VOL. 13, NO. 8, APRIL 2018 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences

www.arpnjournals.com

DIJKSTRA ALGORITHM BASED INTELLIGENT PATH PLANNING WITH TOPOLOGICAL MAP AND WIRELESS COMMUNICATION

Lyle Parungao[1], Fabian Hein[2] and Wansu Lim[3]
[1]School of Electronics Engineering, Mapúa Institute of Technology, Philippines
[2]Department of Software Engineering, University Heilbronn, Germany
[3]Department of Information and Technology Convergence, Kumoh National Institute of Technology, South Korea
Email: wansu.lim@kumoh.ac.kr

ABSTRACT

In this paper, an idea of intelligent path planning was introduced. Using information received from a server that is transmitted through a wireless communication, the data is processed to edit a provided topological map, with which a shortest path calculation will be executed. Information about the destination and blocked road parts is sent to a mobile robot using a wireless ad hoc network communication. The mobile robot will process this information in a topological 2D-array map and ignore the blocked parts of the road for the shortest path calculation based on Dijkstra's algorithm. Aside from the intelligent path planning, an automated driving algorithm was also implemented using infra-red sensors installed on the mobile robot to navigate the robot to its destination.

Keywords: general purpose input outputs, infra-red sensors, transmitter / receiver exchange.

1. INTRODUCTION

Automated delivering robots inside the factories, streets, and buildings are becoming more popular because of the convenience that it gives [1]. To increase the robots' functionality, intelligent path planning is being incorporated. There are components that need to be considered in intelligent path planning.

The first component is the algorithm that calculates the shortest path to have an efficient path planning. There are a number of varieties of algorithm that can be used. One of them is Dijkstra's algorithm. Dijkstra's algorithm is used to calculate the shortest path between nodes in a graph and is from the class of the Greedy-algorithms which is an algorithmic paradigm that pursue problem solving that are making optimal choices in finding feasible solutions. There are a lot of different variants of the Dijkstra's algorithm that exist. Most of these variants use fixed nodes as their source in finding the shortest path and produce a shortest path tree [2] [3] [4] [5].

The second component is wireless communication. It is the way of imparting or exchanging information in the system and is where the wireless ad hoc network comes in. Ad hoc network will allow the Raspberry Pi shield to communicate wirelessly through Wi-Fi. Wireless ad hoc networks are decentralized networks that based its specification for a suite of high-level communication protocols to create personal area networks and are usually used in small scale projects which needs wireless connection [6] [7] [8]. Bluetooth and ZigBeecan also be used; however, wireless ad hoc networks are less complex and uses short range low data wireless communication system that suites well with small scale projects.

The third component to consider is the map. Factories, streets, and buildings usually have floor plans or routes. Maps can be used to make a representation of areas for small scale projects to have an overview on how mobile delivering robots can navigate through its destination. A topological map is a simplified type of diagram that contains only vital information. Information that is unnecessary is not considered since it is just a scalable representation of a map. It is easy to modify but the relationship between coordinate points remains constant. Modifications have to be done because of the ever-changing situations due to traffic [3] [4] [9] [10].

The last component to consider is navigation. Infra-red sensors (IR sensors) are the one responsible for that since it uses specific light sensor to detect distance of a light that reflects because of an obstacle that depends on the IR value. The IR sensors are used to avoid these obstacles and helps in the maneuvering of the mobile robot [11].

Nowadays, a lot of factories use automatic driving robots to deliver components and tools around the factory hall to provide the workers with the equipment they need. Mobile robots often have to dodge into 'parking areas' on the side of a route to avoid a collision with other robots which are on the same route as them [1] [12]. Based on [12], this paper wants to consider on how to improve the idea of these automatic driving robots.

In our case the robots will avoid blocked parts on their route and find another way around them right before they depart. Blocked parts can arise because of traffic jam or obstacles. Hence, considering upon meeting a robot on a chosen path it is not inevitably considered as a blocked part since other routes have even more traffic on them. Because the robots would not need to wait in the traffic or for other robots to pass and in conclusion to that they will save time during their route before arriving at the destination. By introducing wireless ad hoc networks, we can communicate with the mobile robot and send the robot its destination as well as information about sections of the map which are already blocked. By improving Dijkstra's algorithm using wireless communication, it will be possible to avoid highly frequented routes even though it would be the shortest route. Furthermore, blocked parts on the map will be considered in the shortest path calculation,

so that they can be avoided and in determining blocked parts it will be send through wireless communication coming from the server since the server has an overview about all the robots that takes a certain path. Topological map which is used for the calculation and navigation is implemented in the mobile robot. In addition to that IR sensors on the mobile robot are used to navigate. To evaluate the efficiency of our system, time based results will be acquired for our data. The server is not included in the implementation of this paper.

2. INTELLIGENT PATH PLANNING

A. Overview

The wireless communication between the server and the mobile robot uses a Wi-Fi based Wireless ad hoc network. To calculate the shortest path, while using intelligent path planning for avoiding blocked parts on the road, the Dijkstra's algorithm is used and an editable topological map is provided in the memory of the mobile robot. Furthermore, to provide a reliable automated driving algorithm the mobile robot is equipped with six IR-sensors, from which three are at the front bumper, one at the rear bumper and one on each side of the mobile robot.

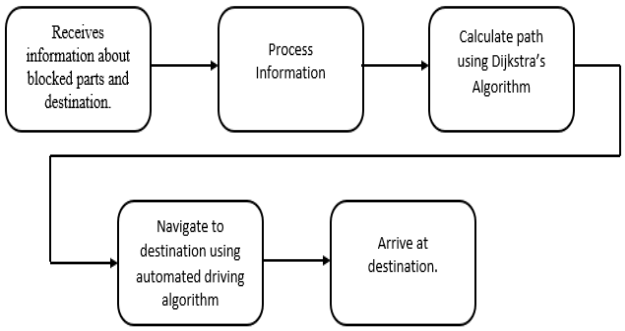

Figure-1. Block diagram of the general sequence of events for the mobile robot.

The last thing we have to be aware of, is implementing the software, which combines all of these mentioned parts and create like this the whole functional system of our proposed idea (Figure-1).

The following sections show every single aspect mentioned in the overview in detail and how they are combined to provide the intelligent path planning idea.

B. Wireless communication

The first step for implementing the intelligent path planning is to provide the wireless communication between the mobile robot and a server. It is done using wireless ad hoc network technology. Ad hoc network does not rely on pre-existing infrastructures such as routers because it serves as a node to node connection between two interfaces. Forwarding data is done through accessing the remote desktop connection of the ad hoc interface of the mobile robot with the server. Therefore, it allows the server to make some changes and commands to the robot without the need of configuring complex infrastructures and it just enables to build a joined network with self-configuring.

C. Blocked parts on the topological map

To avoid highly frequented routes and find the fastest way to the mobile robot's destination, two important information are sent by the server to the mobile robot through wireless communication using wireless ad hoc network and the information is subsequently processed by the mobile robot. One information contains the final destination of our mobile robot, given in coordinates. With a topological 2D-array map provided in the code of the mobile robot, it is capable to know exactly where its destination is. The second important information is the blocked road parts. The blocked parts on the road are going to be transmitted the same way as the mobile robot's destination - through coordinates. With the given coordinates of the blocked parts on the road, the topological map gets modified, so that the blocked parts are present in the map.

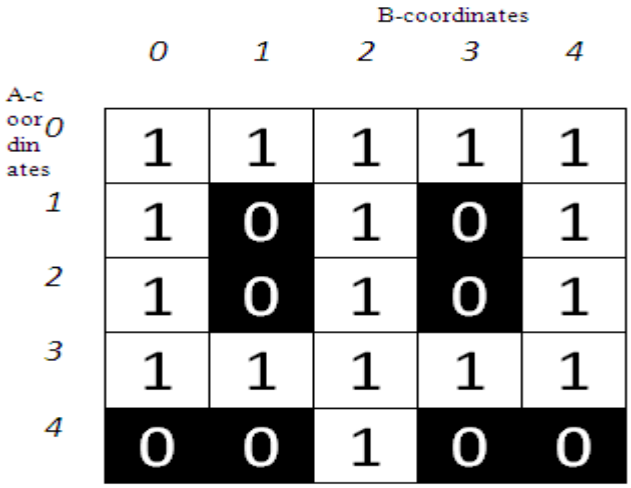

Figure-2. Proposed topological map.

Figure-2 illustrates the idea of the topological map. Every field can be described through a-coordinates and b-coordinates with the syntax '(a,b)' as displayed on the sides of the illustration. If a field contains a '0', it means that there is a wall or an obstacle. Otherwise if it's a '1', it is considered as free space. The start point of the mobile robot is at the coordinate '(4, 2)'.The implementation of the topological map in this example is a 2D-array and looks like this:

Example: Implementation of 2D-array

```
1. int map[5][5] = {{1, 1, 1, 1, 1},
2.                  {1, 0, 1, 0, 1},
3.                  {1, 0, 1, 0, 1},
4.                  {1, 1, 1, 1, 1},
5.                  {0, 0, 1, 0, 0}};
6. //The start point which is at '(4,2)' can be accessed with:
7. map[4][2];
```

A-coordinates \ B-coordinates	0	1	2	3	4
0	(0,0)	(0,1)	(0,2)	(0,3)	(0,4)
1	(1,0)	(1,1)	(1,2)	(1,3)	(1,4)
2	(2,0)	(2,1)	(2,2)	(2,3)	(2,4)
3	(3,0)	(3,1)	(3,2)	(3,3)	(3,4)
4	(4,0)	(4,1)	(4,2)	(4,3)	(4,4)

Figure-3. Visualization of the coordinates.

Figure-3 points up how the coordinates are used for the fields. The first subscript is the column on the left side. The second subscript is the row on top of the illustration.

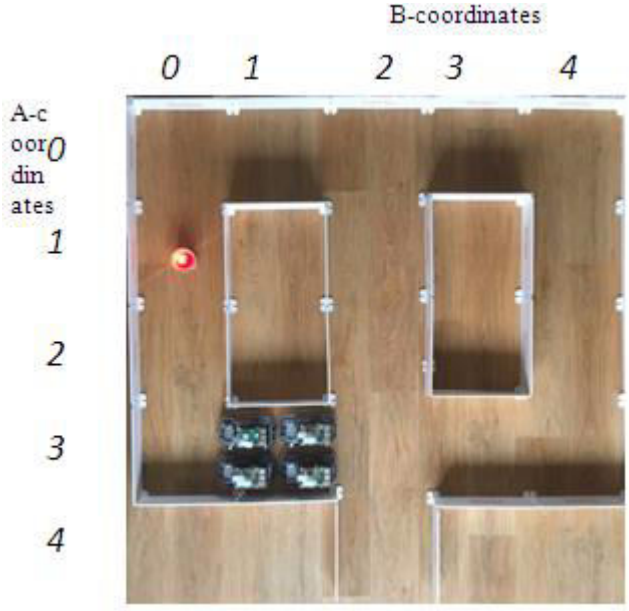

Figure-4. Actual photo of the map.

Figure-4 shows an actual photo of our example map. In this case, we added a physical implementation of a blocked part, using four mobile robots. Furthermore, a red/yellow dot represents the destination in this photo.

The coordinates for the mobile robot's destination and the blocked road parts are transmitted through coordinates using wireless communication coming from the server. In our proposed idea, the server is responsible for the calculation which road parts are already used, meaning blocked.

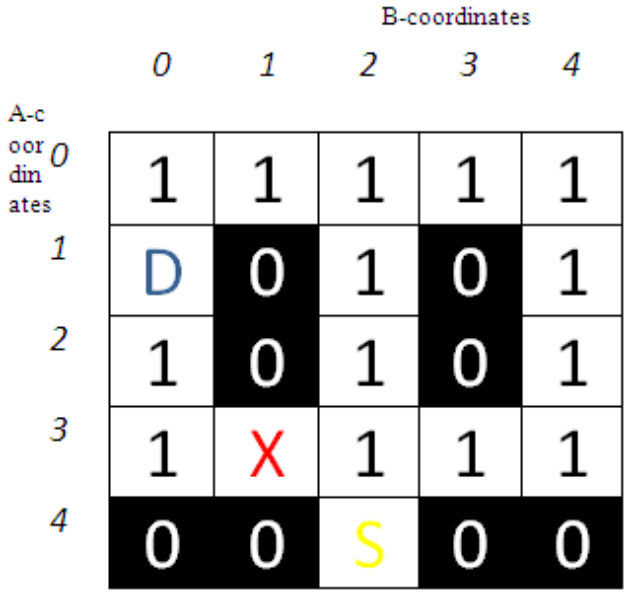

Figure-5. Proposed topological map include transmitted details.

In this example (Figure-5) the blocked part and the destination of the actual photo of Figure-4 are used. The destination is '(1, 0)', marked with 'D' and the blocked road part is at '(3, 1)' marked with 'X'. Furthermore, 'S' represents the mobile robots start point.

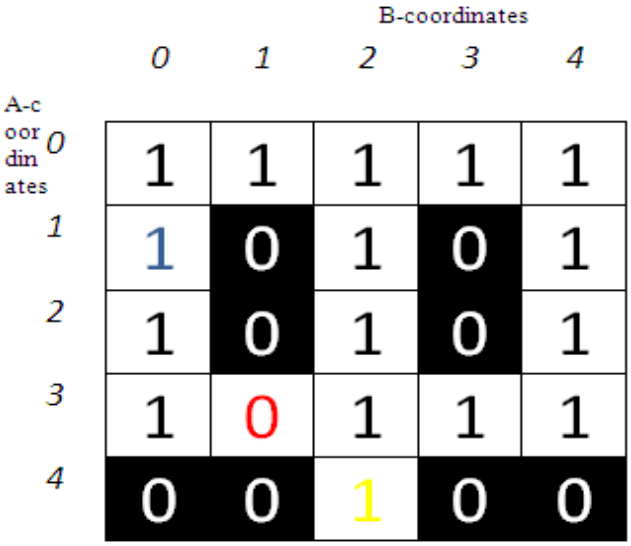

Figure-6. Changed proposed topological map because of blocked part.

Because of the blocked road part at '(3, 1)' the '1' of the field is changed to a '0' (Figure-6). The field is now treated as a wall/obstacle. This change is made in the topological map representation (2D-array) by the code.

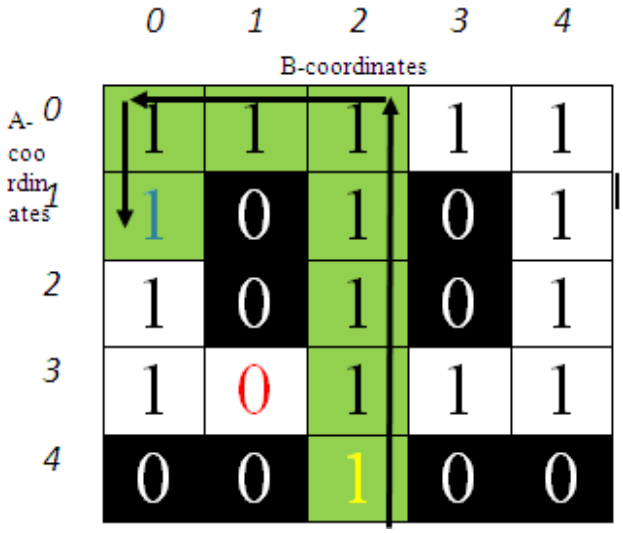

Figure-7. Chosen route on the topological map.

The upcoming calculation for the shortest routeto the mobile robot's destination, which is executed by the Dijkstra's algorithm will now no longer considering the blocked parts on the road. Figure-7 shows the example of the calculated route via the Dijkstra's algorithm. The green path is the route the mobile robot will take and the black arrows display its direction from the start point to the destination.

D. Dijkstra's shortest path algorithm

The Dijkstra's algorithm is an algorithm that determines shortest distance in a given path while considering the least effort or lowest cost of the distance between the starting points to the destination point. Dijkstra's uses nodes in its calculation and saves the points where the cost of distance is low. It will continue it calculation to the shortest path beginning from a starting point, and then exclude routes that has high cost of distance when making an update.

Dijkstra's will use "infinite" for the routes that have never been explored and the starting point will be marked "0". Calculation of paths through neighbor nodes will be considered temporary distances if it has not yet been finalized. If calculated distance of a node is smaller than the current one, Dijkstra's will update its main calculations. Nodes that have been pick as shortest distance from another node will be marked permanent so that it will be stored in the central idea of the algorithm.

Algorithm 1 Pseudo code for Dijkstra's Algorithm

```
1.  function Dijkstra(Graph, source):
2.  dist[source]  := 0 // Distance from source to source
3.  for each vertex v in Graph: // Initializations
4.  if v ≠ source
5.  dist[v] := infinity // Unknown distance function
    from source to v
6.  previous[v]  := undefined // Previous node in
    optimal path
7.  end if //from source
8.  add v to Q   // All nodes initially in Q
9.  end for
10. while Q is not empty: // The main loop
11. u := vertex in Q with min dist[u]  // Source node in
    first case
12. remove u from Q
13. for each neighbor v of u:   //where v has not yet
    been removed from Q
14. alt := dist[u] + length(u, v)
15. if alt < dist[v]: // A shorter path to v has been found
16. dist[v]  := alt
17.  previous[v]  := u
18. end if
19. end for
20. end while
21. return dist[], previous[]
22. end function
```

This pseudo code can be used as a reference for coding in any higher-level programming language.

E. Automated driving algorithm

After the route calculation is done, the mobile robot has to drive on the calculated route to its destination. To provide functional and usable automated driving algorithm distance detection via the on-board IR-sensors is used. The algorithm avoids walls and obstacles. While driving forward, the algorithm always pays attention to upcoming walls on the sides or in front. In reaction of the walls, the mobile robot will change its lane to avoid these. Furthermore, if the IR-sensors recognizes a diversion as portrayed in the topological map, it will turn to its right track which will lead to the mobile robot's destination. Refer to "IR SENSOR & SWITCH" figure from Raspberry Pi ALTINO Programming Manual page 20.

F. Code-architecture

Before starting to drive using the automated driving algorithm, the mobile robot waits in standby mode until it receives data through the wireless ad hoc network communication. The received data is going to be read and analyzed by the code. The submitted information is necessary for the intelligent path planning algorithm to avoid the blocked road parts. It contains also the mobile robot's destination to reach. With the combination of the intelligent path planning algorithm and the Dijkstra's algorithm, it is now possible to find the shortest route to the mobile robot's destination while avoiding highly frequented routes. For the Dijkstra's calculation, we will provide a Dijkstra's Python script which needs the start and destination coordination as well as the topological 2D-array map. As a return of the executed Python script, the main code will receive the calculated shortest path through a char array of several coordinates.

Example: Execute function

```
1. startCo = {4,2};
2. destCo = {1,0};
3. map2D[5][5] = {{1, 1, 1, 1, 1},
4. {1, 0, 1, 0, 1},
5. {1, 0, 1, 0, 1},
6.  {1, 0, 1, 1, 1},
7.  {0, 0, 1, 0, 0}};
8. PyObject_CallObject(djikstra,map2D,startCo,destCo)

9. //the received return value looks like this:
10. route = "10,00,01,02,12,22,32,42";
```

The next step after the shortest path calculation through the Dijkstra's algorithm is the navigation to the destination. With the calculated road, the mobile robot can navigate to its destination using the on-board IR-sensors. Like that, it is possible for the mobile robot to recognize and avoid walls or obstacles on its way. The whole navigation is working with an automatic-driving algorithm. In association with the provided topological map and the calculated path, the mobile robot will know when and in which direction it is necessary to turn using a navigation algorithm. After arriving its destination, the mobile robot's target is fulfilled and the code ends.

3. IMPLEMENTATION

A. Contribution to Dijkstra´s algorithm

The code for the Dijkstra´s algorithm that is used in this implementation is written in Python language and is executed separately from the main code written in C. It is done through transmitting the needed arguments from the C file to the Python script that the Dijkstra´s has to receive to calculate the shortest path. The over handed parameters are the topological map as well as the start and destination coordinates. As a return, the Python script will transmit the shortest path to the main C code. Because the Dijkstra´s algorithm needs a weighted graph to be able to calculate the shortest path, we have written an algorithm which generates a weighted graph tree out of the topological map. The algorithm sets every field of our topological map as nodes and every weight of the connections between two nodes to the same value. The algorithm is stored in the same Python script as the Dijkstra´s algorithm.

Algorithm 2 Generate weighted graph

```
1.  def buildGraph(arr):
2.  graph = {}
3.  max_row = len(arr)
4.  max_column = len(arr[0])
5.  for index_row, row in enumerate(arr):
6.  forindex_column, column in enumerate(row):
7.  cell_name = str(index_row) + str(index_column)
8.  cell_value = column
9.  sub_graph = get_weights(arr, index_row,
    index_column,max_row, max_column,  cell_value)
10. graph[cell_name] = sub_graph
11. return graph
12. def get_weights(arr, index_row, index_column,
    max_row, max_column,  cell_value):
13. weights = {}
14. behind_column_index = max(0, index_column - 1)
15. front_column_index = min(max_column - 1,
    index_column + 1)
16. above_row_index = max(0, index_row - 1)
17. below_row_index = min(max_row - 1, index_row + 1)
18. if(behind_column_index != index_column):
19. weights[str(index_row) + str(behind_column_index)]
    =NAND(cell_value,
    arr[index_row][behind_column_index])
20. if(front_column_index != index_column):
21. weights[str(index_row) + str(front_column_index)] =
    NAND(cell_value,
    arr[index_row][front_column_index])
22. if(above_row_index != index_row):
23. weights[str(above_row_index) + str(index_column)] =
    NAND(cell_value,
    arr[above_row_index][index_column])
24. if(below_row_index != index_row):
25. weights[str(below_row_index) + str(index_column)] =
    NAND(cell_value,
    arr[below_row_index][index_column])
26. return weights
27. def NAND(a, b):
28. if int(not (a and b)) == 1:
29.  return float("inf")
30. else:
31. return 1
```

The algorithm is executed calling the build Graph-function. It has to receive the topological map and will return the weighted graph.

B. Navigation algorithm

After receiving the calculated path the mobile robot should take, a navigation algorithm will calculate the needed driving directions in the right order to reach the mobile robots destination. For the algorithm to work it needs two information. The path displayed in a topological map and the simulated current position of the mobile robot.

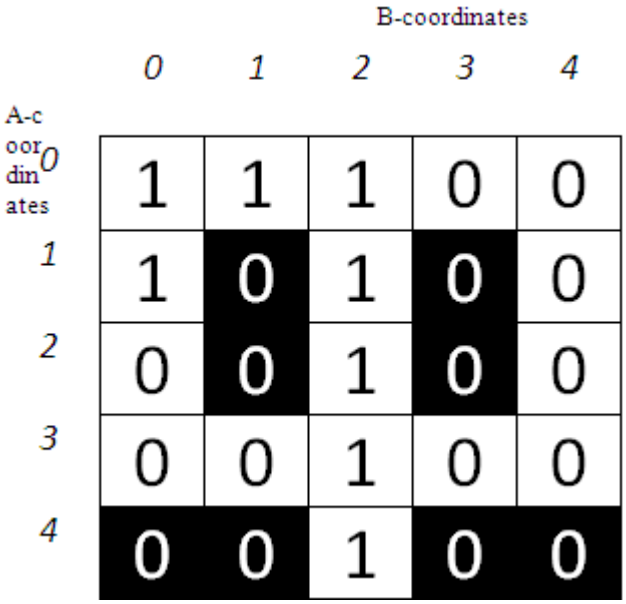

Figure-8. Path displayed in a topological map.

Figure-8 shows an example of how the path displayed in a topological map will look like. Every field of the map will be '0' at the beginning. After receiving the shortest path in coordinates from the Dijkstra's algorithm, every field with the coordinates contained in the shortest path will change to '1'. Like this, the path got visualized.

Algorithm 3.1 getDirection-function

```
1.  char getDirection(int path[5][5], char position[2]){
2.  path[position[0]-'0'][position[1]-'0'] = 0;
3.  if(position[0]!='0'){
4.  if(path[(position[0]-'0')-1][position[1]-'0']==1){
5.  position[0] = position[0]-1;
6.  return 'U';
7.  }
8.  }
9.  if(position[0]!='4'){
10. if(path[(position[0]-'0')+1][position[1]-'0']==1){
11. position[0] = position[0]+1;
12. return 'D';
13. }
14. }
15. if(position[1]!='0'){
16. if(path[position[0]-'0'][(position[1]-'0')-1]==1){
17. position[1] = position[1]-1;
18. return 'L';
19. }
20. }
21. if(position[1]!='4'){
22. if(path[position[0]-'0'][(position[1]-'0')+1]==1){
23.  position[1] = position[1]+1;
24. return 'R';
25. }
26. }
27. return 'F';
28. }
```

The next step is to call the get Direction-function. The function search for a current position the next adjacent '1' in the topological 2D-array map. Depending on where the adjacent '1' is, the information if it is up, down, left or right will be saved. For the information up a 'U' will be saved, for down a 'D' and so on. When there is no next adjacent '1' is left, there will be saved an 'F' for finished.

In the example shown in Figure-9, the following saved information will look like this: 'UUUULLDF'.

Algorithm 3.2 getSteering-function

```
1.  void getSteering(int steering[20], char
    pathDir[20]){
2.  int i = 0;
3.  steering[i] = 2;
4.  i++;
5.  while(pathDir[i] != 'F'){
6.  if(pathDir[i-1] == 'U'){
7.  if(pathDir[i] == 'U'){
8.  steering[i] = 2;
9.  }
10. else if(pathDir[i] == 'L'){
11.  steering[i] = 1;
12. }
13. else if(pathDir[i] == 'R'){
14. steering[i] = 3;
15. }
16. }
17. else if(pathDir[i-1] == 'D'){
18. if(pathDir[i] == 'D'){
19. steering[i] = 2;
20. }
21. else if(pathDir[i] == 'L'){
22. steering[i] = 3;
23. }
24. else if(pathDir[i] == 'R'){
25. steering[i] = 1;
26. }
27. }
28. else if(pathDir[i-1] == 'L'){
29. if(pathDir[i] == 'L'){
30. steering[i] = 2;
31. }
32. else if(pathDir[i] == 'U'){
33. steering[i] = 3;
34. }
35. else if(pathDir[i] == 'D'){
36. steering[i] = 1;
37. }
38. }
39. else if(pathDir[i-1] == 'R'){
40. if(pathDir[i] == 'R'){
41. steering[i] = 2;
42. }
43. else if(pathDir[i] == 'U'){
44. steering[i] = 1;
45. }
46. else if(pathDir[i] == 'D'){
47. steering[i] = 3;
48. }
49. }
50.  i++;
51. }
52. steering[i] = 4;
53. }
```

The last thing to do for the navigation after calculating the information about the direction is to provide an axis transformation to get the right steering value. For the used mobile robot, the value '1' means left steering, value '2' no steering, and value '3' means right steering. To calculate the right steering direction, the previous direction must be considered. The get Steering-function take care about this and saves the steering information. In the example shown in Figure-9, the following saved information will look like this: '22221214'. The value '4' is saved to indicate the destination.

C. Automatic driving algorithm

The implemented automatic driving algorithm will now lead the mobile robot to its destination using the calculated steering values in the right order as well as to the mobile robot attached IR-sensors.

Algorithm 4. Pseudo code for automatic driving

```
1.  while(autoDrivingMode){
2.  getNewSensorData();
3.  drive(foreward);
4.  steering(steeringValue[i]);
5.  depending on steeringValue:{
6.  display ("<" or ">" or "^");
7.  }
8.  if(leftSensor==triggered &&
    rightSensor==triggered){
9.  if(leftSensor's distance is shorter){
10. drive(backward);
11. steering(left);
12. delay(milliseconds);
13. drive(foreward);
14. steering(right);
15. delay(milliseconds);
16. }
17. else if(rightSensor's distance is shorter){
18. drive(backward);
19. steering(right);
20. delay(milliseconds);
21. drive(foreward);
22. steering(left);
23. delay(milliseconds);
24. }
25. }
26. else if(leftSensor==triggered){
27. drive(backward);
28. steering(left);
29. delay(milliseconds);
30. drive(foreward);
31. steering(right);
32. delay(milliseconds);
33. }
34. else if(rightSensor==triggered){
35. drive(backward);
36. steering(right);
37. delay(milliseconds);
38. drive(foreward);
39. steering(left);
40. delay(milliseconds);
41. }
42. else if(straightSensor==triggered){
43. drive(backward);
44. steering(straight);
45. delay(milliseconds);
46. drive(foreward);
47. steering(steeringValue[i]);
48. delay(milliseconds);
49. }
50.
51. if(loop did run a couple of times){
52. i++;  //increase index to get next steeringValue
53. }
54. if(steeringValue[i] == 4){
55. drive(stop);
56. autoDrivingMode = 0;
57. display('X');
58. }
59. }
```

At the beginning of the algorithm, the current steering value is used to steer the mobile robot in the right direction. In addition, the steering direction is displayed on a LED display attached to the mobile robot. Then, the IR sensors are going to be checked if they detect an obstacle or a wall. If both, the right and the left one are triggered; the algorithm will check which obstacle is nearer and react to the obstacle which is closer to the IR sensor. Otherwise the algorithm will just check the sensors each and react to them. If this loop did run a couple of times, the next steering value will be loaded. This continues the same way

until the last steering value is loaded. Because the last one is always '4', which would be an invalid steering value for the mobile robot, the automatic driving algorithm will stop and the LED display is going to display an 'X' to point out the end of the code. The couple of loop repeats before changing to the next steering value must be modified, depending on the mobile robot's speed, the surface, and the size of the fields of the map. This can be easily done by increasing or decreasing the repeat value of the loo preferring to line 51 of 'Algorithm 4'.

4. PERFORMANCE EVALUATION

For the performance evaluation, we tested the implemented Dijkstra's algorithm, as well as our implemented intelligent path planning algorithm. For that we provided three tests. The first two of them tested the reliability of our implemented Dijkstra's algorithm. The third one verified our intelligent path planning algorithm that is a combination of our topological map manipulation because of blocked road parts and the Dijkstra's algorithm.

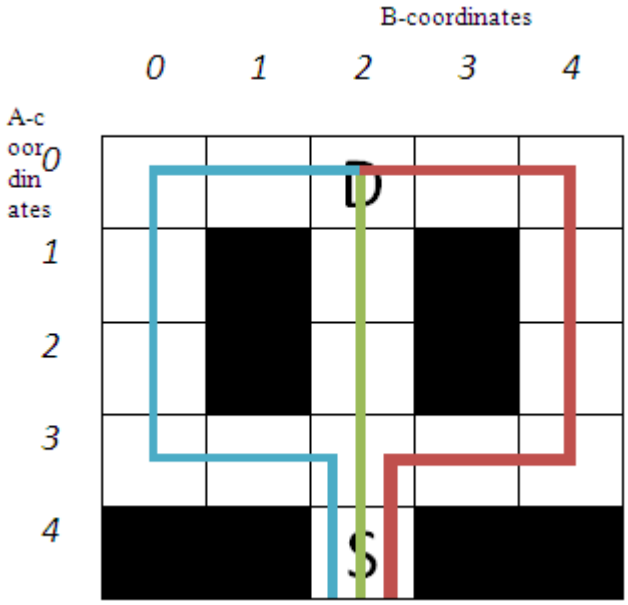

Figure-9. First route evaluation.

Figure-9 shows our first test. We set the mobile robot's destination at the coordinate '(0, 2)'. Then we measured first how much time the mobile robot took to drive all three possible routes each to the destination. After that we let the Dijkstra's algorithm calculate the shortest path, and observe which path it chooses.

Table-1. Time table for first route evaluation.

Route	Start-destination time	Dijkstra's choice
Blue	24.6 seconds	X
Green	4.0 seconds	✓
Red	25.5 seconds	X

As seen in Table-1, the time that the mobile robot took to reach its destination from the starting point using the three routes was measured. Also, the Dijkstra's chose the green route, since it was the shortest possible path among the three. The checkmark indicates the chosen path by the Dijkstra's algorithm and the X-mark indicates possible routes that are not chosen because of the longer time it takes to go to the destination resulting of a longer distance.

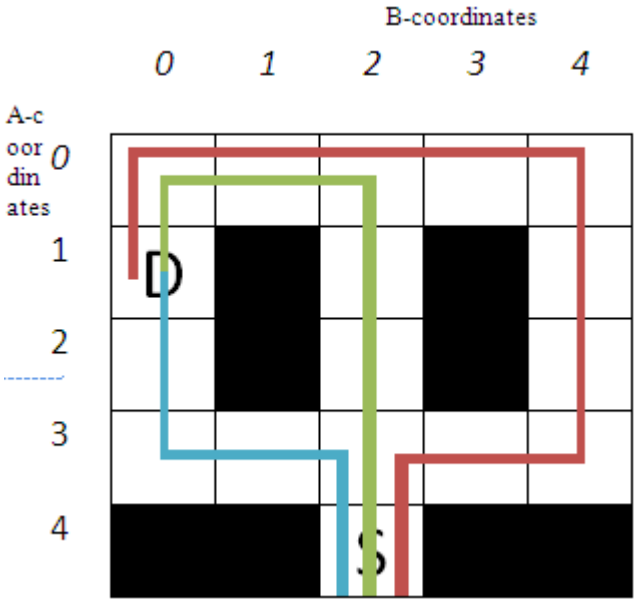

Figure-10. Second route evaluation.

Figure-10 shows our second test to validate the Dijkstra's algorithm. It works on the same principle as our first test as described for Figure-9. The destination was now set to the coordinate '(1,0)'.

Table-2. Time table for second route evaluation.

Route	Start-destination time	Dijkstra's choice
Blue	12.6 seconds	✓
Green	18.8 seconds	X
Red	34.1 seconds	X

Table-2 based on the same structure as shown in Table-1. It also shows the time that it takes for the mobile robot to drive from its start point to its destination using the three possible routes. In this test, the blue route was the fastest one. As seen in Table-2, it was also the Dijkstra's algorithm choice.

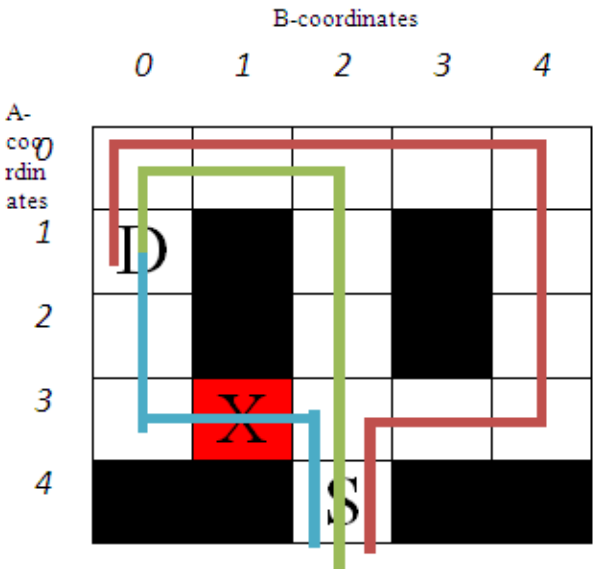

Figure-11. Blocked route evaluation.

The last performance evaluation was used to validate our implemented intelligent path planning algorithm. As seen in Figure-11, we set our destination as it was in our second route test, but we added a blocked route part at '(3, 1)'.

Table-3. Time table for blocked route evaluation.

Route	Start-destination time	Dijkstra's choice
Blue	12.6 seconds	X
Green	18.8 seconds	✓
Red	34.1 seconds	X

As seen in Table-3, even though the blue route would have been the fastest, the Dijkstra's choice was the green route. That's because our intelligent path planning algorithm did successfully its job, and marked the coordinate '(3, 1)' as blocked, so that the Dijkstra's calculation would not consider any paths using this field.

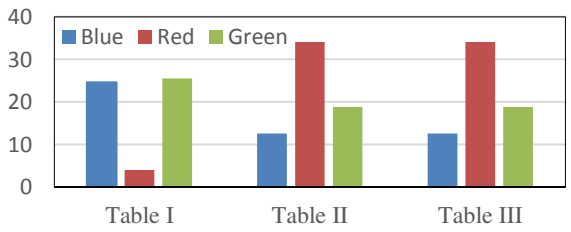

Figure-12. Visualization of results.

Figure-12 visualizes a summary of the time results of the three different evaluations displayed on the ordinate. The different colored bars represent the three different routes taken per evaluation (refer to Figures 9, 10, and 11 and their Tables 1, 2, and 3).

5. CONCLUSIONS

An intelligent path planning that uses a shortest path algorithm specifically the Dijkstra´s algorithm is proposed and with the use of wireless ad hoc network it is able to provide a wireless communication to improve this proposed idea. First, sited sources that covers the topic and systems that are going to be implement in this project are shown in order to know if it is possible to be done and what are the limitations of this proposed idea. Second, an algorithm like Dijkstra´s was chosen to make the study less complicated but effective. Also, wireless ad hoc network provide simple concept of a wireless communication that makes this idea possible. It is presented throughout the paper regarding the approach on how this intelligent path planning are done while anchoring it in all our references. Images, table, and graphs are also presented in the paper to provide clear understanding about the proposed idea. Lastly, the obtained results show that our intelligent path planning algorithm works as expected in three different approach and it is successfully implemented the idea of this algorithm, which is able to avoid blocked parts on a route, and find an alternative path, which is the shortest one while avoiding the blocked areas.

6. RECOMMENDATIONS

A. Server-side

As mentioned in the introduction, the implementation of the server is not provided in this paper but it's recommended to extend this proposed idea with a server side or with other mobile robots sharing communication to others like a car-to-car communication.

B. Improvement of Dijkstra's implementation

For further studies about this presented idea we recommend writing the Dijkstra´s algorithm as well as the algorithm for generating the weighted graph in the same language as the main code so that it can be executed without transmitting data from one language to another.

C. Improvement of automated driving algorithm

For an even better automatic driving algorithm we recommend implementing more than three IR sensors. In the presented automatic driving algorithm only the three sensors attached to the front bumper of the mobile robot have been used.

ACKNOWLEDGEMENTS

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP; Ministry of Science, ICT & Future Planning) (No. 2017R1C1B5016837).

REFERENCES

[1] Y. Hada. 2004. Delivery service robot using distributed acquisition, actuators and intelligence. In International Conference on Intelligent Robots and Systems, Sendai, Japan.

[2] T. Terzimehic. 2011. Path Finding Simulator for Mobile Robot Navigation.

[3] L. Guo. 2012. Intelligent Path Planning for Automated Guided vehicles System based on Topological map.

[4] D. Esparza. 2013. Topological Mobile Robot Navigation Using Artificial Landmarks. Latin American Robotics Symposium, 2013.

[5] S. Sakib. 2014. Maze solving algorithm for line following robot and derivation of linear path distance from nonlinear path. In Int'l Conf. Computer and Information Technology, Bangladesh.

[6] E. Ogawa. 2017. A Trustworthiness-Based Ad-Hoc Routing Protocol in Wireless Networks. in Advanced Information Networking and Applications (AINA), 2017 IEEE 31st International Conference, Taipei, Taiwan, Taiwan.

[7] W. Chen. 2016. History-based multi-node collaborative localization in mobile wireless ad hoc networks. In Communications (ICC), 2016 IEEE International Conference, Kuala Lumpur, Malaysia.

[8] X. Li. 2016. Performance modeling and analysis of distributed multi-hop wireless ad hoc networks. in Communications (ICC), 2016 IEEE International Conference, Kuala Lumpur, Malaysia.

[9] V. A. Shim. 2014. Direction-driven navigation using cognitive map for mobile robots. in International Conference on Intelligent Robots and Systems, Chicago, USA.

[10] IEEE. 2015. IEEE Standard for Robot Map Data Representation for Navigation.

[11] S. Lee. 2006. Rotating IR Sensor System for 2.5D Sensing. In International Conference on Intelligent Robots and Systems, Beijing, China.

[12] A. Lourenco. 2016. On the design of the robo-partner Intra-factory logistics autonomous robot. In International Conference on Systems, Man, and Cybernetics, Budapest, Hungary.