Control of Mobile Robotics
Spring 2016

Lab 4

Path Planning

2 May, 2016

Gerardo Sanchez
Adeoluwa Oyenusi

Task Description

The objective of this lab is to develop a path planning algorithm to navigate an arbitrary maze from any starting robot location and orientation. Given the starting position, the robot must be able to construct a valid path in order to get to the ending position/goal. Figure 1 depicts an example of the maze and a proposed path.

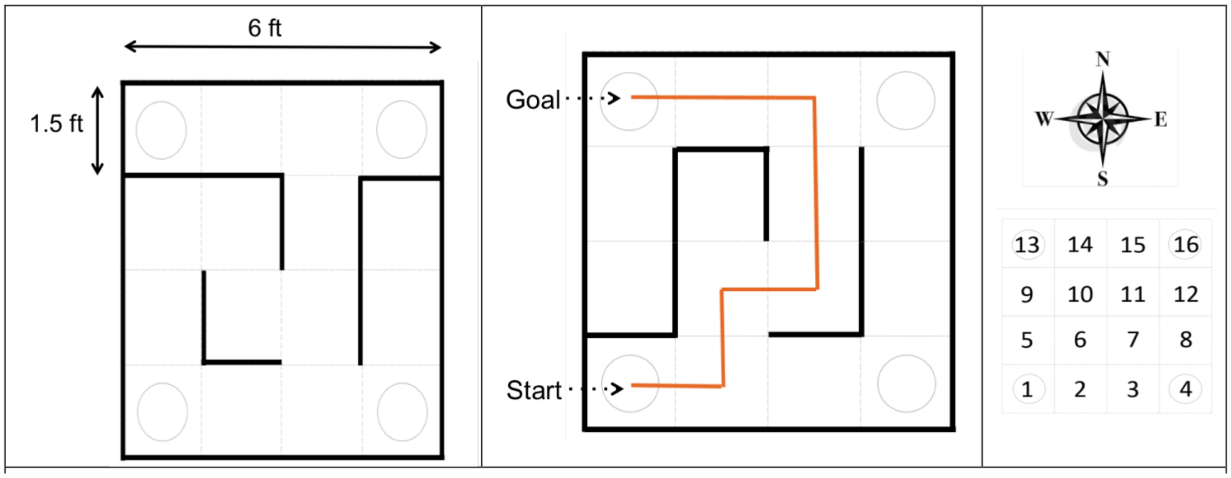

Figure 1 – Maze Structure

Path Planning

Video Link: https://youtu.be/n7NO8-AoZj4

To navigate a known maze, a path must be constructed and it will be done using the BFS graph traversal algorithm. The various maze configurations are shown in figure 2. As mentioned in the lab description, the shortest path is required and a BFS implementation will help achieve this. The maze will be represented as a Graph using an adjacency list. The adjacency list is configured at run time to reflect all valid movements in the maze. From this, the BFS algorithm will return the shortest possible path to reach the goal. The code description will go in depth with the specifications of the algorithm. In addition to this, the robot will need to be intelligent to leverage this information to move in accordance with the path. For example, the robot will need to acknowledge the proposed orientation and it will need to assert it is in the desired orientation before moving. In addition to the path planning, the robot must be able to leverage this data to travel in the physical world. To achieve this, wall following is integrated if there are left walls that allow it. If there is no immediate left wall, it will blindly travel forward in half-cell increments until the goal is finished. The presence of walls will allow the robot to error correct when offsets are introduced due to battery levels and slippage. Figure 3 is a visual representation of the critical part of the path planning algorithm.

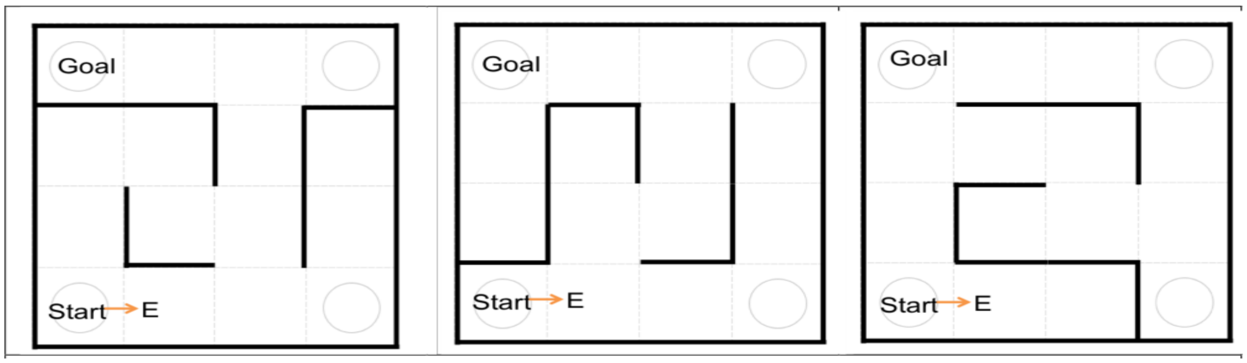

Figure 2 – Maze configurations for Execution

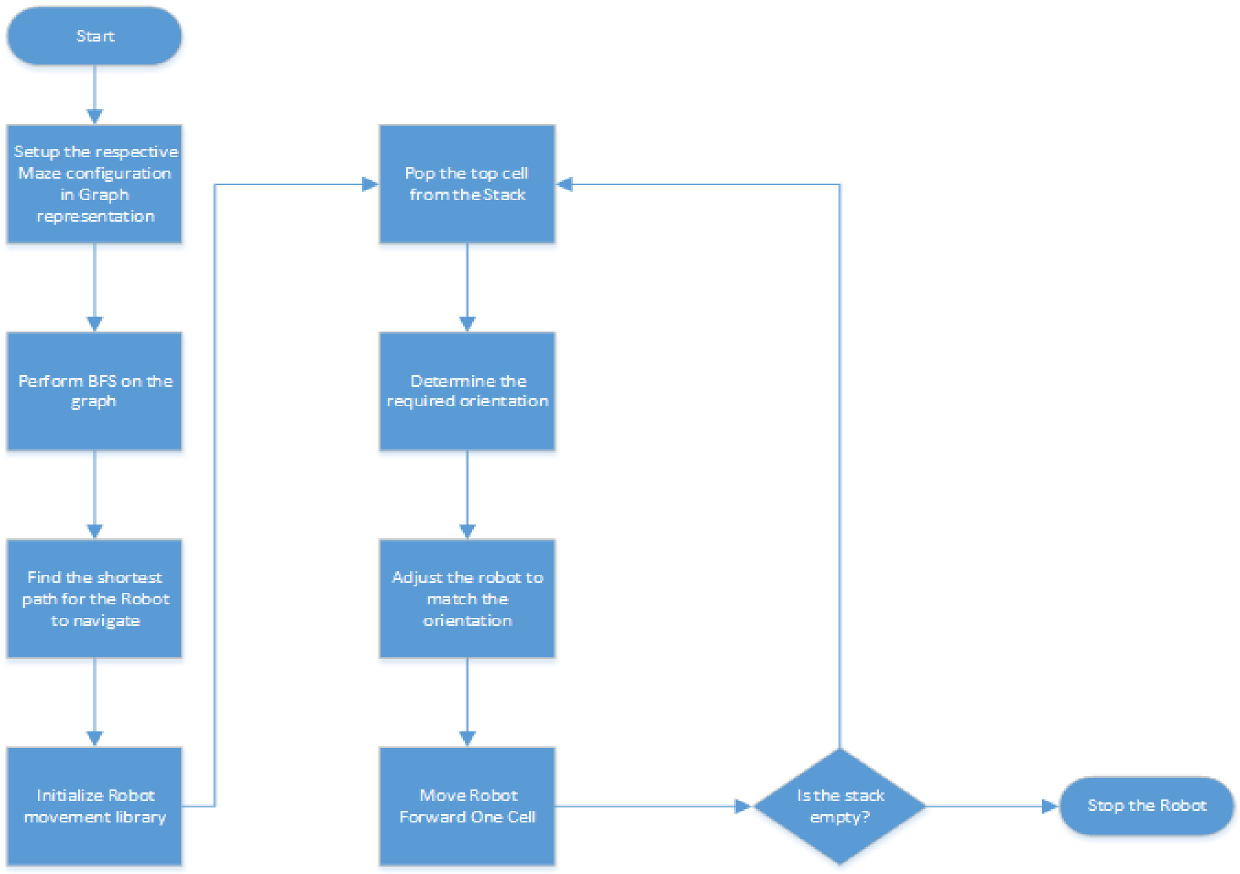
Start
Setup the respective Maze configuration in Graph representation
Perform BFS on the graph
Find the shortest path for the Robot to navigate
Initialize Robot movement library
Pop the top cell from the Stack
Determine the required orientation
Adjust the robot to match the orientation
Move Robot Forward One Cell
Is the stack empty?
Stop the Robot

Figure 3 – Visual Path Planning Algorithm

Code Description

Path Planning

Breadth First Search Algorithm & Path Setup

The breadth first search algorithm simply finds the shortest path from the start to the goal. We analyze all the neighboring cells of a cell and keep doing this until it finds the goal. We use a parent array to find the cell prior to the current cell. The functions setPath() puts the path to the goal in a stack. The stack will be popped for the robot to navigate to the goal.

```
void Graph::breadthFirstSearch(int u,int exit)
{
    //color[u] = GRAY;
    color[u] = 1;
    q.push(u);
    while(q.empty() != true)
    {
        u = q.front();
        q.pop();
        for (int i = 0; i < 4; i++)
        {
            if(color[mazeCell[u].neighbors[i]] == 0 &&
mazeCell[u].neighbors[i] != 0) //to handle values that 0 in the array WHITE
            {
                color[mazeCell[u].neighbors[i]] = 1;//GRAY
                parent[mazeCell[u].neighbors[i]] = u;
                if (mazeCell[u].neighbors[i] == exit)
                    return;
                else
                    q.push(mazeCell[u].neighbors[i]);
            }
        }

    }
}


void Graph::setPath()
{
    int temp = exit;//exit
    //populates the stack from the parent array
    while (!(temp == entry))//start
    {
        hold.push(temp);
        temp = parent[temp];
    }

}

stack<int> Graph::getPath()
{
    return hold;
}
```

Got To Goal

Our goToGoal() function takes in the starting cell, the path to the goal, and the current orientation of the robot as its parameters. We keep popping the stack and move from our current cell to the cell that we want to go to. Since the robot has the ability to face different directions, we needed a way to track the orientation of the robot and find how the current cell relates to the next cell. We use the function adjRbtDir() to handle this. Furthermore, this function also moves the robot to match the intended orientation. For example, if the robot is currently facing east and needs to move to a cell north, it has to first turn left, before starting to move forward.

```
void goToGoal(int startCell, stack<int> tempPath, char
myDirection)//navigation to the goal
{
  int currentCell = startCell;
  int nextCell;
  char orientation = 'N';
  int wfThresh = 110;

  //int frwdTime = 3800;
  int halfTime = 1800;
  lcd.clear();
  bool isfirstTime = true;
  bool islastTime = tempPath.empty();

  //start in correct Dir
  orientation = getOrientation(currentCell,nextCell);
//  adjRbtDir(orientation, myDirection);

  while(!(islastTime))
  {
    lcd.setBacklight(WHITE);
    nextCell = tempPath.top();
    tempPath.pop();
    islastTime = tempPath.empty();
    /*
            New changes 30 Apr 2016
        */

    for(int i = 0; i < 2; ++i)
    {
      if(isfirstTime)
      {
        isfirstTime = false;
        i = 1;
      }

      if(i == 1)
      {
        lcd.setCursor(8,0);
        lcd.print("    ");
        //adj rbt orientation in the middle of the cell
        //obtain the direction and assert the robot is in that direction
        orientation = getOrientation(currentCell,nextCell);
        adjRbtDir(orientation, myDirection);
        printLCD(currentCell, myDirection, nextCell, orientation);
        //myMover.cmd_vel(90,90);
      }
            shrtL = trueDist(SLSensor, 10);
          if(shrtL < 8) // we think there is a wall here
          {
                //WALL FOLLOW for half a cell and retry
            int cellCntr = 0;
            int wfCntr = 0;
        lcd.setBacklight(RED);
        lcd.setCursor(13, 0);
        lcd.print("WF");
        delay(500);
            while(cellCntr < wfThresh)
            {
          printVal(cellCntr);
                    shrtL = trueDist(SLSensor, 10);
              myMover.wallFollow(&wfCntr, &cellCntr, shrtL);
              ++cellCntr;
            }

          }else if(shrtL > 8)
          {
        lcd.setBacklight(BLUE);
        lcd.setCursor(13,0);
        lcd.print("BL");
        delay(500);
        myMover.moveForwardALittle(halfTime);
          }
     myMover.stopRobot();
     myMover.moveForwardALittle(200);
    }

    lcd.setCursor(8, 0);
    lcd.print("NC!");
    lcd.setCursor(13,0);
    lcd.print("  ");
    currentCell = nextCell;
    myMover.stopRobot();

        /*********************
            End new changes 30 Apr 2016
        */
  }

  //move to the last cell
  shrtL = trueDist(SLSensor, 10);
  if(shrtL <= 8) // we think there is a wall here
  {
    //WALL FOLLOW for half a cell and retry
    int cellCntr = 0;
    int wfCntr = 0;
    lcd.setBacklight(RED);
    lcd.setCursor(13, 0);
    lcd.print("WF");
    delay(500);
    while(cellCntr < wfThresh)
    {
      printVal(cellCntr);
```

```
      shrtL = trueDist(SLSensor, 10);
      myMover.wallFollow(&wfCntr, &cellCntr, shrtL);
      ++cellCntr;
    }
  }else if(shrtL > 8)
  {
    lcd.setBacklight(BLUE);
    lcd.setCursor(13,0);
    lcd.print("BL");
    delay(500);
    myMover.moveForwardALittle(halfTime);
  }
  myMover.moveForwardALittle(300);
  myMover.cmd_vel(90,90);
  lcd.clear();
  lcd.setBacklight(GREEN);
  lcd.print("Done. Suckas.");
  while(1);
}

void adjRbtDir(char orientation, char& myDirection)
{
  int leftDelay = 900;
  int rightDelay = 900;
  //lcd.clear();
  lcd.setCursor(12,1);
  lcd.print(myDirection);
  lcd.print("->");
  lcd.print(orientation);
  lcd.setBacklight(YELLOW);
  delay(500);
  if(!(orientation == myDirection))
    {
      myMover.stopRobot();
      switch(myDirection){
        case 'N':
          if(orientation == 'S') { for(int i = 0; i < 2; ++i)
myMover.rightTurn(rightDelay);}
          if(orientation == 'E') myMover.rightTurn(rightDelay);
          if(orientation == 'W')
          {
            lcd.setBacklight(VIOLET);
            myMover.leftTurn(leftDelay);
          }
          break;
        case 'E':
          if(orientation == 'N') myMover.leftTurn(leftDelay);
          if(orientation == 'S') myMover.rightTurn(rightDelay);
          if(orientation == 'W') {for(int i = 0; i < 2; ++i)
myMover.rightTurn(rightDelay); }
          break;
        case 'S':
          if(orientation == 'W') myMover.rightTurn(rightDelay);
          if(orientation == 'N') {for(int i = 0; i < 2; ++i)
myMover.rightTurn(rightDelay); }
          if(orientation == 'E') myMover.leftTurn(leftDelay);
          break;
        case 'W':
          if(orientation == 'N') myMover.rightTurn(rightDelay);
          if(orientation == 'E') {for(int i = 0; i < 2; ++i)
myMover.rightTurn(rightDelay);}
          if(orientation == 'S') myMover.leftTurn(leftDelay);
          break;
      }
    }
    myMover.cmd_vel(90,90);
    myDirection = orientation;
}
```

Conclusion

After experiencing lab 3, the expectations for the work required were clear. We did much better with debugging due to the variance in battery levels. In addition, the restructuring of code into modular units made integration very seamless. This class was a great experience and it showed the true mesh of hardware with software. A true engineer will be able to accommodate for hardware shortcoming with the use of clever software techniques. We would like to give thanks to Dr. Weitzenfeld and Juan for their guidance throughout the course of these assignments.