2GXOzS_76XC-hsvV3RN9Xd_3/view?usp=sharing>