Table of Contents

Contents

1 Progress Report

1.1 1st Iteration

Should be achieved by:

1.2 2nd Iteration

Should be achieved by:

1.3 3rd Iteration

Should be achieved by:

2 Mechanical Design of MazeBot

2.1 First Iteration

Goals:

- Accurate movement

Mechanical Drawing

Observations & Measurements

Structural Integrity of the Drive System
In order for more accurate movement, we added a high gear ratio. However, since we added the high gear ratio, we are unable to find space to properly secure the left and right drive wheels. When testing, we found one wheel to slipped forward and the other to slipped back when turning which defeats the accuracy of the encoder. Because of this, we are unable to meet our goal of accurate movement.

Robot is too large.
Since the brick is upright, it is top heavy. We needed two rods in the back and one metal ball in the front in order to balance the robot. The additions of the two rods and one metal ball remove the spacial advantage of having the robot's brick be upright. Even though the dimensions of the robot are within the size of one square, it leaves very little room for error. As such, the robot begins to run into walls after the 3rd turn.

Conclusion
In conclusion, we have decided for our next iteration to have the robot's brick be flat on the ground in order to ensure that we have enough room to properly secure our drive system. By properly securing the drive system we hope to not have to readjust every time we enter a cell.

3 Software Design of MazeBot

Our main goal with the software of the mazebot was to create program solved the problem simply and was easy to build upon. Furthermore, we wanted our software to have very few constants that we would need to tested for. For example, in order to move forward one cell, we would need to give the following function the degrees to move each of our drive motors:

```
setMotorTarget(leftMotor, degrees, 75);
```

The degrees needed to move one cell forward could be achieved by constantly testing different values of degrees to achieve the movement to the new cell. However, we chose to calculate the exact degrees that the robot's drive motors would need to move in order to move exactly one cell forward. This approach in contrast to the former has two advantages:

1. It allows us to isolate any problems with moving accurately to a mechanical problem.
2. We would not have an accumulation of error because of us testing incorrectly.

Therefore, we chose to mathematically calculate the degrees that we needed to move the motors rather than testing.

A sketch of the derivation of how many degrees to move forward is shown below:
Therefore:

```
degrees = (SIZE_OF_ONE_CELL / CIRCUMFERENCE_OF_WHEEL) * DRIVE_GEAR_RATIO * ONE_ROTATION
```

A similar derivation exists for turning the robot 90 degrees:

3.1 Variables Used to Define the Position of the Robot in the Maze and the Size of the Maze

- Two constant that represent the initial position of the robot in the maze were declared. This will be entered when we begin our demo.

  ```
  int const START_ROW = ;
  int const START_COL = ;
  ```

- Two constants that represents the target position in the maze were declared. This will be entered when we begin our demo.

  ```
  int const END_ROW = ;
  int const END_COL = ;
  ```

- Two variables that represents the current position of the robot in the maze were declared. This is initialized as the starting position.

  ```
  int currentRow = START_ROW;
  int currentCol = START_COL;
  ```

- An array that represents the orientation that the bot has as it enters each cell were defined. The size of the array is four times larger than the multiple of maze width and maze height because the maximum amount of times that the robot can go into each cell is four times.

  ```
  int entered[MAZE_WIDTH*MAZE_HEIGHT*4];
  int lastEnteredIdx = 0;
  ```

- A constant that represents the dimension of a single cell was defined

  ```
  float const SIZE_OF_ONE_CELL = 22.5425; // in cm
  ```

- Four constants that represent the size of the maze were declared

  ```
  int const MAZE_WIDTH = 4;
  int const MAZE_HEIGHT = 6;
  int const LAST_MAZE_HEIGHT_INDEX = MAZE_HEIGHT - 1;
  int const LAST_MAZE_WIDTH_INDEX = MAZE_WIDTH - 1;
  ```

3.2 Constants and Variables Used for Representation of Directions

- The four constants that represent each directions were declared

```
#define NORTH 0
#define EAST 1
#define SOUTH 2
#define WEST 3
```

- Structure name cell was declared and it has five parameters. This track where the walls are and whether we have visited the cell.

```
typedef struct{
   int NWall;
   int SWall;
   int EWall;
   int WWall;
   char Visited;
}cell;
```

- Variable name "Maze" with the data type cell was declared . This data type includes five parameters: North wall, South wall, East wall, West wall, and Visited (used to tell if the robot has visited the cell or not) as discussed above.

```
cell Maze[MAZE_HEIGHT][MAZE_WIDTH];
```

- Global variable that represents current direction of the robot was declared. This is initialized as north as this is the orientation of the robot when it first enters the maze.

```
int direction = NORTH;
```

3.3 Constants Used for Display

- Two constants that represent the size of the screen width and height were defined

```
#define SCREEN_HEIGHT 127
#define SCREEN_WIDTH 177
```

- Two constants that represent the each cell's size in the screen were defined

```
#define CELL_HEIGHT (SCREEN_HEIGHT / MAZE_HEIGHT)
#define CELL_WIDTH (SCREEN_WIDTH / MAZE_WIDTH)
```

- Two constants are defined which represents the robot's position in each cell in the screen

```
#define CELL_HEIGHT_MIDDLE (CELL_HEIGHT / 2)
#define CELL_WIDTH_MIDDLE (CELL_WIDTH /2)
```

3.4 Constants Used for Moving Mechanism

- The motor only takes in integer values which gets rid of all the decimal values. Moreover, when the value of encoder takes value calulated as the data type int, they decrease the value. Furthermore, this causes the problem for robot not to move in exact distance required. Therefore, three constants were declared which are added to the encoder input values.

```
float const UNCERTAINTY_STRAIGHT = 23;
float const UNCERTAINTY_ROT = 28;
float const UNCERTAINTY_READJUST = 35;
```

- Each speed of the motors were defined with constants for simplification of the code.

```
int const FORWARD = -100;
int const BACKWARD = -FORWARD;
```

- Encoder input constants were declared

```
float const ONE_ROTATION = 360 + UNCERTAINTY_STRAIGHT;
float const QUARTER_ROTATION = 180 + UNCERTAINTY_ROT;
float const DRIVE_GEAR_RATIO = 5;
float const DIAMETER_OF_WHEEL = 5.5; // in cm
float const CIRCUMFERENCE_OF_WHEEL = PI * DIAMETER_OF_WHEEL;
```

- Times that will be used for readjust were defined. Timing algorithm was used because the flat surface at the front of the robot adjusts itself as it pushes the wall for small amount of time

```
int const MILISECS_TO_DRIVE_INTO_WALL = 1100;
```

- A constant that represents how often the robot has to readjust its direction were defined. A variable that increases every time the robot goes into new cells to count for readjust.

```
int const CELLS_TO_READJUST_AFTER = 3;
int timesForwardWithoutReadjust = 0;
```

3.5 Constants Used for Representation of Wall

- A constant which represent the maximum distance possible between the robot and the wall. Maximum distance was defined to prevent robot to detect wall in the other cell.

```
float const DIST_BETWEEN_BOT_AND_WALL = 7.6;
```

- Three constants were defined to distinguish if the existence of walls

```
#define NOT_PRESENT 0
#define PRESENT 1
#define UNKNOWN 2
```

3.6 Constants Used for Beeping Mechanism

- A constant which represent the time and the frequency of the beep when the robot found the target

```
int const MILI_TO_BEEP_FOR = 200;
int const FREQUENCY = 300;
```

3.7 Displaying Function

- Function for displaying current informations of the robot and maze on the screen

  `void drawInfo(int direction);`

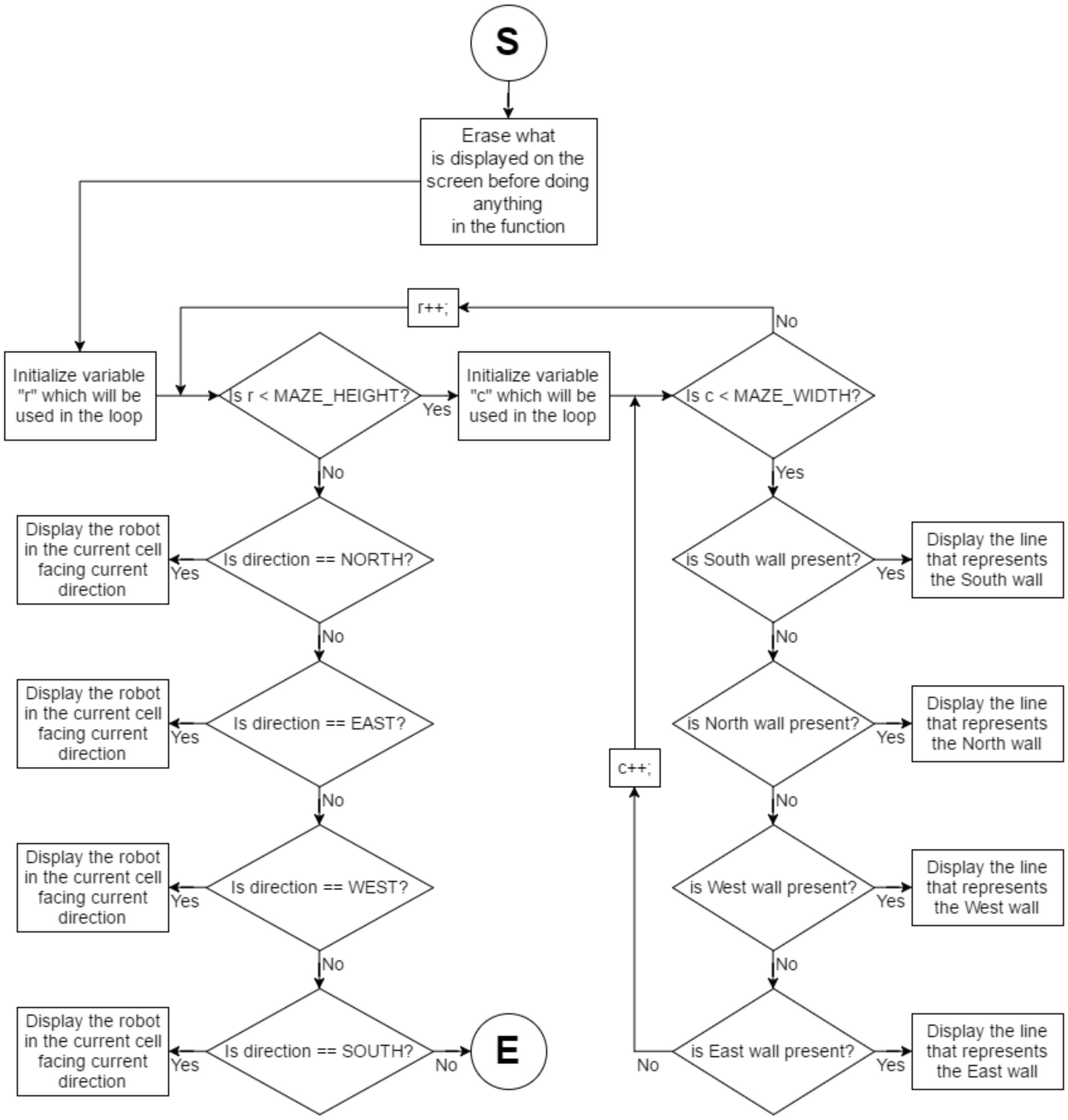

S
Erase what is displayed on the screen before doing anything in the function
r++;
No
Initialize variable "r" which will be used in the loop
Is r < MAZE_HEIGHT?
Yes
Initialize variable "c" which will be used in the loop
Is c < MAZE_WIDTH?
No
Yes
Display the robot in the current cell facing current direction
Is direction == NORTH?
Yes
No
Display the robot in the current cell facing current direction
Is direction == EAST?
Yes
No
Display the robot in the current cell facing current direction
Is direction == WEST?
Yes
No
Display the robot in the current cell facing current direction
Is direction == SOUTH?
Yes
No
E
is South wall present?
Yes
Display the line that represents the South wall
No
is North wall present?
Yes
Display the line that represents the North wall
c++;
No
is West wall present?
Yes
Display the line that represents the West wall
No
No
is East wall present?
Yes
Display the line that represents the East wall

Figure 1: Flow Chart for Displaying Function

- The local variable direction is passed into the function but it does not return any variable
- Global variables and constants used are

```
MAZE_WIDTH
MAZE_HEIGHT
CELL_WIDTH
CELL_HEIGHT
CELL_WIDTH_MIDDLE
CELL_HEIGHT_MIDDLE
```

3.8 Moving Forward Function

- Function to draw current informations of the robot on the screen

  `void goFwdCell(int direction);`

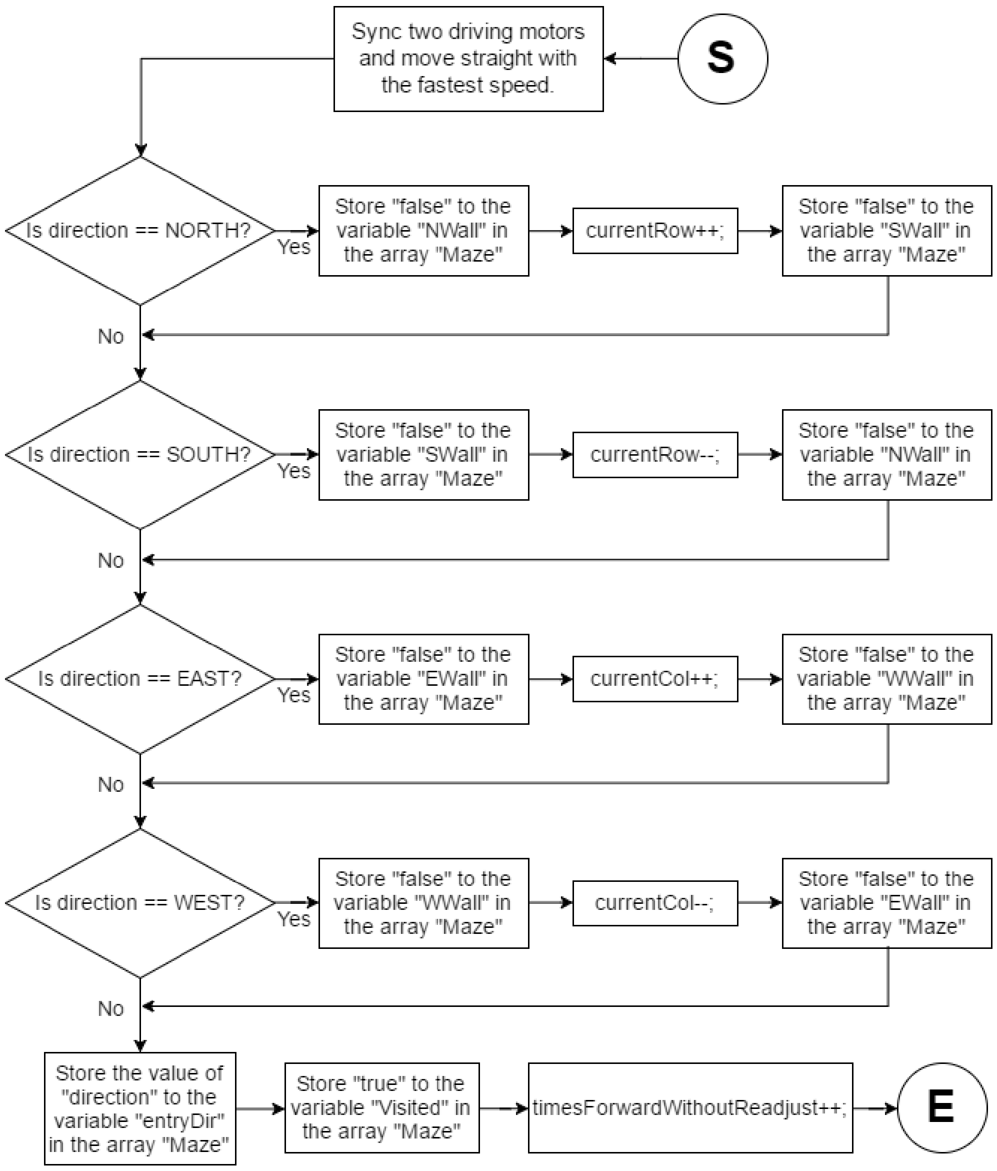

S
Sync two driving motors and move straight with the fastest speed.
Is direction == NORTH?
Yes
Store "false" to the variable "NWall" in the array "Maze"
currentRow++;
Store "false" to the variable "SWall" in the array "Maze"
No
Is direction == SOUTH?
Yes
Store "false" to the variable "SWall" in the array "Maze"
currentRow--;
Store "false" to the variable "NWall" in the array "Maze"
No
Is direction == EAST?
Yes
Store "false" to the variable "EWall" in the array "Maze"
currentCol++;
Store "false" to the variable "WWall" in the array "Maze"
No
Is direction == WEST?
Yes
Store "false" to the variable "WWall" in the array "Maze"
currentCol--;
Store "false" to the variable "EWall" in the array "Maze"
No
Store the value of "direction" to the variable "entryDir" in the array "Maze"
Store "true" to the variable "Visited" in the array "Maze"
timesForwardWithoutReadjust++;
E

Figure 2: Flow Chart for Moving Forward

  - The local variable direction is passed into the function but it does not return any variable
  - Global variables and constants used are

```
SIZE_OF_ONE_CELL
CIRCUMFERENCE_OF_WHEEL
DRIVE_GEAR_RATIO
ONE_ROTATION
FORWARD
timesForwardWithoutReadjust
```

3.9 Turning Functions

- Function for Turning right

  ```
  int Turn90CW(int direction);
  ```

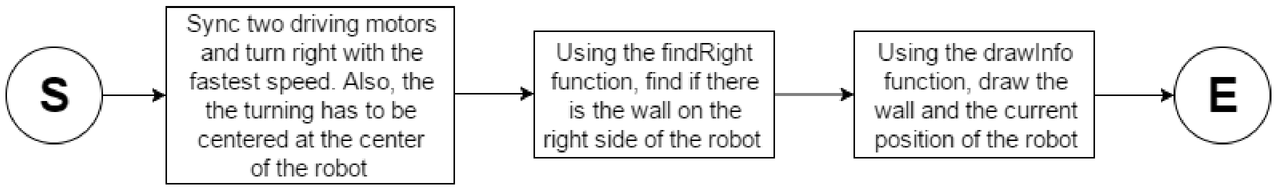

S
Sync two driving motors and turn right with the fastest speed. Also, the the turning has to be centered at the center of the robot
Using the findRight function, find if there is the wall on the right side of the robot
Using the drawInfo function, draw the wall and the current position of the robot
E

Figure 3: Flow Chart for Turning Right

  - The local variable direction is passed into the function and it returns the same variable direction
  - Global variables and constants used are

    ```
    QUARTER_ROTATION;
    DRIVE_GEAR_RATIO;
    FORWARD;
    ```

  - This function calls in other functions

    ```
    int findRight(int direction);
    int drawInfo(int direction);
    ```

- Function for Turning left

  ```
  Turn90CW(int direction);
  ```

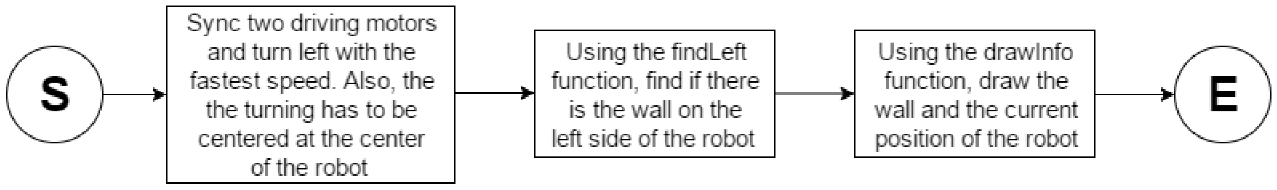

S
Sync two driving motors and turn left with the fastest speed. Also, the the turning has to be centered at the center of the robot
Using the findLeft function, find if there is the wall on the left side of the robot
Using the drawInfo function, draw the wall and the current position of the robot
E

Figure 4: Flow Chart for Turning Left

  - The local variable direction is passes into the function and it returns the same variable direction
  - Global variables and constants used are

    ```
    QUARTER_ROTATION;
    DRIVE_GEAR_RATIO;
    FORWARD;
    ```

  - This function calls in other functions

    ```
    int findLeft(int direction);
    int drawInfo(int direction);
    ```

3.10 Wall Detecting Function

- Function that detects wall with the sensor

```
int thereIsWall();
```

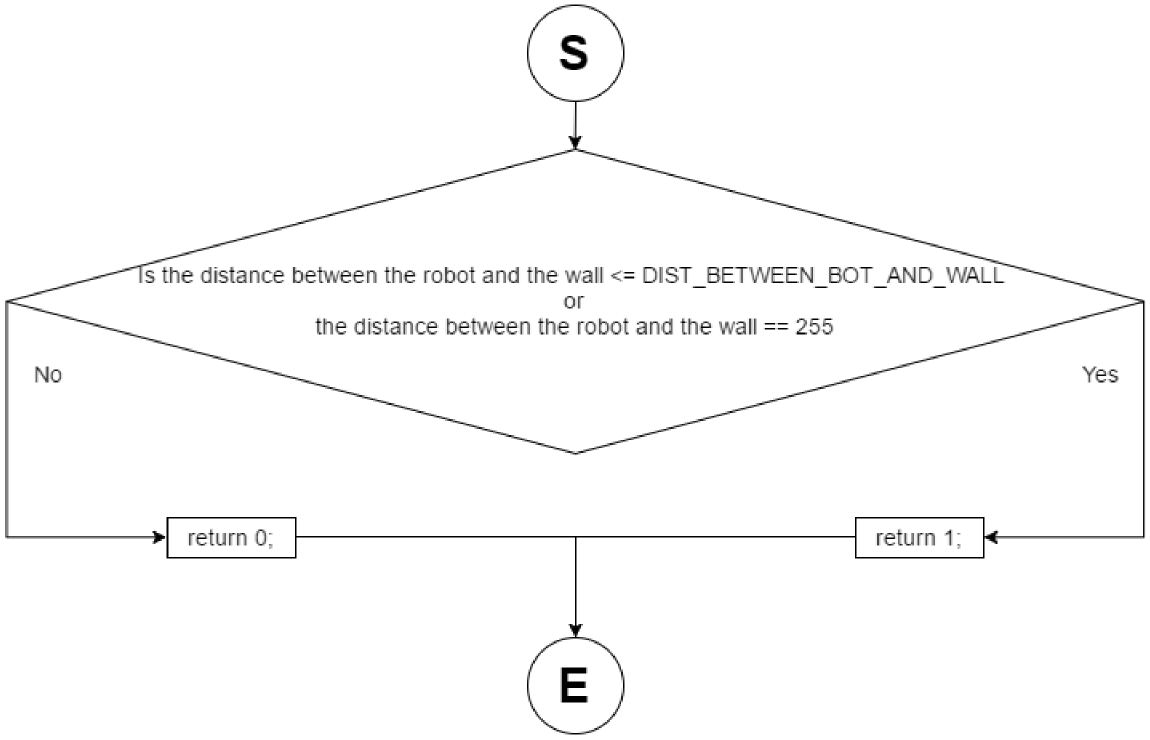

S
Is the distance between the robot and the wall <= DIST_BETWEEN_BOT_AND_WALL
or
the distance between the robot and the wall == 255
No
Yes
return 0;
return 1;
E

Figure 5: Flow Chart for Wall Detecting Function

- Very simple fuction that returns 1 if the sensor detects the wall
- Global variables and constants used are

  `DIST_BETWEEN_BOT_AND_WALL`

3.11 Function for Storing Data of the Walls

- Function that stores data of the walls to the variables

  `void writeWall(int direction);`

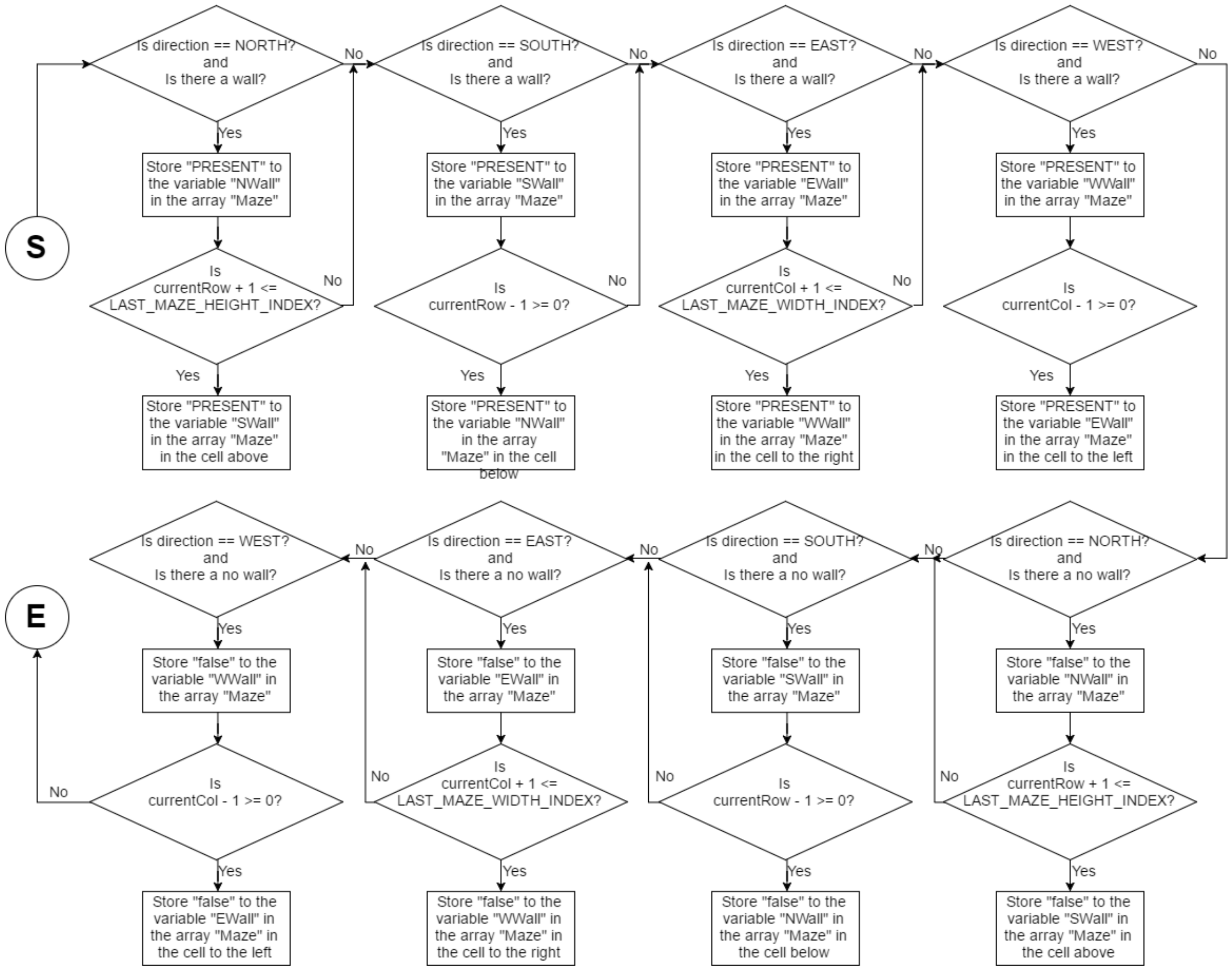

S
Is direction == NORTH? and Is there a wall?
Yes
No
Store "PRESENT" to the variable "NWall" in the array "Maze"
Is currentRow + 1 <= LAST_MAZE_HEIGHT_INDEX?
Store "PRESENT" to the variable "SWall" in the array "Maze" in the cell above
Is direction == SOUTH? and Is there a wall?
Store "PRESENT" to the variable "SWall" in the array "Maze"
Is currentRow - 1 >= 0?
Store "PRESENT" to the variable "NWall" in the array "Maze" in the cell below
Is direction == EAST? and Is there a wall?
Store "PRESENT" to the variable "EWall" in the array "Maze"
Is currentCol + 1 <= LAST_MAZE_WIDTH_INDEX?
Store "PRESENT" to the variable "WWall" in the array "Maze" in the cell to the right
Is direction == WEST? and Is there a wall?
Store "PRESENT" to the variable "WWall" in the array "Maze"
Is currentCol - 1 >= 0?
Store "PRESENT" to the variable "EWall" in the array "Maze" in the cell to the left
Is direction == NORTH? and Is there a no wall?
Store "false" to the variable "NWall" in the array "Maze"
Is currentRow + 1 <= LAST_MAZE_HEIGHT_INDEX?
Store "false" to the variable "SWall" in the array "Maze" in the cell above
Is direction == SOUTH? and Is there a no wall?
Store "false" to the variable "SWall" in the array "Maze"
Is currentRow - 1 >= 0?
Store "false" to the variable "NWall" in the array "Maze" in the cell below
Is direction == EAST? and Is there a no wall?
Store "false" to the variable "EWall" in the array "Maze"
Is currentCol + 1 <= LAST_MAZE_WIDTH_INDEX?
Store "false" to the variable "WWall" in the array "Maze" in the cell to the right
Is direction == WEST? and Is there a no wall?
Store "false" to the variable "WWall" in the array "Maze"
Is currentCol - 1 >= 0?
Store "false" to the variable "EWall" in the array "Maze" in the cell to the left
E

Figure 6: Flow Chart for Storing Data Function

  - The local variable direction is passed into the function but it does not return any variable
  - Global variables and constants used are

    ```
    NORTH
    SOUTH
    EAST
    WEST
    currentRow
    currentCol
    PRESENT
    LAST_MAZE_HEIGHT_INDEX
    LAST_MAZE_WIDTH_INDEX
    ```

  - This function calls in other functions

    `int thereIsWall();`

3.12 Functions for setting up the direction that need to be used

- Function that takes in the current direction of the robot and returns exact opposite direction of what the robot is facing

```
int findBackDir(int currentDirection);
```

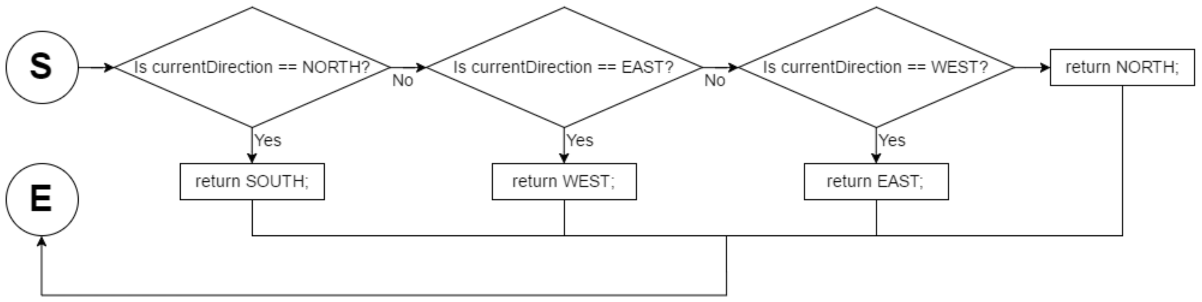

S
Is currentDirection == NORTH?
No
Is currentDirection == EAST?
No
Is currentDirection == WEST?
return NORTH;
Yes
Yes
Yes
return SOUTH;
return WEST;
return EAST;
E

Figure 7: Flow Chart for Finding Back Function

- Function that takes in the current direction of the robot and returns the right direction of the robot

```
int findRight(int currentDirection);
```

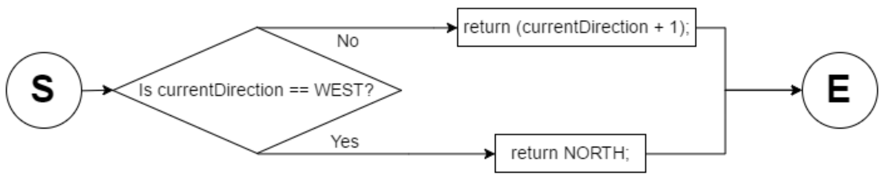


Figure 8: Flow Chart for Finding Right Function

- Function that takes in the current direction of the robot and returns the left direction of the robot

```
int findLeft(int currentDirection);
```

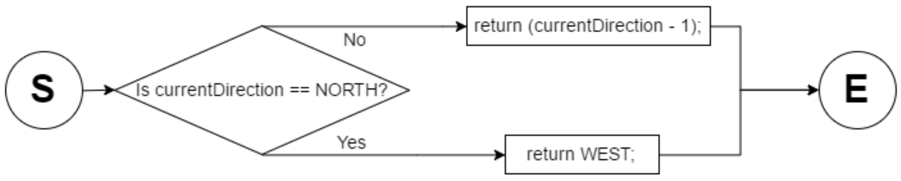


Figure 9: Flow Chart for Finding Left Function

  - Global variables and constants used are

    ```
    NORTH
    SOUTH
    EAST
    WEST
    ```

3.13 Functions for Finding Existence of Wall from the data

- Function that uses the data saved from detecting function

```
int isThereWallInDir(int wallDir);
```

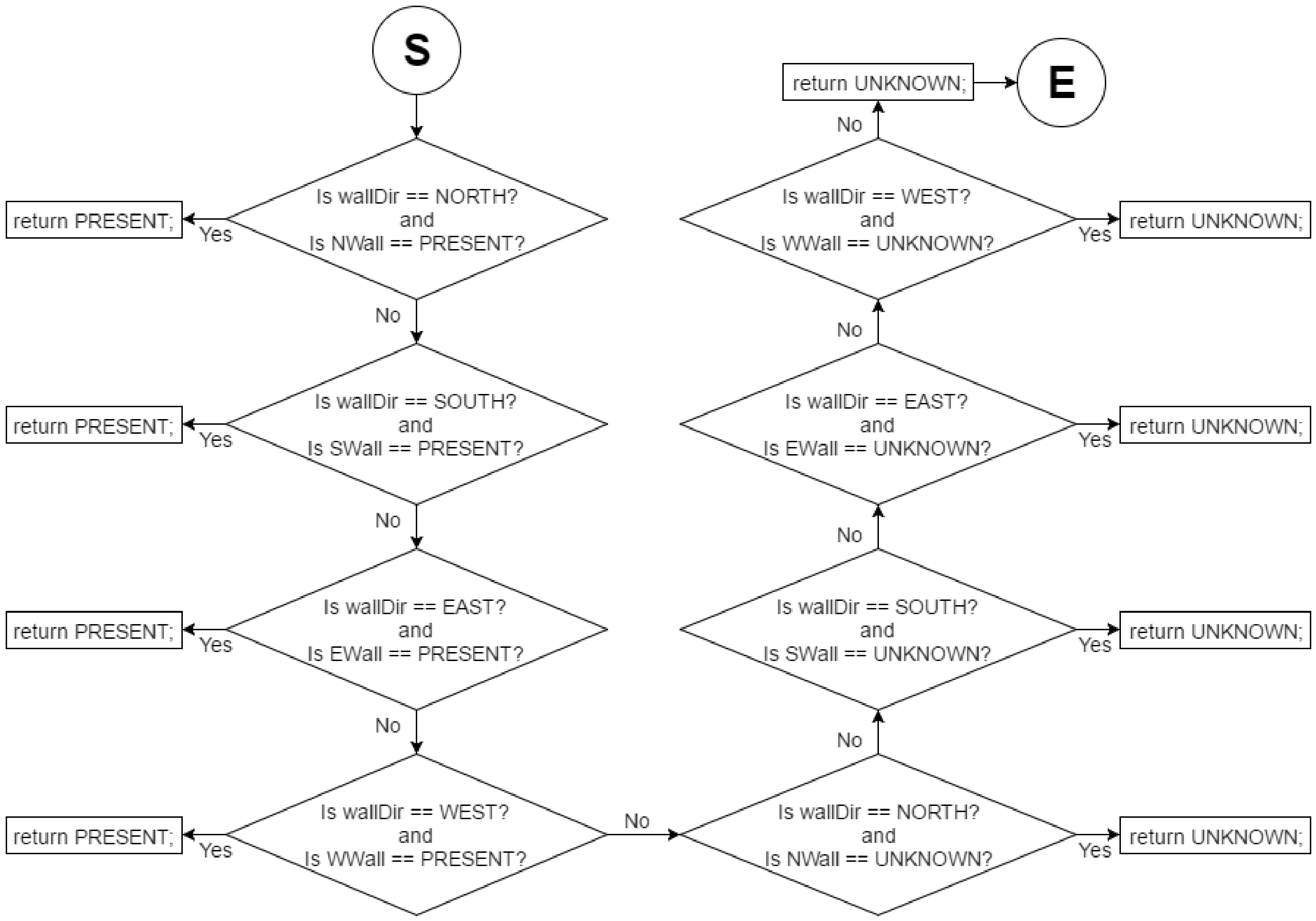

S
Is wallDir == NORTH? and Is NWall == PRESENT?
Yes
return PRESENT;
No
Is wallDir == SOUTH? and Is SWall == PRESENT?
Yes
return PRESENT;
No
Is wallDir == EAST? and Is EWall == PRESENT?
Yes
return PRESENT;
No
Is wallDir == WEST? and Is WWall == PRESENT?
Yes
return PRESENT;
No
Is wallDir == NORTH? and Is NWall == UNKNOWN?
Yes
return UNKNOWN;
No
Is wallDir == SOUTH? and Is SWall == UNKNOWN?
Yes
return UNKNOWN;
No
Is wallDir == EAST? and Is EWall == UNKNOWN?
Yes
return UNKNOWN;
No
Is wallDir == WEST? and Is WWall == UNKNOWN?
Yes
return UNKNOWN;
No
return UNKNOWN;
E

Figure 10: Flow Chart for Finding Existence of Wall from the data

- Global variables and constants used are

```
NORTH
SOUTH
EAST
WEST
PRESENT
UNKNOWN
NOT_PRESENT
```

3.14 Functions for Readjusting in Certain Directions

- Function that readjusts robot's direction by moving towards the wall and comes back to the center of the cell. For this function particularly, we readjust using walls to the front and to the right.

  `void reAdjustCW(int direction);`

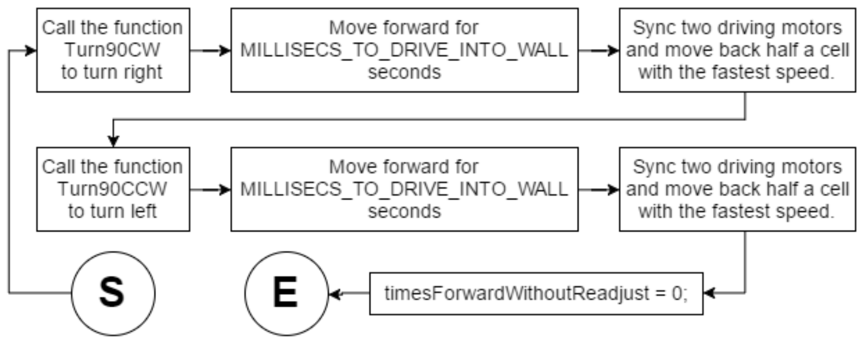

Call the function Turn90CW to turn right
Move forward for MILLISECS_TO_DRIVE_INTO_WALL seconds
Sync two driving motors and move back half a cell with the fastest speed.
Call the function Turn90CCW to turn left
Move forward for MILLISECS_TO_DRIVE_INTO_WALL seconds
Sync two driving motors and move back half a cell with the fastest speed.
S
E
timesForwardWithoutReadjust = 0;

Figure 11: Flow Chart for Readjusting using Front wall and Right wall

- Function that readjusts robot's direction by moving towards the wall and comes back to the center of the cell. For this function particularly, we readjust using walls to the front and to the left.

  `void reAdjustCCW(int direction);`

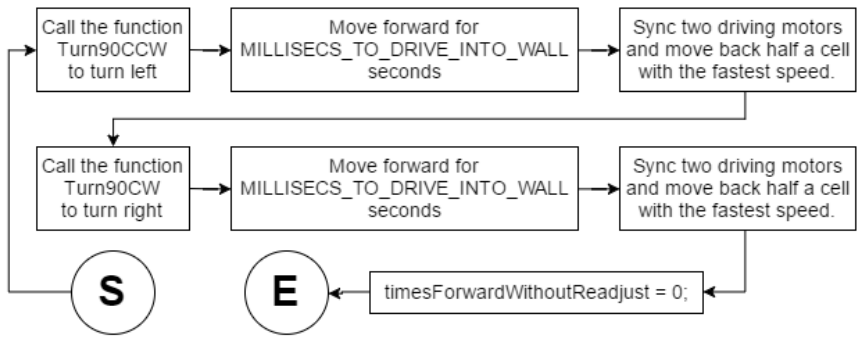

Call the function Turn90CCW to turn left
Move forward for MILLISECS_TO_DRIVE_INTO_WALL seconds
Sync two driving motors and move back half a cell with the fastest speed.
Call the function Turn90CW to turn right
Move forward for MILLISECS_TO_DRIVE_INTO_WALL seconds
Sync two driving motors and move back half a cell with the fastest speed.
S
E
timesForwardWithoutReadjust = 0;

Figure 12: Flow Chart for Readjusting using Front wall and Left wall

  - The local variable direction is passed into the function but it does not return any variable
  - Global variables and constants used are

```
FORWARD
BACKWARD
SIZE_OF_ONE_CELL
CIRCUMFERENCE_OF_WHEEL
DRIVE_GEAR_RATIO
ONE_ROTATION
UNCERTAINTY_READJUST
MILLISECS_TO_DRIVE_INTO_WALL
```

  - This function calls in other functions

```
int Turn90CW(int direction);
int Turn90CCW(int direction);
```

- Function that readjusts robot's direction by moving towards the wall and comes back to the center of the cell. With this Function, the robot uses the saved data to determine to readjust in CW direction or CCW direction

  ```
  void reAdjustWayBack(int direction);
  ```

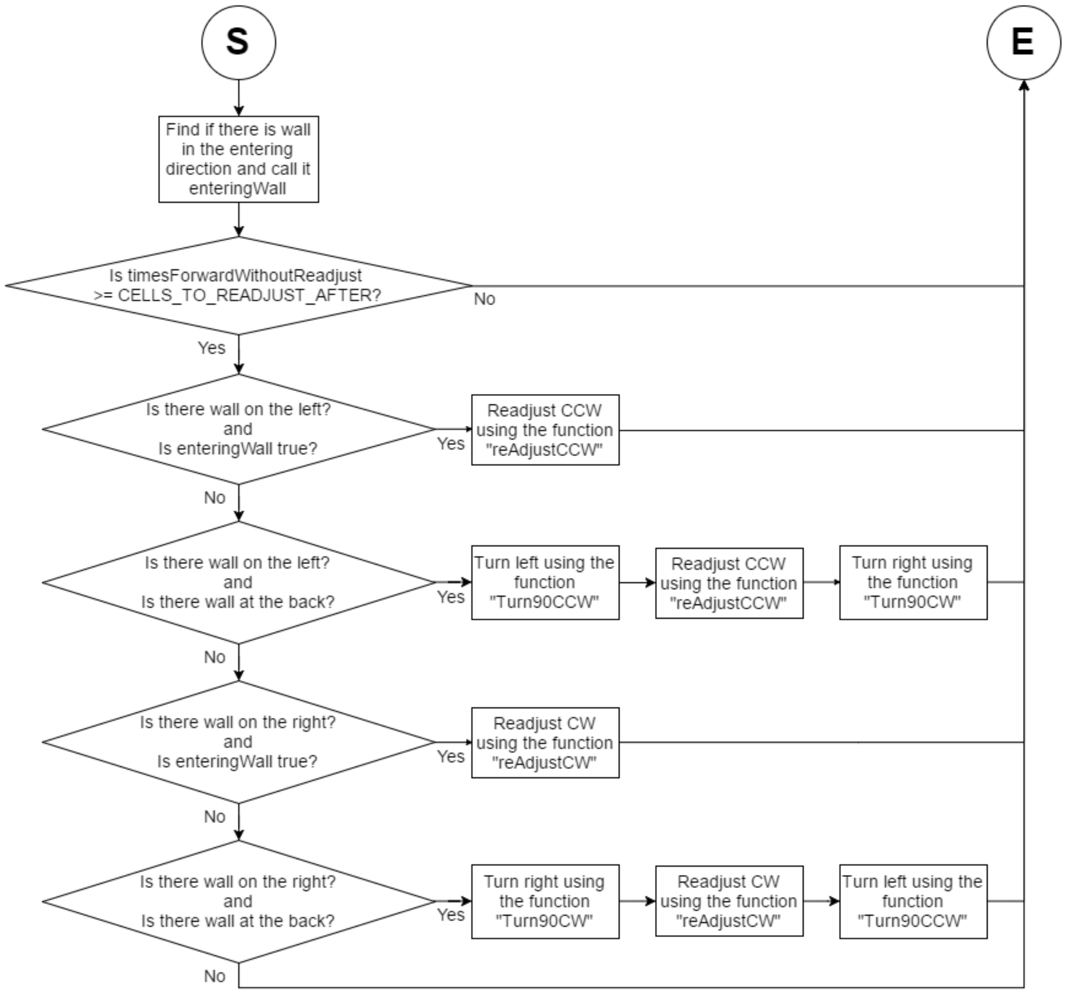

S
E
Find if there is wall in the entering direction and call it enteringWall
Is timesForwardWithoutReadjust >= CELLS_TO_READJUST_AFTER?
No
Yes
Is there wall on the left? and Is enteringWall true?
Yes
Readjust CCW using the function "reAdjustCCW"
No
Is there wall on the left? and Is there wall at the back?
Yes
Turn left using the function "Turn90CCW"
Readjust CCW using the function "reAdjustCCW"
Turn right using the function "Turn90CW"
No
Is there wall on the right? and Is enteringWall true?
Yes
Readjust CW using the function "reAdjustCW"
No
Is there wall on the right? and Is there wall at the back?
Yes
Turn right using the function "Turn90CW"
Readjust CW using the function "reAdjustCW"
Turn left using the function "Turn90CCW"
No

Figure 13: Flow Chart for Readjusting using walls detected

  - The local variable direction is passed into the function but it does not return any variable
  - Global variables and constants used are

    ```
    timesForwardWithoutReadjust
    CELLS_TO_READJUST_AFTER
    PRESENT
    ```

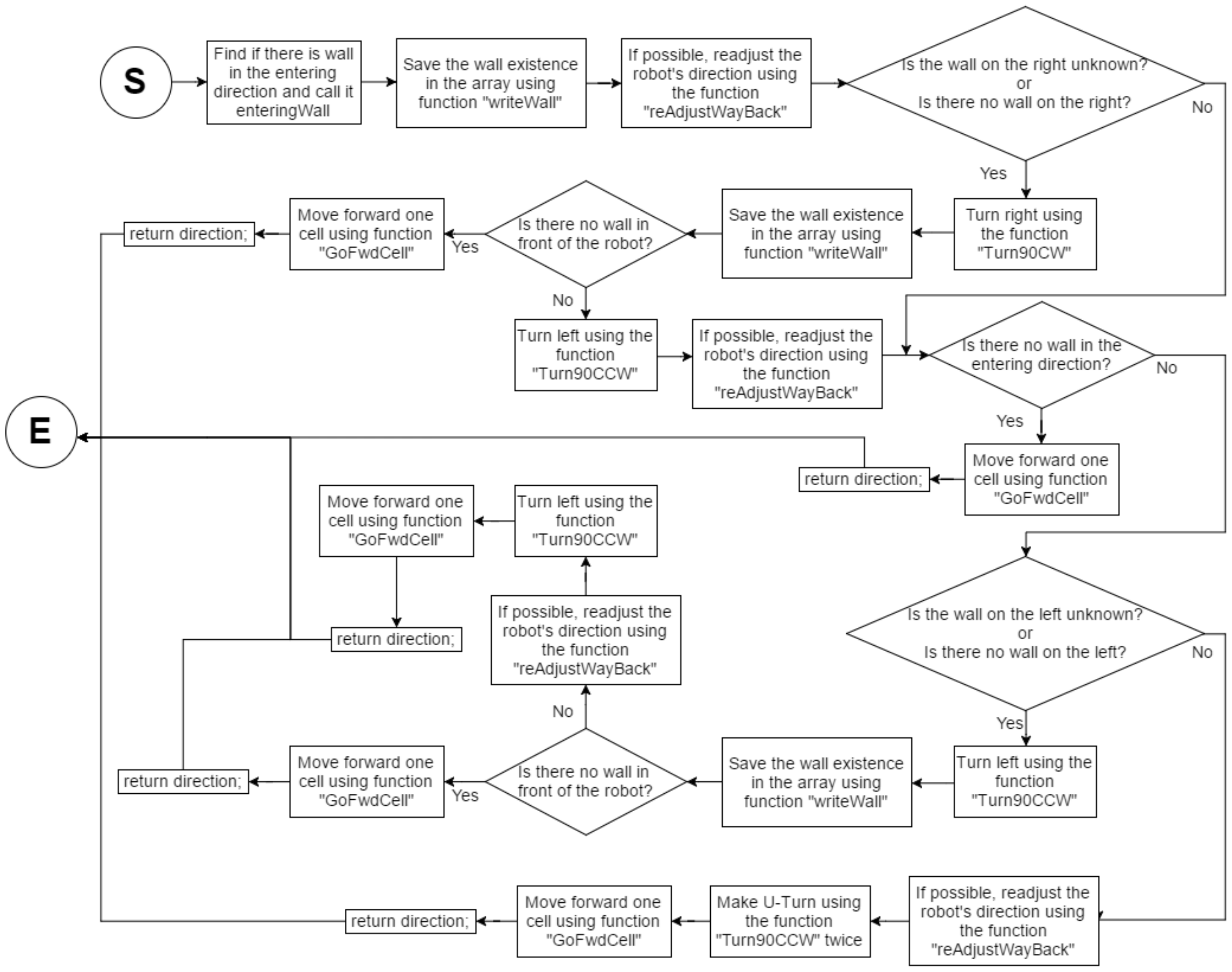
  - This function calls in other functions

    ```
    thereIsWall();
    reAdjustCW(int direction);
    reAdjustCCW(int direction);
    findLeft(int currentDirection);
    findRight(int currentDirection);
    findBackDir(int currentDirection);
    isThereWallInDir(int wallDir);
    ```

3.15 Function for Movement All Together

- Function that contains all other smaller functions for movement and this is the only function called in the main task

```
int MovementWithSensor(int direction);
```



S
Find if there is wall in the entering direction and call it enteringWall
Save the wall existence in the array using function "writeWall"
If possible, readjust the robot's direction using the function "reAdjustWayBack"
Is the wall on the right unknown? or Is there no wall on the right?
Yes
No
Turn right using the function "Turn90CW"
Save the wall existence in the array using function "writeWall"
Is there no wall in front of the robot?
Yes
Move forward one cell using function "GoFwdCell"
return direction;
No
Turn left using the function "Turn90CCW"
If possible, readjust the robot's direction using the function "reAdjustWayBack"
Is there no wall in the entering direction?
Yes
Move forward one cell using function "GoFwdCell"
return direction;
No
E
Is the wall on the left unknown? or Is there no wall on the left?
Yes
Turn left using the function "Turn90CCW"
Save the wall existence in the array using function "writeWall"
Is there no wall in front of the robot?
Yes
Move forward one cell using function "GoFwdCell"
return direction;
No
If possible, readjust the robot's direction using the function "reAdjustWayBack"
Turn left using the function "Turn90CCW"
Move forward one cell using function "GoFwdCell"
return direction;
No
If possible, readjust the robot's direction using the function "reAdjustWayBack"
Make U-Turn using the function "Turn90CCW" twice
Move forward one cell using function "GoFwdCell"
return direction;

Figure 14: Flow Chart for Movement Function

- The local variable direction is passed into the function and it returns the same variable direction
- Global variables and constants used are

```
UNKNOWN
NOT_PRESENT
```

- This function calls in other functions

```
writewall(int direction);
reAdjustWayBack(int direction);
isThereWallInDir(int wallDir);
findRight(int currentDirection);
thereIsWall();
goFwdCell(int direction);
Turn90CCW(int direction);
Turn90CW(int direction);
```

3.16 Functions for Returning Algorithm

- Function that deletes the duplicates from the array which saved up how the robot entered each cell. For example, if the robot moved two opposite directions in order, it is not necessary. Therefore, we delete the duplicates from the array

  ```
  void deleteDuplicates();
  ```

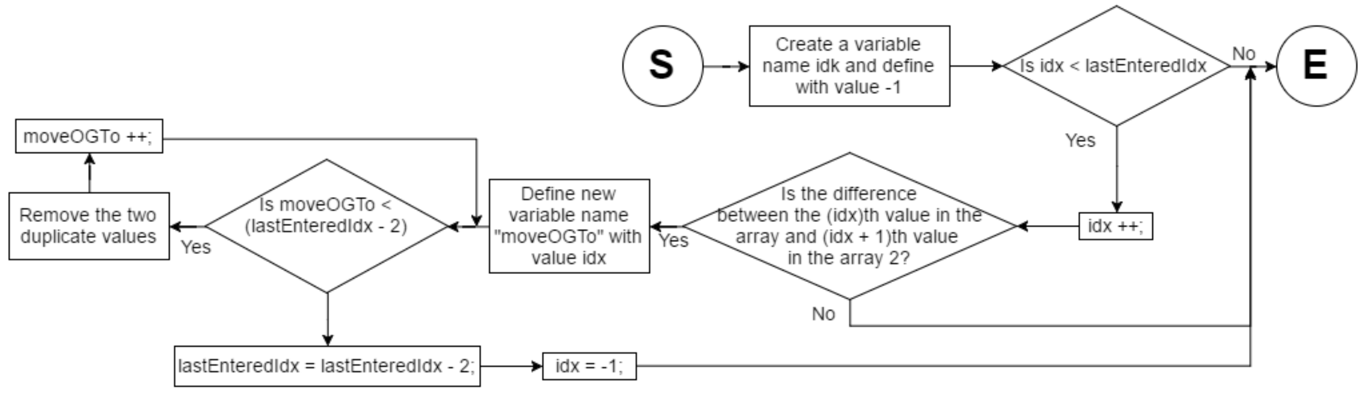

S
Create a variable name idk and define with value -1
Is idx < lastEnteredIdx
No
E
Yes
idx ++;
Is the difference between the (idx)th value in the array and (idx + 1)th value in the array 2?
Yes
No
Define new variable name "moveOGTo" with value idx
Is moveOGTo < (lastEnteredIdx - 2)
Yes
Remove the two duplicate values
moveOGTo ++;
lastEnteredIdx = lastEnteredIdx - 2;
idx = -1;

Figure 15: Flow Chart for Deleting Duplicate Function

- Function that reverses the direction from the array which saved up how the robot entered each cell. For example, if the robot went in to the cell with direction East, then we change it to West. Therefore, we change all the directions to its opposite.

  ```
  void reverseDirection();
  ```

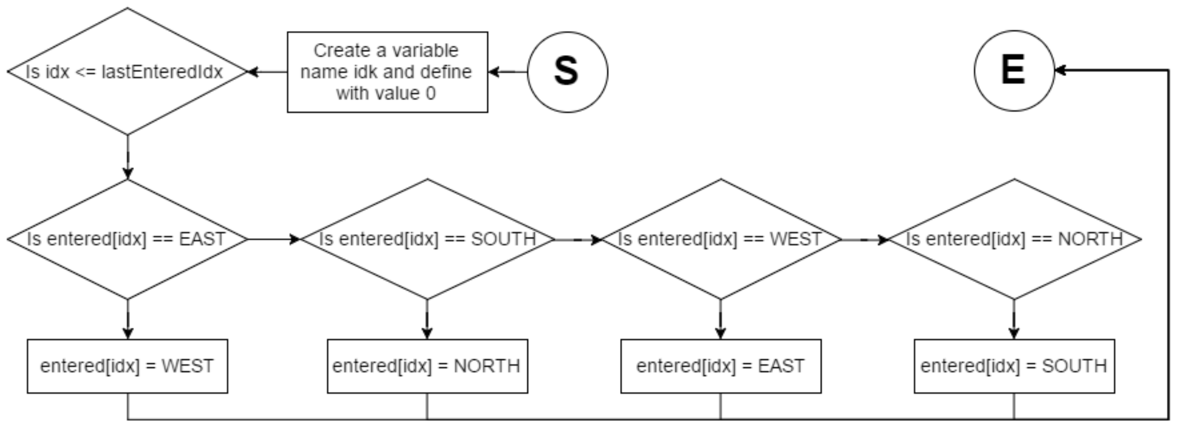


Figure 16: Flow Chart for Reversing Order of Values in the Array

- Function that takes in the variable direction and goes back to the initial position in the cell with the new array created by two functions above

  `void goingBackFastestRoute(int direction);`

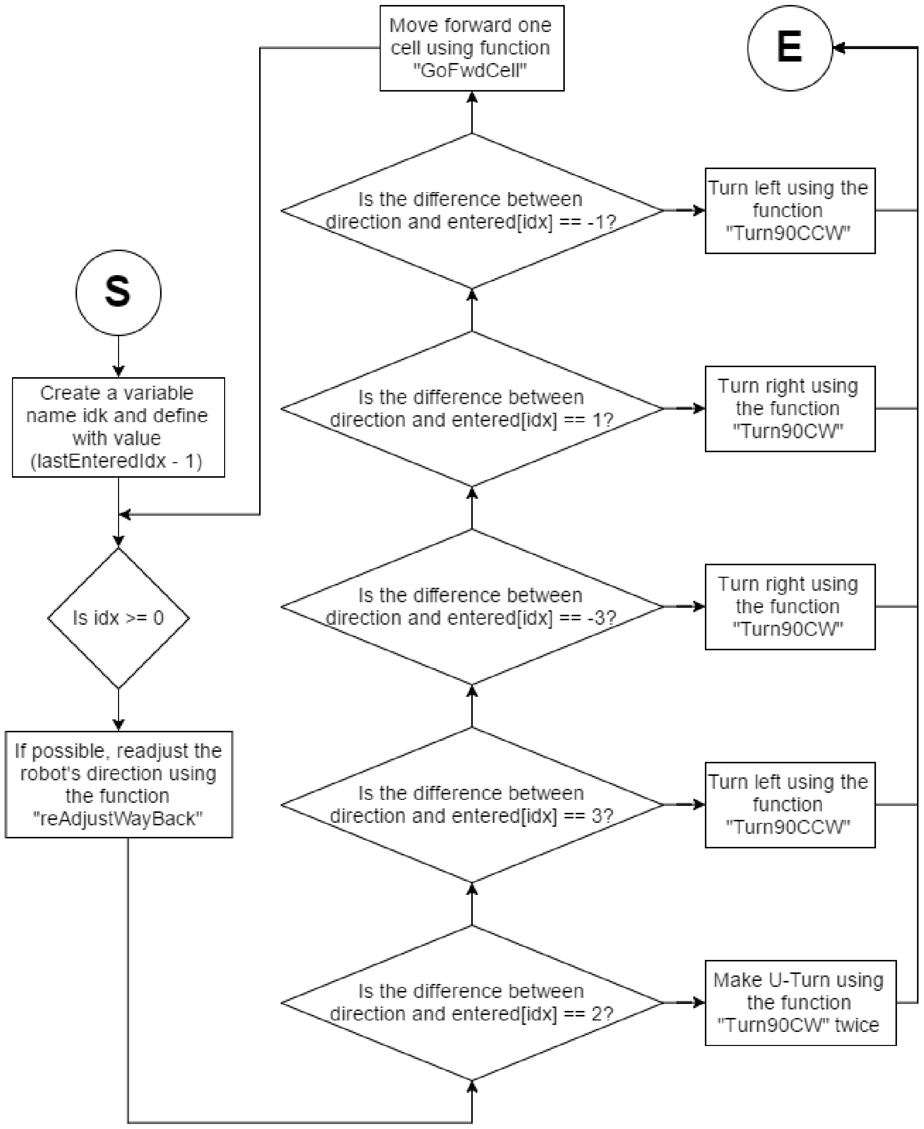

S
Create a variable name idk and define with value (lastEnteredIdx - 1)
Is idx >= 0
If possible, readjust the robot's direction using the function "reAdjustWayBack"
Is the difference between direction and entered[idx] == 2?
Make U-Turn using the function "Turn90CW" twice
Is the difference between direction and entered[idx] == 3?
Turn left using the function "Turn90CCW"
Is the difference between direction and entered[idx] == -3?
Turn right using the function "Turn90CW"
Is the difference between direction and entered[idx] == 1?
Turn right using the function "Turn90CW"
Is the difference between direction and entered[idx] == -1?
Turn left using the function "Turn90CCW"
Move forward one cell using function "GoFwdCell"
E

Figure 17: Flow Chart for Function to Go Back to Initial Position

  - Global variables and constants used are

    ```
    lastEnteredIdx
    EAST
    WEST
    SOUTH
    NORTH
    ```

  - This function calls in other functions

    ```
    reAdjustWayBack(int direction);
    Turn90CW(int direction);
    Turn90CCW(int direction);
    ```

3.17 Main Function

- More than ten functions were declared for simplicity of the main function. This function sums up all smaller functions

  `task main()`

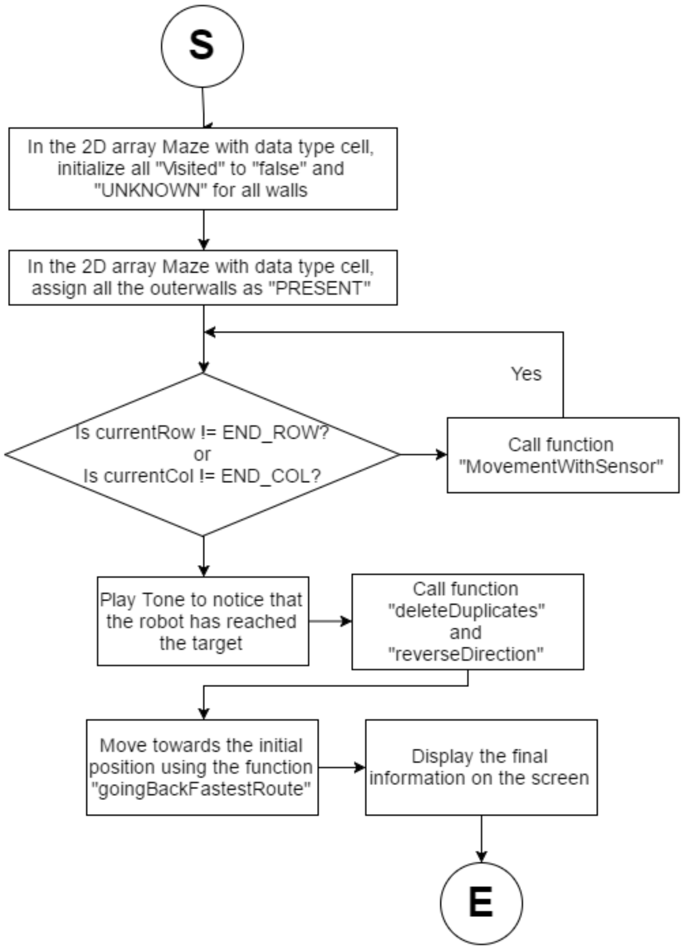

S
In the 2D array Maze with data type cell, initialize all "Visited" to "false" and "UNKNOWN" for all walls
In the 2D array Maze with data type cell, assign all the outerwalls as "PRESENT"
Yes
Is currentRow != END_ROW? or Is currentCol != END_COL?
Call function "MovementWithSensor"
Play Tone to notice that the robot has reached the target
Call function "deleteDuplicates" and "reverseDirection"
Move towards the initial position using the function "goingBackFastestRoute"
Display the final information on the screen
E

Figure 18: Flow Chart of the Main Function

  - Global variables and constants used are

```
MAZE_WIDTH
MAZE_HEIGHT
UNKNOWN
PRESENT
lastEnteredIdx
FREQUENCY
MILI_TO_BEEP_FOR
```

  - This function calls in other functions

```
MovmentWithSensor(int direction);
deleteDuplicates();
reverseDirection();
goingBackFastestRoute(int direction);
drawInfo(int direction);
```

4 Appendix

Source of Code with brief comments