Control of Mobile Robotics
Spring 2016

Lab 3

Sensors and Actuators

11 April, 2016

Gerardo Sanchez
Adeoluwa Oyenusi

Task Description

The objective of this lab is to develop an algorithm to navigate, localize, and build maps for three different mazes. The lab is divided into 3 tasks.

1. This task requires us to develop an algorithm to navigate all of 3 different mazes and the robot must visit each cell at least once. The robot can start in any cell and must complete the navigation without any human intervention.
2. This task requires us to develop an algorithm to navigate the mazes and mark which cells have been visited. “0” represents the visited cells and “X” represents the cells that have not been visited. These will be displayed on the LCD display. Additionally, the LCD must provide a flashing color every time the robot moves to a new grid. When moving to a grid on the right, the color red will be flashed, when moving to a grid on the left, green will be flashed, when moving to a grid up, blue will be flashed, and when moving to a grid down, yellow will be flashed. Other than these cases, the LCD should show a white background.

Navigation

Video link: https://www.youtube.com/watch?v=0rnp-CipwkQ

For the purposes of navigating through a maze, the mobile robot is configured to wall follow and to detect corners. Our navigation algorithm is focused on sensor readings from the LEFT and FRONT infrared sensor. Using left wall following, the system has a variable speed component that will adjust the robot to follow the wall accordingly, given a specified threshold.

In addition to this, there are two scenarios that the robot will encounter in a maze and they have their respective behaviors. For example, when encountering a corner whilst wall following, the robot will sample the FRONT sensor multiple times to ensure the presence of a wall. Upon confirmation of the vicinity of the front and left wall, the robot will initiate a RIGHT turn sequence. The RIGHT turn sequence is set to an arbitrary delay that will rotate the robot 90 degrees to evaluate further conditions in the maze. The robot will take a proper course of action to prevent colliding into walls. Our algorithm comes equipped with a failsafe algorithm that will double check the sensor conditions to avoid false positives.

There are cases where the robot may not have a wall to follow, it will assume that it needs to make a LEFT turn until it finds a wall to follow. This will allow the robot to overcome paths that require more than one left turn. With this final component of the navigation algorithm, the robot proves successful when traversing all three configurations of the mazes.

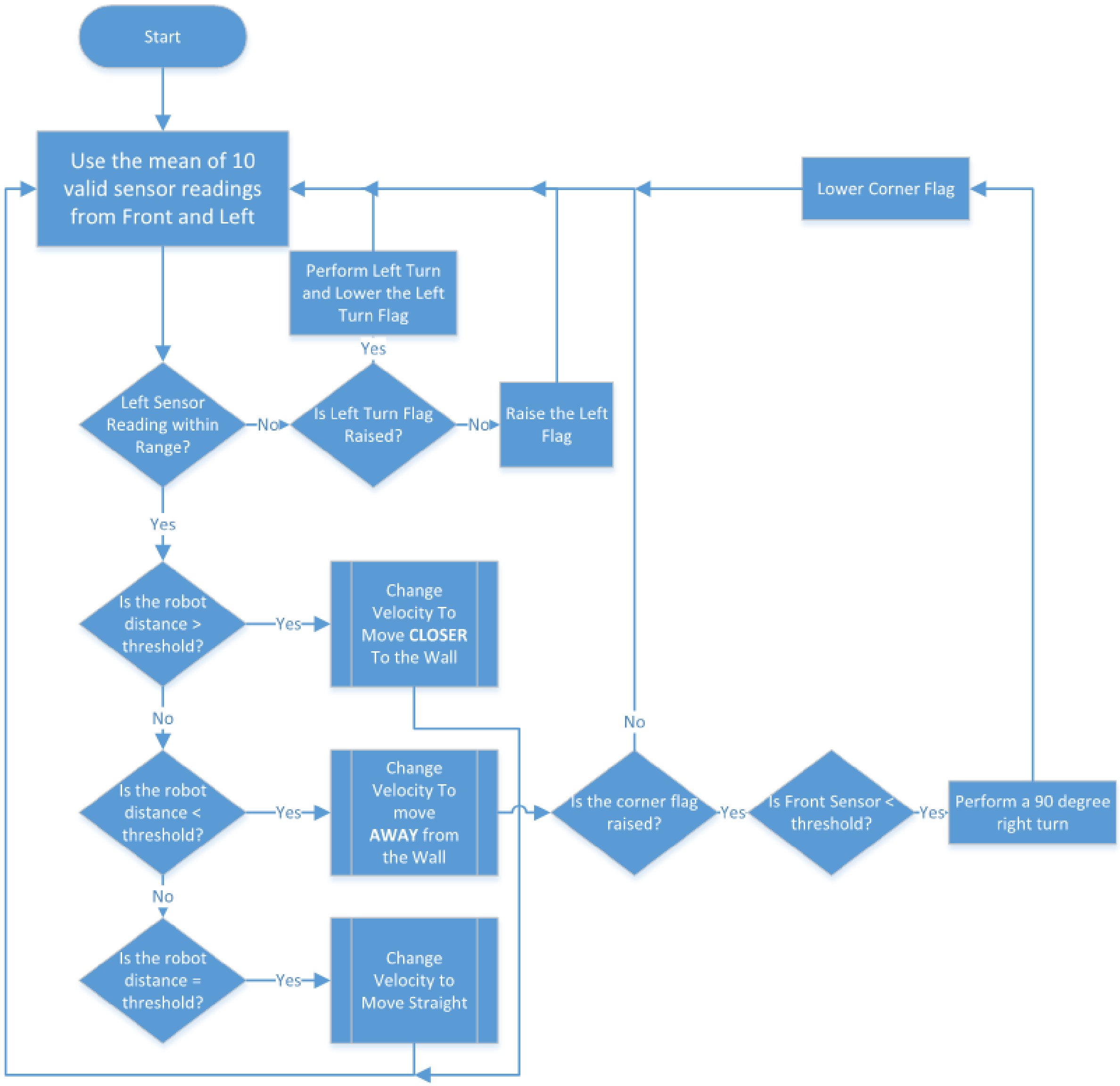

Figure 1 - Navigation Test

Localization

Video link: https://www.youtube.com/watch?v=o6_fIfw4BBo

The localization technique is a continuation of the navigation algorithm which integrates a conscious orientation for the robot to utilize. The localization algorithm requires a known starting position and direction. From this, the robot will be able to recognize any direction and properly mark new cells as visited. A counter has been implemented into the sampling algorithm which represents roughly the number of samples that can occur within the traversal of a cell. This counter will help keep track of when a whole cell has been traversed since the last left or right turn.

We can extrapolate the direction and the counter value to mark cells as visited via a 2D array representation of the maze. The data structure represents all the cells in the grid and it will be scanned constantly throughout the robot's traversal of the maze. The robot will stop only when all cells have been visited; otherwise, it will continue to perform the navigation techniques in the previous part.

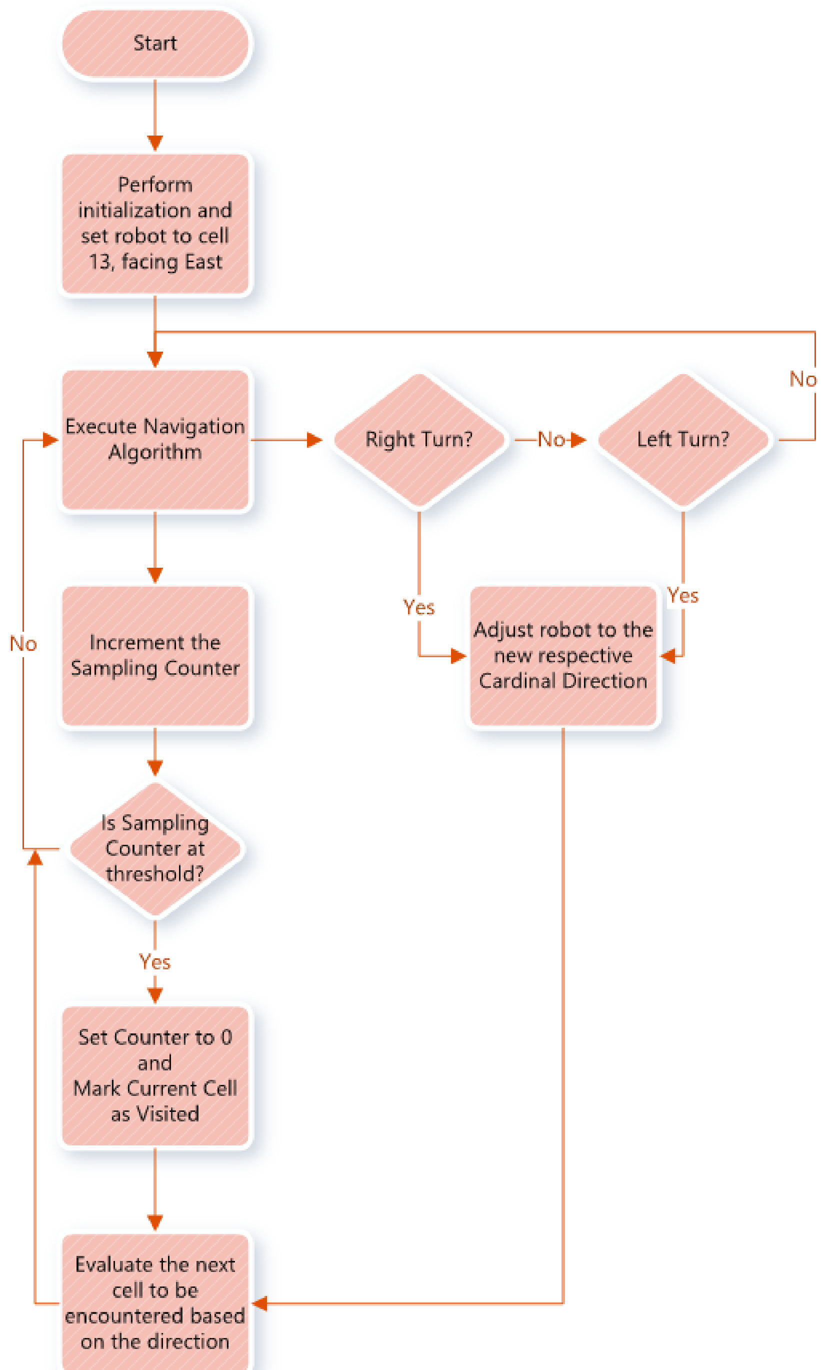

Figure 2 – Localization Test

Code Description

Navigation

Description:

The Navigation function below handles navigating through the different mazes. First we read in our distance from the left and front sensors. Then we use these values to do left wall following. Furthermore, we handle corner cases and left turns. We check if we are less than 9 inches from the left wall, if we are, we do wall follow and also use the cornerFlg to check if we have encountered a corner. Furthermore, we use the function isAWall to help the right turns at the corner. The next condition checks if our left sensor reads a distance greater than 9 inches. In this case, we want to make a left turn. We use our left turn function to do this. We also use a bool variable leftTrnFlg to help us with this. For the wall following part, we are constantly reading in values and making adjustments to the robot to move closer to the wall, farther from it, or align. We use the functions moveCloserToWall, moveAwayFromWall, and AlignWithWall to do this respectively. Lastly the functions, moveALittle and stopRobot make the robot move forward a little and stop it respectively.

Code:

```
void Navigation()
{
  lcd.clear();
  int lngF; //long Front value
  bool cornerFlg = false;
  bool leftTrnFlg = false;

  int counter = 0;
  int wfCntr = 0;
  while(1)
  {
    shrtF = trueDist(SFSensor, 10);
    shrtL = trueDist(SLSensor, 10);

    printLCD(shrtF, shrtL, lVel, rVel);



    if(shrtL < 9)
    {
      wallFollow(&wfCntr, shrtL);

      if(cornerFlg)
      {
        stopRobot();
        if(isAWall(shrtF))
        {
          //perform corner turn
```

```
        rightTurn();
      }
      cornerFlg = false;
      continue;
    }else
    {
      cornerFlg = isAWall(shrtF);

    }

  }else if(shrtL >= 9)
  {
    //suspect that it is time for a LEFT turn sequence
    if(leftTrnFlg)
    {
      leftTurn();
      leftTrnFlg = false;
      continue;

    }else
    {
      stopRobot();
      leftTrnFlg = true;
      continue;
    }

  }
 }
}

bool isAWall(int shrtF)
{
  return shrtF < 5;
}


void wallFollow(int* wfCntr, const int shrtL)
{
  int threshold = 4;
  if(shrtL > threshold)
  {
    //Left Wall is in range, follow it accordingly.
     ++*wfCntr;
     moveCloserToWall();
     if(*wfCntr == 5)
     {
        wfCntr = 0;
        //offset the movement by moving in the opposite direction
        moveAwayFromWall();
     }
   }else if(shrtL < threshold)
   {
      //The robot is too close to wall. Move away.
      ++*wfCntr;
      moveAwayFromWall();
      if(*wfCntr == 3)
```

```
      {
        wfCntr = 0;
        moveCloserToWall();
      }
  } else if(shrtL == threshold)
  {
    *wfCntr = 0;
    alignWithWall();
  }
  return;
}

void rightTurn()
{
  int mlVel = 95;
  int mrVel = 96;
  myMover.cmd_vel(mlVel, mrVel);
  delay(800);
}

void leftTurn()
{
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Left Turn");

  moveForwardALittle(1000);

  stopRobot();

  //perform the left turn!
  int mlVel = 90;
  int mrVel = 80;
  myMover.cmd_vel(mlVel, mrVel);
  delay(1200);

  moveForwardALittle(1000);
}


void moveCloserToWall()
{ //too far from wall
  //myMover.cmd_vel(93, 85);
  int mlVel = 95;
  int mrVel = 83; //SPEED right up
  myMover.cmd_vel(mlVel, mrVel);
  //delay(200);
}

void moveAwayFromWall()
{ //too close to wall
 // myMover.cmd_vel(95, 87);
 int mlVel = 97; //SPEED left up
 int mrVel = 86;
 myMover.cmd_vel(mlVel, mrVel);
 //delay(200);
}
```

```
void alignWithWall()
{ //align
  //myMover.cmd_vel(95, 85);
  int mlVel = 95;
  int mrVel = 86; //right motor is weaker
  myMover.cmd_vel(mlVel, mrVel);

}

void moveForwardALittle(int mDelay)
{ //move forward a little before left turn
  //myMover.cmd_vel(95, 85);//allow robot to move forward a bit before
turning left
  int mlVel = 95;
  int mrVel = 85;
  myMover.cmd_vel(mlVel, mrVel);
  delay(mDelay); //allow the robot to move to the center of the cell to spin
to the left
}

void stopRobot()
{//stops robot
  //myMover.cmd_vel(90,90);
  int mlVel = 90;
  int mrVel = 90;
  myMover.cmd_vel(mlVel, mrVel);
  delay(500);
}
```

Localization

Description

For the localization part of the lab, we used our navigation algorithm to navigate the maze. However, we made some minor adjustments to some functions and we included a struct to help us hold the directions. The function Localization has the algorithm we used for localization. The variables indexDirection and crrntRobotDir help us keep track of where we are. The indexDirection is used as an offset into the array we have for the direction in our struct while crrntRobotDir is used to help us keep track of the directions N, E, S, W. In the localization function, we use a variable called "counter" to count when we move to a new cell. We arrived at this value by doing several trials. Once this counter reaches 97, we update the maze and say we have visited the cell we just finished. When we make left turns and right turn, we reset the counter to 20 and 40 respectively. We do this because we are not totally moving straight and we do not need to count from 0. Our left and right turn functions were slightly updated by adding variables indexDir and rbtDir which change the directions whenever we make a right or left turn. These values get passed in the function updateDirection along with whether it is a left or right turn. In the updateDirection function, we change the index direction of the struct depending of if it was a right or left turn. Our whereTo function calculates the cell we need to move to based on the current cell and calls the updateCellStatus to print to the LCD. The updateCellStatus function simply prints to the LCD the current cell has been visited. Lastly, the function cnfgOrientations is used configure the orientation of our orientation struct when we first run our algorithm.

Code:

```
typedef struct orientation {
  char mDirection;
  int color;
} orientation;

orientation mOrientation[4];


void Localization()
{
  //Localization Variables
  //always starting East in block 13
  int indexDirection = 1; //default index for EAST
  char crrntRobotDir = mOrientation[indexDirection].mDirection;
  lcd.setBacklight(RED);

  int lngF; //long Front value
  bool cornerFlg = false;
  bool leftTrnFlg = false;

  int counter = 40;
  int wfCntr = 0;
  int dumShrtF = 0;

  //cell numbers


  while (1)
  {
    ++counter;
    dumShrtF = trueDist(SFSensor, 20);
    if (dumShrtF != 0)
    {
      shrtF = dumShrtF;
    }
    else
    {
      moveForwardALittle(100);
      continue;
    }
    shrtL = trueDist(SLSensor, 10);

    //Marks off cells that are visited
    if (counter == 97)
    {
      counter = 0;
      updateCellStatus(toCell, '0');

      toCell = whereTo(crrntRobotDir, toCell);
    }
```

```
  if (counter >= 10)
  {
    lcd.setCursor(14, 1);
    lcd.print(counter);
  } else {
    lcd.setCursor(14, 1);
    lcd.print(" ");
    lcd.print(counter);
  }

  if (shrtL < 9)
  {
    wallFollow(&wfCntr, &counter, shrtL);
    if (cornerFlg)
    {
      stopRobot();
      moveForwardALittle(200);
      if (isAWall(shrtF))
      {
        rightTurn(&indexDirection, &crrntRobotDir);

        counter = 40; //RIGHT turn middle of CELL
      }
      cornerFlg = false;
      continue;
    } else
    {
      cornerFlg = isAWall(shrtF);
      continue;
    }
  } else if (shrtL >= 9)
  {
    //suspect that it is time for a LEFT turn sequence
    if (leftTrnFlg)
    {
      if(counter > 90)
      {
        toCell = whereTo(crrntRobotDir, toCell);
      }
      leftTurn(&indexDirection, &crrntRobotDir);
      updateCellStatus(toCell, '0');
      toCell = whereTo(crrntRobotDir, toCell);
      counter = 20;
      leftTrnFlg = false;
      continue;
    } else
    {
      stopRobot();
      leftTrnFlg = true;

      continue;
    }
  }

}
return;
```

```
}



void leftTurn(int* indexDir, char* rbtDir)
{
  moveForwardALittle(1200);

  stopRobot();

  //perform the left turn!
  int mlVel = 86;//previously 90
  int mrVel = 84;//previously 80
  myMover.cmd_vel(mlVel, mrVel);
  delay(800); //Apr 8 turns too wide, decreased from 1300ms

  updateDirection('l', indexDir, rbtDir);
  moveForwardALittle(1300);
}

void rightTurn(int* indexDir, char* rbtDir)
{
  int mlVel = 95;
  int mrVel = 96;
  myMover.cmd_vel(mlVel, mrVel);
  delay(850); //Apr 6 turns too small, increased from 800ms
  updateDirection('r', indexDir, rbtDir);
}

void updateDirection(char turn, int* indxDir, char* rbtDir)
{ //change our cardinal orientation within the cube

  //  east --> right turn --> south (++1)
  //  east --> left turn --> north (--1)

  switch (turn)
  {
    case 'r':
      *indxDir += 1;
      break;
    case 'l':
      *indxDir -= 1;
      break;
  }

  if (*indxDir <= -1) {
    *indxDir = 3;
  } else
  {
    *indxDir = *indxDir % 4;
  }

  //update the crrntRobotDir variable
  *rbtDir = (mOrientation[*indxDir].mDirection);

  //update LCD color
```

```
  lcd.setBacklight(mOrientation[*indxDir].color);
  lcd.setCursor(15, 0);
  lcd.print(*rbtDir);
}
int whereTo(const char cardinalDir, int tempToCell)
{
  /*  Going:
      East  --> +1
      South --> -4
      West  --> -1
      North --> +4
  */
  int toCell = tempToCell;
  switch (cardinalDir)
  {
    case 'N':
      toCell += 4;
      break;
    case 'E':
      toCell += 1;
      break;
    case 'S':
      toCell -= 4;
      break;
    case 'W':
      toCell -= 1;
      break;
  }

  //avoid going out of bounds
  if (toCell > 16)
  {
    toCell = tempToCell;
  } else if (toCell < 1)
  {
    toCell = tempToCell;
  }

  //print the cell that it's CURRENTLY in
  if (toCell >= 10)
  {
    lcd.setCursor(10, 1);
    lcd.print(toCell);
  } else {
    lcd.setCursor(10, 1);
    lcd.print(" ");
    lcd.print(toCell);
  }

  return toCell;


}
```

```
void updateCellStatus(int cellNumber, char status) {

  //char status is only one character to update the status

  switch (cellNumber) {
    case 1:
      lcd.setCursor(5, 1);
      lcd.print(status);
      grid[3][0] = '0';
      break;
    case 2:
      lcd.setCursor(6, 1);
      lcd.print(status);
      grid[3][1] = '0';
      break;
    case 3:
      lcd.setCursor(7, 1);
      lcd.print(status);
      grid[3][2] = '0';
      break;
    case 4:
      lcd.setCursor(8, 1);
      lcd.print(status);
      grid[3][3] = '0';
      break;
    case 5:
      lcd.setCursor(5, 0);
      lcd.print(status);
      grid[2][0] = '0';
      break;
    case 6:
      lcd.setCursor(6, 0);
      lcd.print(status);
      grid[2][1] = '0';
      break;
    case 7:
      lcd.setCursor(7, 0);
      lcd.print(status);
      grid[2][2] = '0';
      break;
    case 8:
      lcd.setCursor(8, 0);
      lcd.print(status);
      grid[2][3] = '0';
      break;
    case 9:
      lcd.setCursor(0, 1);
      lcd.print(status);
      grid[1][0] = '0';
      break;
    case 10:
      lcd.setCursor(1, 1);
      lcd.print(status);
      grid[1][1] = '0';
      break;
    case 11:
      lcd.setCursor(2, 1);
```

```
      lcd.print(status);
      grid[1][2] = '0';
      break;
    case 12:
      lcd.setCursor(3, 1);
      lcd.print(status);
      grid[1][3] = '0';
      break;
    case 13:
      lcd.setCursor(0, 0);
      lcd.print(status);
      grid[0][0] = '0';
      break;
    case 14:
      lcd.setCursor(1, 0);
      lcd.print(status);
      grid[0][1] = '0';
      break;
    case 15:
      lcd.setCursor(2, 0);
      lcd.print(status);
      grid[0][2] = '0';
      break;
    case 16:
      lcd.setCursor(3, 0);
      lcd.print(status);
      grid[0][3] = '0';
      break;
  }

  //print the cell that it JUST completed
  if (cellNumber >= 10)
  {
    lcd.setCursor(10, 0);
    lcd.print(cellNumber);
  } else {
    lcd.setCursor(10, 0);
    lcd.print(" ");
    lcd.print(cellNumber);
  }
}



void cnfgOrientations()
{
  //Red - Right
  //Green - Left
  //Blue - Up
  //Yellow - Down

  //north - UP
  mOrientation[0].mDirection = 'N';
  mOrientation[0].color = BLUE;

  //east - RIGHT
  mOrientation[1].mDirection = 'E';
```

```
  mOrientation[1].color = RED;

  //south - DOWN
  mOrientation[2].mDirection = 'S';
  mOrientation[2].color = YELLOW;

  //west - LEFT
  mOrientation[3].mDirection = 'W';
  mOrientation[3].color = GREEN;

  return;
}
```

Conclusion

This lab proved to be arduous and unintuitive. The variance in battery levels made it very hard to develop working code for the system. While the T.A. and the professor helped with techniques for the shortcomings, it would have been nice to know how to deal with such problems before exhausting all the other ideas. Otherwise, it was pleasurable being able to implement traversal functionality into a system, it is a novel concept and was very rewarding.