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
XOURNAL LOL

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 3^{rd} 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 Represent the Size of the Maze

- Declared one variable that represents the dimension of a single cell

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

- Declared four variables that represent the size of the maze

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

- Declared four constants that represent the initial position of the robot in the maze. These will be entered when we begin our demo.

```
int const START_ROW = ;
int const START_COL = ;
int const END_ROW = ;
int const END_COL = ;
```

- Declared two variables that represents the target position in the maze

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

- Declared two variables that represents the current position of the robot in the maze

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

- Declared an array that represents the orientation that the bot has as it enters each cell.

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

3.3 Variables Used for Representation of Directions

- The four integer values that we used to represent each direction were declared as constants.

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

- Structure name cell was declared and it has five parameters. We 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;
```

- Declared the variable name "Maze" that has the data type cell. 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];
```

- Declared a global variable that represents current direction of the robot. This is initialized as north as this is the orientation of the bot when it first enters the maz

```
int direction = NORTH;
```

3.4 Variables Used for Moving Mechanism - Going Forward

- Since minor differences exist between each motors, the robot was not moving as straight as it sould be. Therefore, one of the motor had to slow down in order to make the robot move exactly straight as we wanted.

```
int const SPEED_ON_MOTOR_DIF = 10;
```

- The uncertainty of two motors and backlash of the gears exists, this caused the robot to not move exactly the distance required. Therefore, uncertainty had to be added in order to move robot by the exact distance required.

```
float const UNCERTAINTY_STRAIGHT = 23;
```

- Each speed of the motors were defined with constants to simplify the code.

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

3.5 void goFwdCell()

- **Example of our code**

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

Briefly explain the algorithm of the maze bot

3.6 Function Description

Explain what each function does

3.7 Flow Chart

Flow chart of the algorithms (I think there is a online thing where u can make flow chart)

3.8 Function Parameters

Yea

4 Appendix

Source of Code with brief comments