Project Proposal
Pathify: Real-Time Path Planning and Optimization

Jong Seo Yoon, Timi Dayo-Kayode, Michael Edegware

Abstract

There are many algorithms which tackles the problem of a mobile robot's path planning within a map. A mobile robot can navigate through the map using computer vision to distinguish a wall and a floor, unique points, and alternate paths it may find along the way, and shortest path algorithms can be used to solve the problem of finding a path between two vertices in a graph such that the sum of the weights of its constituent edges are minimized. In this project, we would like to devise a method to combine computer vision approach to mobile robot's navigation along with shortest path algorithm, such as A search algorithm or Dijkstra's algorithm, to simulate a mobile robot reaching different parts of the map and optimizing its path along the way. As a result, we expect to show how a mobile robot in the real world can adapt and find the shortest path to the destination despite its lack of knowledge of a complete map of the area.*

Introduction

Our project proposes a revised shortest path problem by determining the shortest path from the initial position to the destination using the topological map and the weighted graph generated while the mobile robot navigates the map without any information. A unique point may be of any color or marked using symbols on the wall, and each color or symbol will represent a unique place on the map. Our revised shortest path problem is divided into two parts; shortest path algorithm to find the shortest path from the starting point to the destination and computer vision approach to navigate the unknown map. Shortest path algorithm, such as A* search algorithm or Dijkstra's algorithm, will be used to find the shortest path after the mobile robot has found the destination, and computer vision will be used to differentiate between walls and paths, and also identify unique points while the mobile robot is in motion.

In our simulator, we assume that the mobile robot will move at a fixed maximum and turning speed. However, to simulate a mobile robot in the real world, an acceleration rate will be used. This assumption creates a possibility that given all possible paths known to the mobile robot to the target destination, the shortest path by distance may not be the fastest path to get to the destination. For instance, we may encounter a situation where the path to the unique point is measured faster in time than the shortest path on the map due to amount of turns the mobile robot has to perform along the path. Moreover, walls and floors for the path will have a fixed color which will be different from any other unique color and symbols which represent different destinations, a simplification necessary for the mobile robot to easily detect the difference.

Because the mobile robot builds the knowledge about the map as it navigates throughout the map, any paths in which the mobile robot has not taken is not considered during the shortest path calculation phase. Thus, the shortest path calculated by the A* search algorithm or Dijkstra's algorithm may not be the actual shortest path if all paths on the map were to be considered. Using this approach however, we expect the mobile robot to reduce the time it takes to get to the destination as different paths are explored while visiting other destinations on the map.

Problem Formulation and Technical Approach

The basic problem formulation begins by creating a map in which the mobile robot will be able to explore. The input will be a unique location the mobile robot will be asked to find, and if the robot doesn't know about the unique location, it will randomly navigate around the map until it finds the unique location and save it in the memory. Once saved, the robot will return home and the shortest path algorithm will be run using the weighted graph to determine the shortest path to the unique location it found, along with every other previous unique location saved in its memory. We want to calculate the shortest path to all unique locations known by the mobile robot because every time the robot navigates around the map, the weighted graph may improve as it explores other parts of the map, and it may find shorter path to unique locations it visited in the past.

Figure 1 shows a sample map in which we will run our simulation in. We define a map as a maze without an end goal for an exit. Instead, the map will contain a starting point for the mobile robot, or home, and unique points will be randomly placed around the map representing meaningful places such as hospital, police station, or even other people's houses. Every unique point on the map will have different symbols or colors from the wall. The map will allow intersections to imitate the city streets, but no more than four paths will be allowed on the intersection to reduce complexity. In order for the mobile robot to determine if it has reached the destination, we first plan on using predetermined point on every unique location and have the mobile robot navigate to the predetermined point in the case where the mobile robot does not have the unique location in its memory.

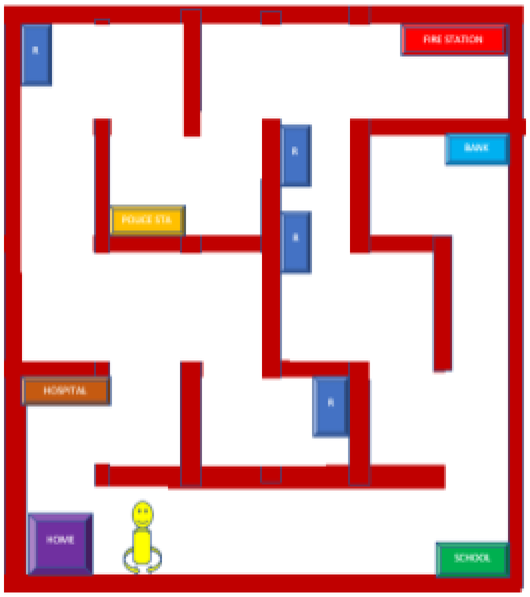


Fig 1. Sample map

Because the mobile robot's knowledge of the map gets updated constantly as it navigates around the map, the shortest path problem must be adaptive to change based on the current scope of knowledge of the map. To tackle the shortest path problem, we need to develop a way for the mobile robot to create nodes and its distance from the previous node as it navigates around the map. As the topological map gets created, the mobile robot will create a node based on whether it needs to turn, or at every interval in time. For instance, the robot will create a node of the graph once it reaches an edge of the map, and it will also create a node after a finite amount of time to tackle the mobile robot going in a straight line. At every node, it will calculate the time it took to get to the node on the map from the previous node and set it as the vertex between two nodes. If two nodes are close to each other, meaning the time it took for the mobile robot to reach from the first node to the second node is less than a certain time interval, then two nodes will be combined into the second node and recalculate the vertices by adding the first node vertices to the second node. Figure 2 depicts how it is done: When the robot is told to find a certain unique point, the robot starts moving in a random direction, in this case from A to B. It will create a node whenever the robot detects more than one path, calculate the time it took to get to the node it created, and move on. Once the mobile robot reaches the final destination, it will go back to the home location, and calculate the shortest path using either of the shortest path algorithm, which in this case will be A, B, C, D, E, H, I. Although we clearly see from the figure that this is not the shortest path, this is the only shortest path the mobile robot currently knows. However, as the mobile robot reaches different locations on the map, it will keep updating the weighted graph in which it has discovered, thus shortest path will be found once the robot discovers every path on the map. Also, notice on figure 2 that two weights between node E and H are different. If the two weights are different, we take the lower value as the weight between the two nodes for calculation.

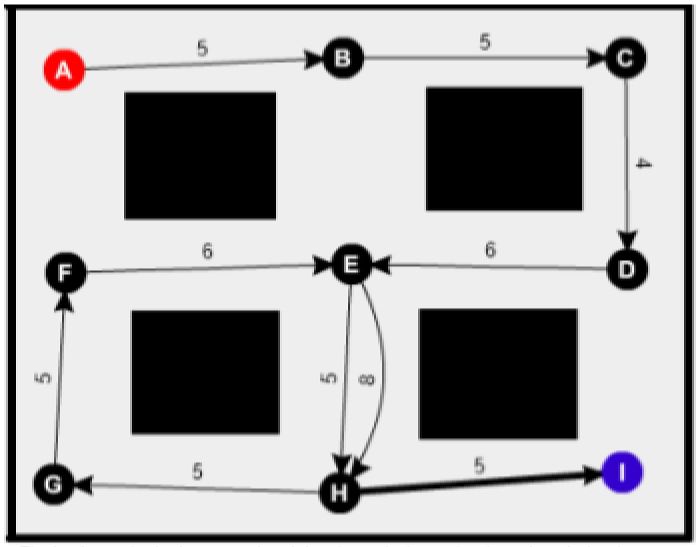
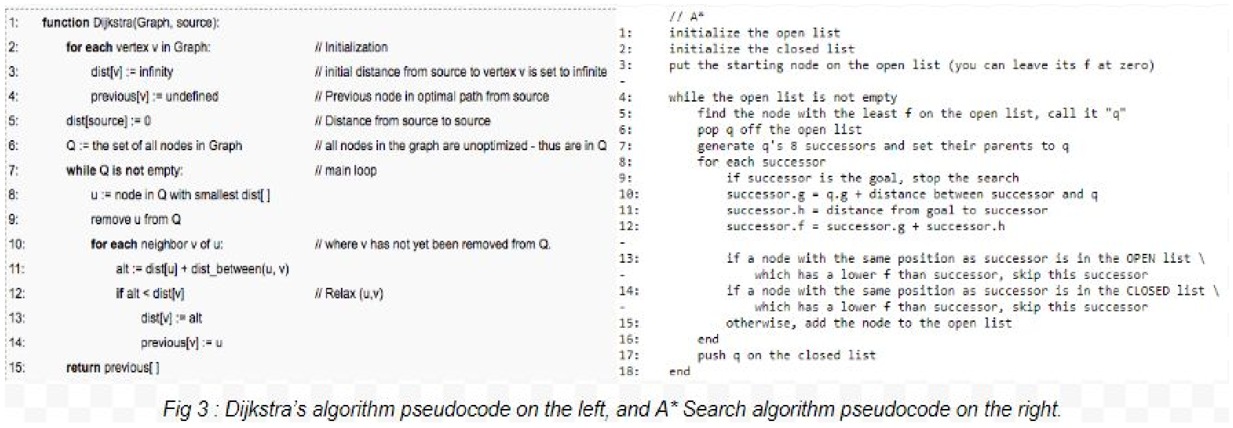


Fig 2 : A topological map and weighted graph drawn by the robot during its navigation. Numbers on edges represent time it took to get from one point to the next point. Red color A is the starting point, and the blue color I is the destination point. In this sample, the robot moved in the following order: A, B, C, D, E, H, G, F, E, H, I.

Figure 3 below is the pseudocode for Dijkstra’s algorithm and A* search algorithm. Both algorithms are widely known as the best shortest path searching algorithm in weighted graphs. Dijkstra’s algorithm was developed by computer scientist Edsger W. Dijkstra. Dijkstra’s algorithm works by recursively finding the shortest path to the destination node in a weighted graph and create a shortest path tree. Dijkstra’s algorithm works by marking all nodes unvisited, and iterate to calculate distance between every unvisited node from the current node. On the other hand, A* search algorithm was first developed by the researchers who worked on “Shakey the Robot” at Artificial Intelligence Center of Stanford Research Institute. A* search algorithm is also a widely used shortest path algorithm for graph traversal due to its performance and accuracy. It solves the shortest path problem by searching among all possible paths from the initial node to the destination(goal) node in a weighted graph, and pick the one that incurs the smallest cost.



```
function Dijkstra(Graph, source):
    for each vertex v in Graph:                       // Initialization
        dist[v] := infinity                           // initial distance from source to vertex v is set to infinite
        previous[v] := undefined                      // Previous node in optimal path from source
    dist[source] := 0                                 // Distance from source to source
    Q := the set of all nodes in Graph                // all nodes in the graph are unoptimized - thus are in Q
    while Q is not empty:                             // main loop
        u := node in Q with smallest dist[ ]
        remove u from Q
        for each neighbor v of u:                     // where v has not yet been removed from Q.
            alt := dist[u] + dist_between(u, v)
            if alt < dist[v]                          // Relax (u,v)
                dist[v] := alt
                previous[v] := u
    return previous[ ]
```

```
// A*
initialize the open list
initialize the closed list
put the starting node on the open list (you can leave its f at zero)

while the open list is not empty
    find the node with the least f on the open list, call it "q"
    pop q off the open list
    generate q's 8 successors and set their parents to q
    for each successor
        if successor is the goal, stop the search
        successor.g = q.g + distance between successor and q
        successor.h = distance from goal to successor
        successor.f = successor.g + successor.h

        if a node with the same position as successor is in the OPEN list \
            which has a lower f than successor, skip this successor
        if a node with the same position as successor is in the CLOSED list \
            which has a lower f than successor, skip this successor
        otherwise, add the node to the open list
    end
    push q on the closed list
end
```

Fig 3 : Dijkstra's algorithm pseudocode on the left, and A Search algorithm pseudocode on the right.*

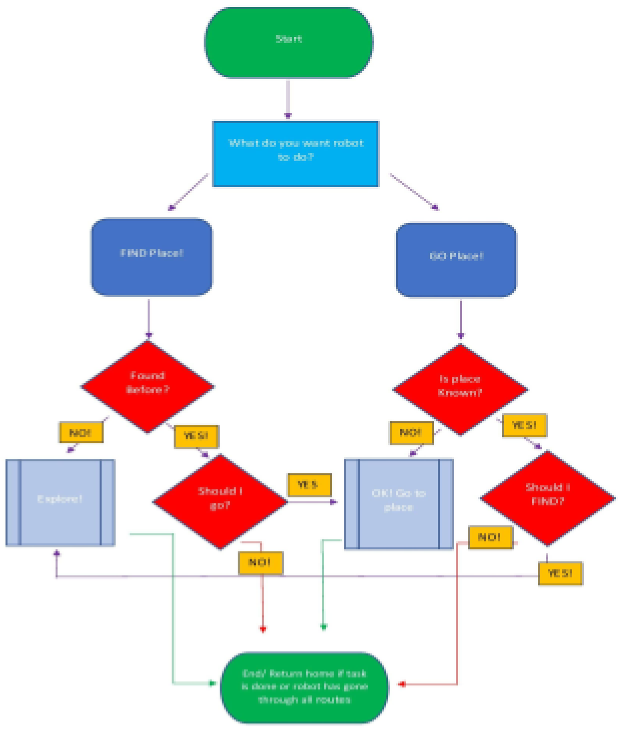


Fig 4 : Mobile robot's workflow diagram.

The flowchart in Fig 4 shows the summary of the technical functionality of the robot. The robot is given two kinds of instruction: FIND AND GO. In FIND mode, the mobile robot goes around the city to learn and map locations. In the GO mode, the robot goes to a location it has found before, following the shortest distance calculated from its map and the current location using the maze algorithm. If the robot is asked to find the place, it first searches its memory and see if it location has already been explored. If yes, it asks whether to switch to GO mode. If no, it will start exploring. On the other hand, if asked to GO, it first see if it had found the place before and proceeds to the place. If not, then it asks whether to switch to FIND mode. In either mode, the robot returns home once it has reached the location.

Expected Results / Experimental Validation

The criteria for success is for the mobile robot to be able to navigate around the map, and create nodes along the way in order to find the shortest path. As the robot discovers paths on the map, we expect the time it takes for the mobile robot to get to the destination will decrease as the robot discovers more paths around the map. Our first goal will be to simulate the mobile robot's navigation in Robot Operating System(ROS), and then try it on the real mobile robot if time allows.

To validate our experiment, we are interested in two results. We are interested in whether the robot will be able to navigate through the map and detect unique locations, and the performance of the shortest path algorithms which will be dependent on the weighted graph the mobile robot creates during the navigating phase. We are interested in the shape of the weighted graph the robot will generate as it moves around the map, because the value of edges and the shape of the graph will determine the mobile robot's performance. We will also run both Dijkstra's algorithm and A* search algorithm to see which algorithm performs better. To measure performance between the two algorithms, we will measure the time it took to figure out the shortest path, and if the shortest path found by both algorithms differs, then we will measure the time of the mobile robot to get to the destination using shortest path calculated from both algorithms. For the experiment with the real mobile robot, we plan to use the part of the floor with couple of rooms, where unique objects will be placed in each room. We will then run the same computer vision and shortest path algorithms to determine if the robot can find the object, calculate the shortest path, and reduce the time it takes to travel to the same object on the next run.

Timeline / Schedule

Task	Estimated Duration	Estimated End Date	Assigned to
Learn to use ROS	Fulltime	Fulltime	All
Coding shortest path algorithms	2 weeks	11/16/2017	Timi Dayo-Kayode
Create few 3D maps on ROS	2 weeks	11/16/2017	Michael Edegware
Create mobile robot on ROS and integrate computer vision for navigation	2 weeks	11/16/2017	Jong Seo Yoon
Weighted graph generation from the mobile robot	2 weeks	11/25/2017	Timi Dayo-Kayode and Jong Seo Yoon
Test on the ROS simulator and tweak, revise and fix bugs as needed	3 weeks	12/03/2017	All
Voice command through voice recognition	1 week	12/03/2017	Michael Edegware
Test on the real mobile robot	End of semester	End of semester	All

References

1. Hwang, Kao-Shing, Chih-Wen Li, and Wei-Cheng Jiang. "Adaptive exploration strategies for reinforcement learning." In *System Science and Engineering (ICSSE), 2017 International Conference on*, pp. 16-19. IEEE, 2017.
2. Mirowski, Piotr, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andy Ballard, Andrea Banino, Misha Denil et al. "Learning to navigate in complex environments." *arXiv preprint arXiv:1611.03673* (2016).
3. Smart, Paul R., and Katia Sycara. "Place Recognition and Topological Map Learning in a Virtual Cognitive Robot." In *Proceedings on the International Conference on Artificial Intelligence (ICAI)*, p. 3. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2015.
4. Meikle, Stuart, and Rob Yates. "Computer vision algorithms for autonomous mobile robot map building and path planning." In *System Sciences, 1998., Proceedings of the Thirty-First Hawaii International Conference on*, vol. 3, pp. 292-301. IEEE, 1998.
5. Zhu, Weiyu, and Stephen Levinson. "Vision-based reinforcement learning for robot navigation." In *Neural Networks, 2001. Proceedings. IJCNN'01. International Joint Conference on*, vol. 2, pp. 1025-1030. IEEE, 2001.
6. Se, Stephen, David Lowe, and Jim Little. "Vision-based mobile robot localization and mapping using scale-invariant features." In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, vol. 2, pp. 2051-2058. IEEE, 2001.
7. Jan, Gene Eu, Ki Yin Chang, and Ian Parberry. "Optimal path planning for mobile robot navigation." *IEEE/ASME transactions on mechatronics* 13, no. 4 (2008): 451-460.
8. Center, Artificial Intellgence. "SHAKEY THE ROBOT." (1984).