Exploring the Maze with Pioneer 3DX

劉彥均, 王鈞奕, 馬凱傑

National Taiwan University, Dept. of CSIE

Taipei, Taiwan, R.O.C.

Abstract

In this project, we construct a system concentrating on maze exploring, map making, planning, and wall avoidance using Pioneer 3DX. Given a simple maze without loops (for simplicity). The robot is able to travel all possible roads or forks until it gets to the end. After reaching the end for the first time, the robot would memorize the map and do path planning, so, for the second, it would go to the end point without going to the wrong way. Besides, whenever the robot is traveling in the maze, it would constantly correct its direction so as not to hit the wall.

I. Introduction

About this final project, we first search the video about robots on the internet to decide our topic. We find some amazing videos about robots, but we think it's too difficult for us. However, we find an interesting video about the MicroMouse exploring a maze. We thought it is an interesting topic and we have the ability to accomplish this topic. And then, there are four Pioneers for us to do our final project. Therefore, we can use Pioneer as the MicroMouse. By this way, we decide our topic, exploring the maze by pioneer, for the final project of this course.

II. System Overview

A. Main Control System

Controlling the robot's velocity, rotational velocity, direction according to some criteria, for example, going straight, making a turn, avoidance of hitting the wall, and so on.

B. Map Making and Path Planning System

Considering the maze we are working on, we decide to use topological map to represent it. For each turning point (except for dead roads), we create a node with 4 links (one for each direction) linking to the other nodes. Hence, all nodes form a topological map. After the completion of the map, we would apply our path planning to it to find the most efficient path. The details of the map making and path planning will be discussed in the later Methodology section.

C. Robot Vision System

Because we have to find a method to decide whether the Pioneer reaches the goal, we decide to use the camera, which is connected to the computer as the vision of Pioneer. Besides, we use a red object as the goal of our maze. Therefore, when the Pioneer walks to the goal of our maze, the camera will see the red object and the computer will let the Pioneer to stop. This is the only thing that our robot vision does.

D. Sensing System

For robots, sensing is critically important. In this project, we use Pioneer 3DX's built-in sonar as our sensing device. We divided the sensing area into three parts, front, left, right. For each area, the sonar system returns the range to the nearest obstacle in the area. If the reading is higher than 5000, it means it doesn't sense anything in the feasible area of the sonar.

III. Methodology

For the map exploring part, we first construct a topological map while exploring the map. Whenever the robot makes turns and the point has not been visited before, the robot will add a new node on the turning point, which is connected to the previous node with directions labeled.

A. Fundamental Robot Control Behaviors

"Turn Left", "Turn Right", "Turn Around" are the basic behaviors in our system. We use "stop and turn" concept to implement these behaviors, and the 90-degree turning is implemented by using the Pioneer-built-in "setHeading()" function.

B. Map Making

As mentioned before, the map is represented by topological map. The detailed idea is to construct a 4-direction linked list data structure.

Node Information：

1. Number
2. X, Y coordinates
3. 4 directional node pointers
4. Several Boolean flags

Node Creating Procedures：

1. In the beginning of the maze exploring, we create the first node, marking it as the "starting node" and "previous node".
2. Once we encounter a fork, we create another node and link it with the "previous node" with the direction according to the robot's current heading direction and, remarking the newly created node as the "previous node".
3. Repeat 2 until the robot reaches the end and stops.

C. Fork Road Detection:

We use sonar to measure the distance in front, on right side, and on left side. When the increasing distance on left side or right side is larger than a certain value (we set the value to be 300, which is 30 cm), then the fork road occur. If the robot doesn't encounter fork road, it will go straight.

D. DFS Search:

Like the original DFS search, the robot will search the fork road and its branch first. To remember the road the robot has traversed before, and not to construct the node at the same point twice, we create several flags to aid the robot accomplish DSF search. The detail about the flags is listed as below.

1. Dead road flag：

To remember the road visited before is a "dead road", we create four boolean variables in the node structure (northdead, westdead, southdead, eastdead), and label the dead roads by label_deadroad(NodePtr sPtr, int angle) function. Therefore, the robot will not travel the dead road twice by checking the values of the four flags.

2. deadroad_occur flag：

This is a flag to help the robot make different decisions when exploring the maze

i. The timing to change the value of deadroad_occur

At first, this flag is set to be false.When the robot encounter a dead road, we set deadroad_occur to be true.

When deadroad_occur is true and the robot found new branches it hasn't visited before, we set deadroad_occur to be false.

We set the deadroad_occur flag to be true after the robot encountered a dead road and turned around, that is, there is no road either in front, or on the left or right side.

ii. When deadroad_occur flag is false

Make turns and create nodes whenever the robot detects a new branch. Go on exploring the maze.

iii. When deadroad_occur flag is true

When the robot goes back to the previous node, it has to consider two conditions:

Condition 1：

If there exist some other roads except the source road and is not a dead road, cancel deadroad_occur flag and go on to ii. Part.

Condition 2：

If there doesn't exist any road except source road and dead road, go back to the source road and label this road as dead road. deadroad_occur flag is still true.

E. Path Planning

Before the beginning of the second round, we would do path planning for the robot on the map built previously. The basic idea is to use Depth First Search to traverse the map.

Path Planning Procedures：

1. First check if we reach the end by the number of the node we are at right now. If yes, set the global "found flag" true, save the current node information in a global list array and return, else, do "Path Planning Procedures" on the next node by the order of east, south, west, north.
2. At this point, the neighboring nodes have been traversed, if the "found flag" is true, save the current node information in the list array, else, simply return.

After these procedures, the global list array is the path information we expect to have.

Note：In order to preventing loops in the recursive procedures, we mark it as "checked" to indicate we've been to this node before.

F. Wall Avoidance Technique

Because the accumulated errors due to every motion like "Turn Left" or "Turn Right", the robot would be more likely to hit the walls after some time. Therefore we have to apply the technique to make the robot run smoothly.

Wall Avoidance Procedures：

1. Use the readings provided by the sonar sensing system, if the reading is below a threshold value (indicating the robot is too close to the wall), make a slight turn by differentiating the speeds of the two wheels, else, do nothing and return.
2. Since the previous turn might cause the robot to head to the wrong direction, we have to correct it. In order to correct it, this procedure is also implemented by the Pioneer-built-in "setHeading()" function.

G. Determining whether Reaching the Goal

At this point, the neighboring nodes have been traversed, if the "found flag" is true, save the current node information in the list array, else, simply return. In this part, we use the image captured by camera to determining whether the Pioneer reach the goal of our maze. Because we use the red object as the goal of our maze, we need to distinguish between the red pixels and the other pixels in the image. Therefore, we use the "openCV" as our tool to process the image. However, instead of reading the whole image's pixels, we only read the pixels in the central area of the image. Because if we read all the pixels to deciding whether there is a red object, it will see the red picture on the wall of our maze as the goal. Then, the Pioneer will stop. As a result, we only read the central area of the image, because we can roughly see it as the road in the maze. By this way, when the Pioneer exploring the maze, if it see a red object on its way, it will stop. Besides, in order to avoid some reading error, we set a threshold to decide whether it is a red object. If the number of red pixels more than 20, we can see it as we find the goal.

H. Determining between Red Pixels and Others

As a result of using red object as the goal, we need to decide whether a pixel is red or not. We use the method of turning RGB to HSI. For every pixel, we can use the value of RGB and a special function to turn it into HSI. The symbol H stands for the color in our world. And we have already know that the H value of red pixels is approximately from 345 to 360. Therefore, if we use RGB of certain pixel to get a value H from 345 to 360 by the function, we can see it as red pixel. And the function is as follow:

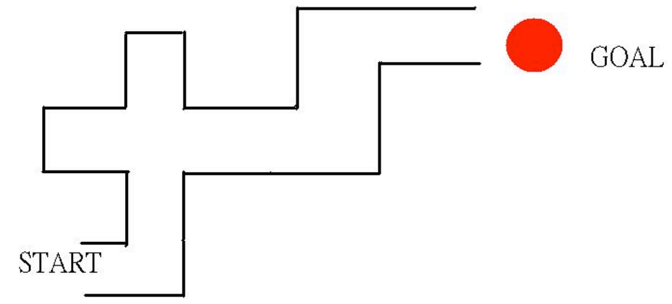
$$h = \cos^{-1}\left\{\frac{0.5\cdot\left[(r-g)+(r-b)\right]}{\left[(r-g)^2+(r-b)(g-b)\right]^{1/2}}\right\} \qquad h \in [0,\pi] \text{ for } b \le g$$

$$h = 2\pi - \cos^{-1}\left\{\frac{0.5\cdot\left[(r-g)+(r-b)\right]}{\left[(r-g)^2+(r-b)(g-b)\right]^{1/2}}\right\} \qquad h \in [\pi,2\pi] \text{ for } b > g$$

The RGB value can be derived from the tool "openCV". By this way, we can use this to determining whether reaching the goal.

IV. Simulation And Experiment

This following image is one of the mazes we have experimented on.



First, we put the Pioneer at the START of the maze and then start our program. It will go straight until it encounters the first fork. At the first fork, because turning left is the only way the Pioneer can go, it will absolutely turn left. Then, it will encounter next fork after going a while. At this fork, it may turn right or left. If it turns right first, it can reach the goal without any errors. However, if it turns left first, it will get the correct way by doing several "trial and error". It will encounter the dead road in the left way. Then, it turns around and goes to the same fork again. And it will turn left again to go to the front way of the fork. Then, it will encounter the dead road again. It turns around again and goes to the same fork again. This time it turns left to go to the right way of the fork because the other ways, except for the source way, are labeled dead. Then, it can see the red object by turning left and turning right. After finding the red object, it will see it as the goal and then stop. Then, we can put the Pioneer to the start again. It will find the goal smoothly without "trial and error". It will turn left, turn right, turn left, and turn right in sequence. After these actions, the program will stop.

V. Conclusion

This is our first attempt to use a robot to finish a specific task. Therefore, we set several rules on the model of environment to make the task easier. For example, we assume the distance between the robot and the wall of the maze is within a certain range, and we assume the maze is well-formed and won't have holes or chinks on the wall. Those rules make it easier for us to accomplish our goal.

Still, there exist some more for us to improve. We list them as follows

A. Speed of the Robot

Now we adopt a certain constant speed for the robot to explore the map. However, to make the robot explore the map in a more efficient way, we should let the robot dash if there is no obstacle in front and we should brake the robot whenever the robot has to make turns.

B. Better Distance Measured Device

Now we us sonar sensor for distance measuring, but there are some disadvantages for using sonar, like it can only measure short distance, and error might occur because of reflection. It will be better to use laser sensor.

C. Loop Situation in a Maze

We haven't handled the loop situation. However, we have already come up with a possible solution. Since we have already record the x, y coordinate values when we construct the node, we can check if the node we construct is visited before. The difficulty is that the x, y values may not be accurate because of accumulated error resulted from friction. Therefore, we have to use some sensor to help us locate the robot to fix these errors.

Acknowledgment

For this project, we would like to thank our teacher assistants, 黃國楨 and 曾士桓, for providing the Pioneer 3DX, camera and several tools for building the maze and their enthusiasm for helping us. Besides, we'd like to thank our teacher, professor 傅立成, for his passion in teaching us.

References

[1] William K. Pratt, Digital Image Processing, 4th, Wiley-Interscience,2007.

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, Introduction to Algorithms, 3rd, The MIT Press, 2009.

[3] Robin R. Murphy, An Introduction to AI Robotics, 1st, The MIT Press, 2000.

Job Distribution

A. 劉彥均：Robot Vision, Image Processing

B. 王鈞奕：Maze Exploring

C. 馬凱傑：Fundamental Robot Control Behaviors, Map Making, Wall Avoidance

Our jobs are well divided!