Journal of Marine Science and Technology

Volume 25 | Issue 1 Article 4

EXPLORATION METHOD IMPROVEMENTS OF AUTONOMOUS ROBOT FOR A 2-D ENVIRONMENT NAVIGATION

Nien-Yu Chen
Institute of Mechatronic Engineering, National Taipei University of Technology, Taipei City, Taiwan, R.O.C.,
optimal_uu@hotmail.com

Jinsiang Shaw
Institute of Mechatronic Engineering, National Taipei University of Technology, Taipei City, Taiwan, R.O.C.

Hsien-I Lin
Institute of Automation Technology, National Taipei University of Technology, Taipei City, Taiwan, R.O.C.

EXPLORATION METHOD IMPROVEMENTS OF AUTONOMOUS ROBOT FOR A 2-D ENVIRONMENT NAVIGATION

Acknowledgements

This work was supported by Minister of Science and Technology, Taiwan, under grant number NSC 102-2221-E-027-039.

Journal of Marine Science and Technology, Vol. 25, No. 1, pp. 34-42 (2017)
DOI: 10.6119/JMST-016-0719-1

EXPLORATION METHOD IMPROVEMENTS OF AUTONOMOUS ROBOT FOR A 2-D ENVIRONMENT NAVIGATION

Nien-Yu Chen[1], Jinsiang Shaw[1], and Hsien-I Lin[2]

Key words: autonomous robot, iterative closest point, path planning, robot navigation.

ABSTRACT

This paper presents a method for integrating different algorithms for building an autonomous robot. The developed robot has the ability to construct the 2-D map of an unknown environment, localize itself in the map, explore undiscovered area, and path finding. In order to efficiently explore a map, we propose an exploration method. Additionally, improvements of conventional A* algorithm are proposed. Experimental results show that the developed robot is capable of navigating an unknown indoor environment with random obstacles.

I. INTRODUCTION

Recently autonomous robot systems are gaining much attention in our daily life. Cleaning and surveillance robots are revealed to the public. For example, iRobot roomba robotic vacuum cleaner is a great success of the development of autonomous robot systems (Forlizzi and DiSalvo, 2006). DARPA grand challenge-autonomous vehicles (Chen et al., 2004) and Google driverless car project (Levinson et al., 2011) showed that autonomous robot cars drove in cities. The U.S. states of California and Nevada permitted the operation of autonomous cars in 2012.

For autonomous robots, the ability to perceive environments is vital. The difficulties in acquiring this ability include map building, localization, path finding, and map exploration algorithms. The first two are usually solved by SLAM methods (Smith and Cheeseman, 1986a; Smith and Cheeseman, 1986b; Thompson et al., 2011; Tong et al., 2013). There are four common algorithms for SLAM: EKF, SEIF, FastSLAM, and GraphSLAM.

Paper submitted 10/03/14; revised 02/15/16; accepted 07/19/16. Author for correspondence: Nien-Yu Chen (e-mail: optimal_uu@hotmail.com).
[1] Institute of Mechatronic Engineering, National Taipei University of Technology, Taipei City, Taiwan, R.O.C.
[2] Institute of Automation Technology, National Taipei University of Technology, Taipei City, Taiwan, R.O.C.

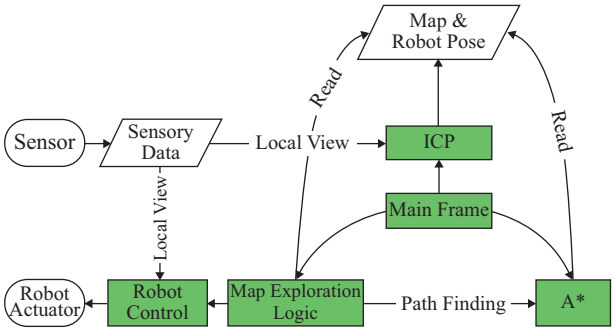


Fig. 1. System architecture.

Here we use another much simpler approach-Iterative Closest Point (ICP) (Besl and Mckay, 1992) for building map and localization. Together with efficient path finding and map exploration algorithms to be discussed, an autonomous robot capable of navigating an unknown indoor environment with random obstacles can be developed.

Block diagram of the developed robot system is illustrated in Fig. 1. There are three major algorithms in this system-ICP, A* and map exploration logic. A* is for path finding, and map exploration logic is for robot navigation (Kurabayashi et al., 1996; Koenig et al., 2001; Fang et al., 2005). In this paper, each of these three algorithms will be addressed in detail in the following sections. Finally, experiments will be carried out to show effectiveness of the algorithms for a 2-D environment navigation.

II. LOCALIZATION AND MAPPING

1. Data Representation

The first step of building an autonomous robot is to decide the representation method of its map data (Elfes, 1989; Kuipers and Byun, 1991; Bandera et al., 2001; Guivant et al., 2002; Schroter et al., 2004). In this paper, we use occupancy grid map. It presents maps in grids. Each grid keeps the probability of being occupied by obstacles, as illustrated in Fig. 2. All grids initialize its probability to be 0.5, and

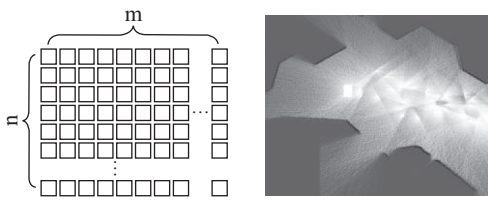


Fig. 2. A 2-D occupancy grid map.

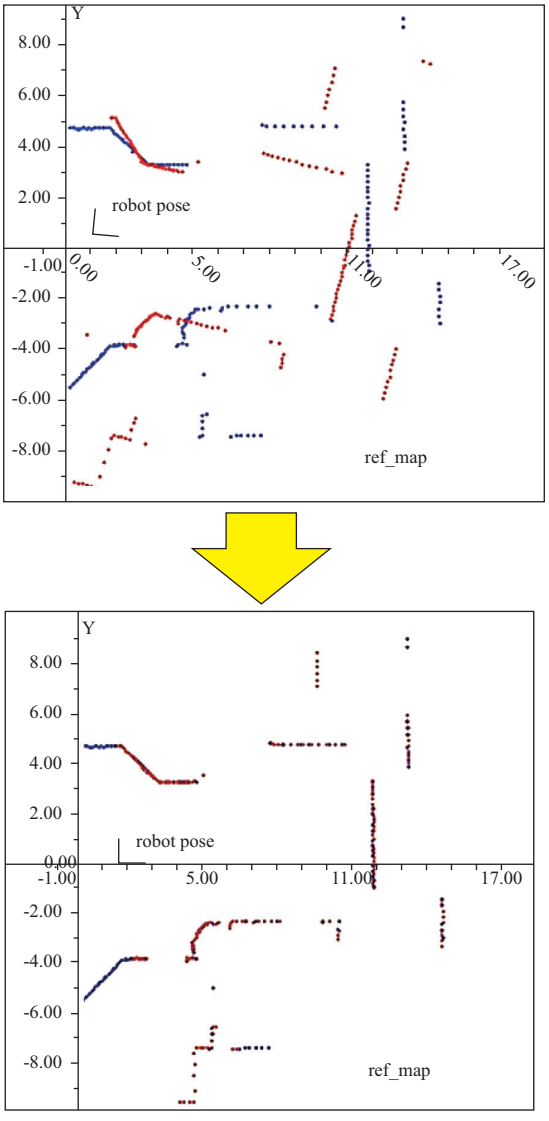


Fig. 3. Rigid alignment.

- If probability < LOW: occupied (black area in Fig. 2)
- If probability > HIGH: empty (white area)
- If LOW ≦ probability ≦ HIGH: uncertain cell (gray)

where LOW and HIGH ∈ (0, 1) are prescribed thresholds.

2. ICP

Once we know how to represent a map. The next step is to construct our environment map from sensory data. This constructing problem can be considered as a rigid alignment (point set registration) problem. For example, suppose we have two sets of sensory data P and Q. The data in P and Q are partially the same, which means they have some identical points, as illustrated in Fig. 3. We can match these two data sets as long as we can find the rotation and translation between P and Q.

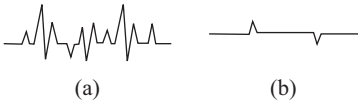


Fig. 4. Meshes with features.

ICP takes six stages to find the rotation and translation:

- Down sampling
- Matching
- Weighting
- Rejection
- Building error function
- Minimizing the error function

1) Down Sampling

When Besl proposed the very first ICP, he used all available points in both meshes. However, the performance hits its limit when the number of points grows. So, another two common sampling strategies were proposed:

- Random sampling (Masuda et al., 1996)
- Uniform subsampling (Turk and Levoy, 1994)

Radom sampling reduces the number of point simply by random selection. And Uniform subsampling selects points uniformly across the buckets. Although these two strategies reduce the number of points, they don't consider the feature of meshes. For example, Fig. 4(b) has fewer features than in Fig. 4(a), random and uniform sampling methods might fail in Fig. 4(b) by missing features. And this is why normal-space sampling (Rusinkiewicz, 2001) and intensity sampling (Weik, 1997) were proposed.

The idea of normal-space sampling is choosing points such that the distribution of normals among selected points is as large as possible. Intensity sampling, on the other hand, has the selection of points with high intensity gradient. Both of these strategies consider the features of meshes.

No matter what strategy you use, you can choose to exert the sampling on only one mesh, or on both meshes. Sampling on both meshes might help a little on the performance, but not much (Rusinkiewicz, 2001).

2) Matching

The second stage of ICP is to find the corresponding points between meshes. In this stage, The original ICP (Besl, 1992) simply uses the closest point as the correspondence. For instance, Fig. 5(a) presents two meshes, and Fig. 5(b) shows the result of finding closest points.

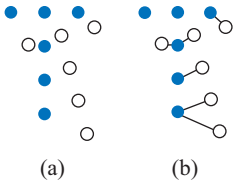


Fig. 5. Finding closest point.

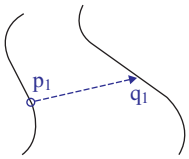


Fig. 6. Normal Shooting.

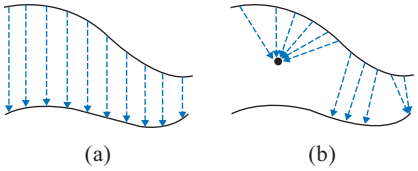


Fig. 7. Reverse calibration.

There are two issues in this closest point finding strategy. First, the closest point is not always the real correspondence. You can tell this from Fig. 5(b). Second, this strategy is very time-consuming. The cost is O (NM), where N and M represents the number of points in both meshes. Therefore, we usually use k-d tree to accelerate the process. K-d tree separates mesh spaces into several individual dimensions. In this way, we only find corresponding point in the same dimension, which increases the performance to O (logN).

There are some other matching strategies:

- Normal shooting (Chen, 1991)
- Reverse calibration (Blais, 1995)

Normal shooting finds the intersection of the source point's normal with the destination surface. For example, in the scene of Fig. 6, p_1 finds its correspondence, q_1, along its normal.

Reverse calibration, Fig. 7(a), on the other hand, projects the source point onto the destination mesh. This method has a remarkable performance, in constant time, because it uses projection to find correspondences. Another advantage is this method doesn't easily be affected by noise. For instance, Fig. 7(b) presents a scene with noise in the middle. The original closest point method will generate a lot of incorrect correspondences because the noise has a shorter distance to the source points.

Any of the above strategies can also use restrictions in addition, such as:

- Angle between normals (Pulli, 1999)

For example, Fig. 8(a) is a case that uses ordinary closest point to find correspondences. Fig. 8(b), also uses ordinary closest point strategy, but it only matches correspondence when the source and the destination have normals within 20 degrees.

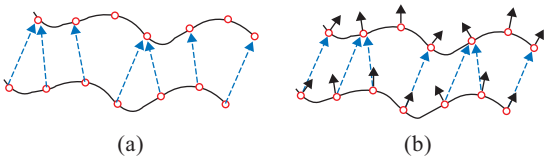


Fig. 8. Matching with restriction of angle between normals.

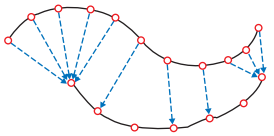


Fig. 9. Rejection of boundaries.

3) Weighting

After matching, we might also assign different weights to corresponding point pairs. There are some available weighting strategies:

- Constant weight
- Weighting based on distance
- Weighting based on normals

The second weighting strategy uses the following formula to calculate the weights:

$$Weight = \frac{Dist_{\max} - Dist(p,q)}{Dist_{\max}}$$

And the third strategy uses:

$$Weight = n_1 \bullet n_2$$

where n_1 and n_2 represent the normals of the source point and destination point.

The idea of these two strategies is: The greater distance (or difference between normals), the less accuracy. However, weighting doesn't affect the speed of convergence that much. The choice of a weighting strategy should be based on the accuracy.

4) Rejection

The purpose of this stage is to eliminate the impact of noise and incorrect corresponding pairs. An easy way to do this is to reject pairs that have distance more than a given threshold. Or, you can reject the worst n% of pairs based on the distance (Pulli, 1999). However, this stage doesn't improve the speed of convergence either. The affection of this stage is majorly on the accuracy.

An interesting strategy of this stage is to reject boundaries on meshes (Turk, 1994). Since boundaries usually cause lots of incorrect pairs, as in Fig. 9, this strategy is usually recommended.

5) Building and Minimizing Error Function

The classic ICP has the following error function:

$$E = \sum_{i=1}^{N} | Q_i - (RP_i + T) |^2 \tag{1}$$

where R denotes the rotation matrix, and T denotes the translation matrix.

Since Eq. (1) has two unknowns (R and T, it is impossible to solve an equation with two unknowns). So we have to rewrite Eq. (1) by the following steps:

First, find the centers of P and Q:

$$\begin{cases} p = \dfrac{1}{N}\sum_{i=1}^{N} P_i \\ q = \dfrac{1}{N}\sum_{i=1}^{N} Q_i \end{cases} \tag{2}$$

and then shift P and Q to the origin of coordinates:

$$\begin{cases} p_i' = P_i - p \\ q_i' = Q_i - q \end{cases} \tag{3}$$

after shifting P and Q to the origin, we can rewrite Eq. (1):

$$E = \sum_{i=1}^{N} | q_i' - Rp_i' |^2 \tag{4}$$

Compare Eq. (1) to Eq. (4), we can see the new Eq. (4) has only one (rotation) unknown in it, which is solvable:

$$\begin{aligned} E &= \sum_{i=1}^{N} (q_i' - Rp_i')^t (q_i' - Rp_i') \\ &= \sum_{i=1}^{N} (q_i'^t q_i' + p_i'^t R^t R p_i' - q'^t_i R p_i' - q'^t_i R^t p_i'^t) \\ &= \sum_{i=1}^{N} (q_i'^t q_i' + p_i'^t p_i' - 2 q_i'^t R p_i') \end{aligned} \tag{5}$$

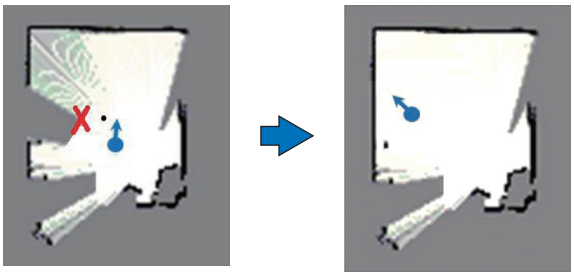
To minimize this error function (5), we only have to maximize the later part ($q_i'^t R p'$) after minus sign.

In order to maximize $q_i'^t R p'$ in Eq. (5), let

$$Trace(RH) = \sum_{i=1}^{N} q_i'^t R p_i' \tag{6}$$

where

$$H = \sum_{i=1}^{N} p_i' q_i'^t \tag{7}$$



(a) (b)

Fig. 10. Situation 1.

In this way, we switch the original minimizing problem to another question: "How to maximize Trace (RH)?"

Thus, let us find R so that RH is symmetric positive define. Then we know for sure that Trace (RH) is maximal. If $H = U\Sigma V^t$ is the SVD (Singular Value Decomposition), we define

$$R = VU^t . \tag{8}$$

Now let us check RH:

$$RH = (UV^t)(U\Sigma V^t) = V\Sigma V^t , \tag{9}$$

which is a symmetric matrix and its eigenvalues are positive, meaning that RH is symmetric positive define.

Consequently, we can find the translation matrix T in Eq. (1), because:

$$T = q - Rp \tag{10}$$

After matching P and Q, we have a basic map on our own. Moreover, by calculating the rotation and translation, we get the position and orientation of the robot too, which means we solved the mapping and localization problems simultaneously.

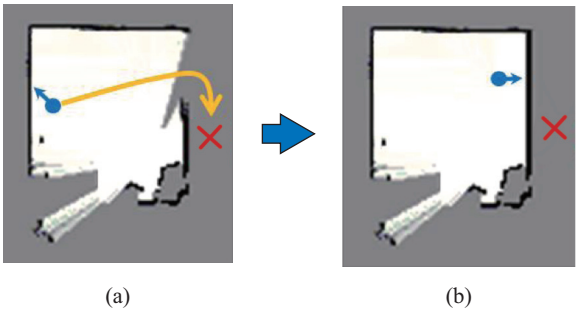
III. MAP EXPLORATION

Using ICP gives the robot the ability to draw a map and localize itself. But the robot still doesn't know how to navigate through the environment yet. This is why we need a map exploration logic.

To explore a map, we first assume the robot is located in a 2-D occupancy grid map. Each grid in the map has three possible statues:

(1) Occupied by obstacle. (Black area in Fig. 10)
(2) Not occupied (Empty). (White area in Fig. 10)
(3) Uncertain. (Gray area in Fig. 10)

Of course, all the grid cells in the map are initialized to be "Uncertain", and the robot has at least one sensor (such as camera or infrared, Fig. 15) to "see".



(a) (b)

Fig. 11. Situation 2.

Fig. 12. Situation 3.

Now, to explore this map, we constantly make the robot search for the nearest "Uncertain" cell. If an uncertain cell is found, we use a path-finding algorithm to find a valid path, and then make the robot move along the path.

As long as we repeatedly do the "search-path finding-move" action, the robot will eventually explore the whole environment. However, there are three possibilities for finding a path:

1. Path found → Robot move → Move successfully.
2. Path found → Robot move → Robot can't move (obstacle found)
3. No available path.

1. Situation 1

Suppose the robot is in the middle of a map, illustrated as Fig. 10(a). The nearest cell (denoted by a red cross) is on the upper-left of the robot, and there is a path to this target cell. Moving the robot will be able to update the map on the upper-left corner, illustrated as Fig. 10(b).

2. Situation 2

Illustrated as Fig. 11. When robot is moving, there is a chance we might find a new obstacle on the way. If so, we have to stop the robot and rearrange a new path for the robot.

3. Situation 3

In this situation, the target cell is unreachable to the robot. As illustrated in Fig. 12, the hatched area is isolated to the robot, there is no path for the robot to move.

As we can see, this exploration method is simple. As long as we handle these three situations carefully, the robot will be able to explore the whole environment eventually.

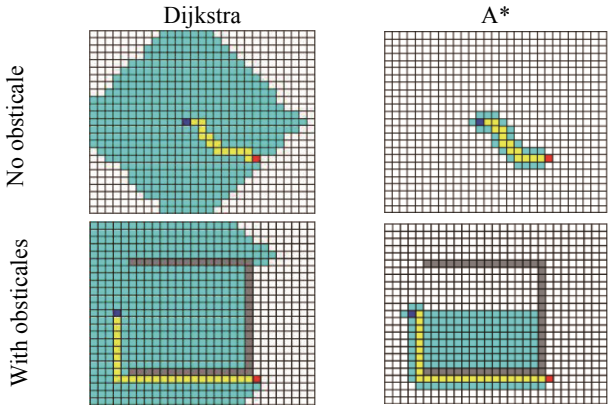


Fig. 13. Dijkstra vs. A*.

However, there are two other problems here. First, this method needs a path finding algorithm for arranging the moving path. Second, there might be an efficiency issue, because it simply arranges path for every undiscovered grid. These two problems are to be dealt with in the following section.

IV. PATH FINDING

1. Path Finding Algorithms

There are three well-known algorithms that can be used to find an optimal path: Dijkstra, Ant and A* algorithm.

Dijkstra is a greedy algorithm that solves a single-source shortest-path problem when all edges have non-negative weights. It starts at the source vertex, S. And grows a tree, T, that ultimately spans all vertices reachable from S. Vertices are added to T in order of distance i.e., first S, then the vertex closest to S, then the next closest, and so on. The original Dijkstra runs in time $O(|V^2|)$, where $|V|$ is the number of nodes in the tree. This algorithm can be improved by min-priority queue (implemented by a Fibonacci heap) and running in $O(|E| + |V|\log|V|)$, where $|E|$ is the number of edges. This algorithm guarantees to find the optimal path, However, the performance is poorer than A* algorithm (next section), which has the worst case $O(|E|)$. Fig. 13 illustrates the search path of Dijkstra and A*.

On the other hand, the original idea of ant algorithm comes from observing the exploitation of food resources among ants, in which ants' individually limited cognitive abilities have collectively been able to find the shortest path between a food source and the nest. The first ant finds the food source, via any way, then returns to the nest, leaving behind a trail pheromone. Ants indiscriminately follow any possible ways, but the strengthening of the runway makes it more attractive as the shortest route. Ants take the shortest route; long portions of other ways lose their trail pheromones. According to LW Santoso's study, ant algorithm is less stable and requires a long time to do a search.

2. Conventional A* Algorithm

A* uses the following function to estimate the cost of a

possible path:

$$f(n) = g(n) + h(n) \quad (10)$$

It divides the path into two parts, and evaluates these two parts separately:

1. g(n): The cost from the start point to current node.
2. h(n): The estimated cost from current node to the goal.

The separating point is called a node, we now keep the total estimated cost f(n) on this node.

The conventional A* has the following pseudo-code. We usually use OPEN and CLOSED list to record nodes. The OPEN list contains those nodes that are candidates for examining. Initially, the OPEN list contains only one element: the starting point. The CLOSE list contains those nodes that have already been examined. Initially, the CLOSE list is empty.

```
1   Add START to OPEN list
2   while (OPEN is not empty)
3   {
4       Get node n from OPEN that has the lowest f(n)
5       move n to CLOSED
6
7       if (n is GOAL)
8          return path
9
10      for (each n' == CanMove (n, direction))
11      {
12         g(n') = g(n) + MOVECOST
13         h(n') = Manhattan(n')
14         f(n') = g(n') + h(n')
15
16         if (n' in OPEN || CLOSED list)
17            if (new n' is not better)
18               continue
19         remove n' from OPEN
20         remove n' from CLOSED
21
22         add n' to OPEN
23         set n as a parent of n'
24      }
25  }
26  if we get to here, then there is No Solution.
```

According to the pseudo-code, when GOAL is moved to CLOSED list, A* returns the path. But what is this "path" ? To understand this "path", let us take a look on line 23. Here we learned each node in A* has its parent. When GOAL is moved to CLOSED list, it means we can find GOAL's parent (and the parent of GOAL's parent, and so on). Just move along with parents in CLOSED list, we eventually have the whole return path.

Use A* algorithm, we should be able to find a valid path for the robot. This solves the first problem we have mentioned earlier.

To solve the second problem, we look closer to the pseudo-code again. That "No Solution" on the last line means: "To make sure there is no valid path for the robot, we have to walk through the whole algorithm".

"No Solution" is the worst case of A*. In Fig. 12, The grids in the hatched area are all instances of A* worst case. They are the reason why our exploration method encounters an efficiency problem. So, the idea of solving this efficiency problem is to reduce the incidence of A* worst case.

3. Isolated Area & Flood-Fill Algorithm

To reduce the incidence of A* worst case. We look carefully at Fig. 12 again, the hatched area is isolated to the robot. If we visit all the grid cells in hatched area one by one, it will be a time-consuming task. For example, if there are 1000 grid cells in hatched area, the robot will use A* algorithm on all 1000 cells, which takes a lot of time.

In order to solve this problem, we find , in Fig. 12, all the grid cells in hatched area are all adjacent to each others. If we can apply flood-fill algorithm (George Law, 2013) on this area, it will be able to gather all the cells in the same area together. And then, no matter how many cells in the hatched area are, the robot only has to use A* algorithm once, which takes time O(1).

Therefore, whenever the robot find an unreachable (No Solution) grid, flood-fill algorithm is applied to mark all grids in the same area. In this way, no extra cost on these grids is spent.

4. Lazy Evaluation

In addition to flood-fill algorithm, the A* can also be improved by using lazy evaluation. Consider the situation in Fig. 14(a), the robot is in a room, and the moving target is marked with a cross. Now, is it efficient to evaluate the whole path? The answer is obviously not. The latter part of the path has a good chance to be located behind a wall. If we can cut short the path, it will speed up the A* path-finding process.

Consequently, we modify the A* pseudo-code as follows:

```
……
4           Get node n from OPEN that has the lowest f(n)
5           move n to CLOSED
6
7           if (n is GOAL || UNCERTAIN_CELL encountered)
8              return path
……
```

The difference between the conventional A* and the modified A* is the UNCERTAIN_CELL, which means whenever A* encounters an "Uncertain cell", the A* evaluating process stops.

But why? According to the pseudo-code of A*, the time complexity of A* increases with the length of the path. The longer the path, the more time A* takes. In the case of Fig. 14(a), the latter part of the path has a good chance to be located behind a wall. Return path on the half way such as Fig. 14(b) is a time-saving strategy. Even if the latter part of the path is not located

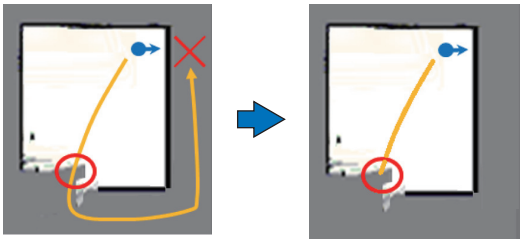


Fig. 14. Lazy evaluation of A*.

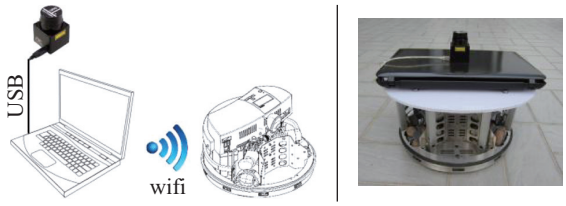


Fig. 15. Robot hardware setting.

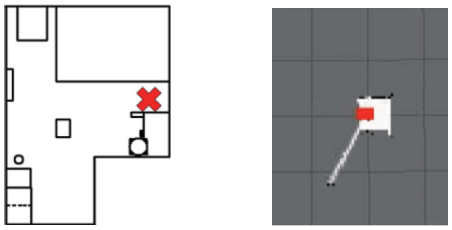


Fig. 16. Indoor space and initial condition of the robot.

behind a wall, we can always call another A* process and construct another path.

V. EXPERIMENTAL RESULTS

In this section, the above-mentioned algorithms are implemented to create an autonomous robot. The hardware of the robot system, as shown in Fig. 15, includes a Festo Robotino robot, a Hokuyo laser rangefinder (sensor), and a laptop computer (controller). Note that the Robotino is an omnidirectional robot, it's easier to move in all directions; and the rangefinder can scan an angle of 240° with detection distance between 2 cm to 4 m.

This experiment took place in a room with obstacles, as shown in Fig. 16. The room is about 35 square meters. The cross sign marks the initial position of the robot. The robot was facing to the right from the beginning, it could only see a small area in the front initially.

According to the map exploration logic, The robot searched for the nearest uncertain cell, which was located behind the robot (denoted by a yellow plus sign). The robot turned over, as shown in Fig. 17, and discovered some extra area.

The second step, the robot searched for the next uncertain cell in the map. This time, the uncertain cell was located on the south of the robot. So the robot used A* algorithm, and found a valid path to the cell. As shown in Fig. 18. In the first two steps,

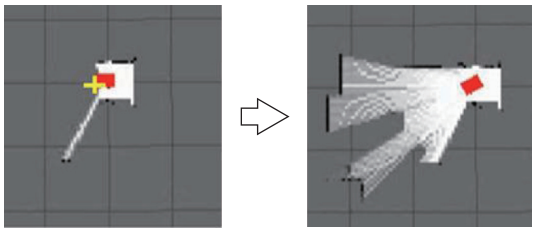


Fig. 17. First step of robot navigation.

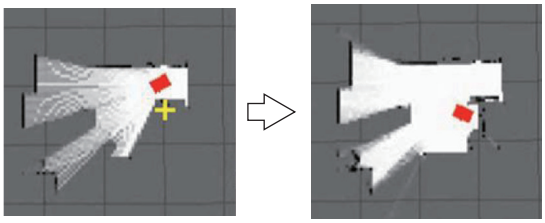


Fig. 18. Second step of robot navigation.

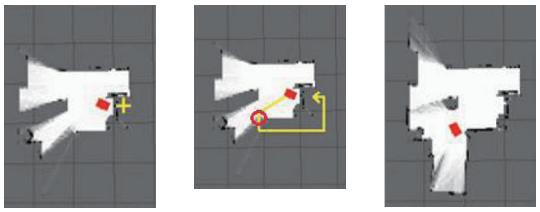


Fig. 19. Third step of robot navigation.

Fig. 20. Fourth step of robot navigation.

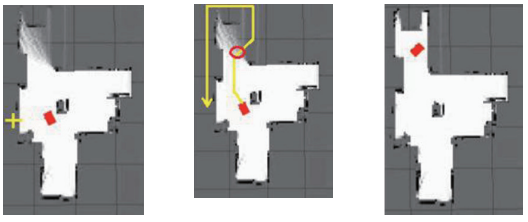


Fig. 21. Fifth step of robot navigation.

there was no obstacles in the way.

Next step (see Fig. 19), the robot found an uncertain cell on the right. However, according to lazy evaluation logic, the latter part of the path was located on the unknown area, we don't have to complete the whole A* process. So the path was cut in half to save time.

The lazy evaluation speed-up strategy also affected the next two steps, as shown in Figs. 20 and 21.

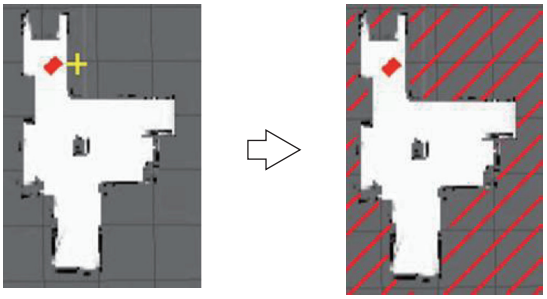


Fig. 22. Sixth step of robot navigation.

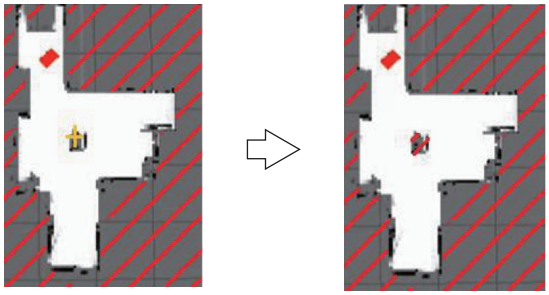


Fig. 23. Seventh step of robot navigation.

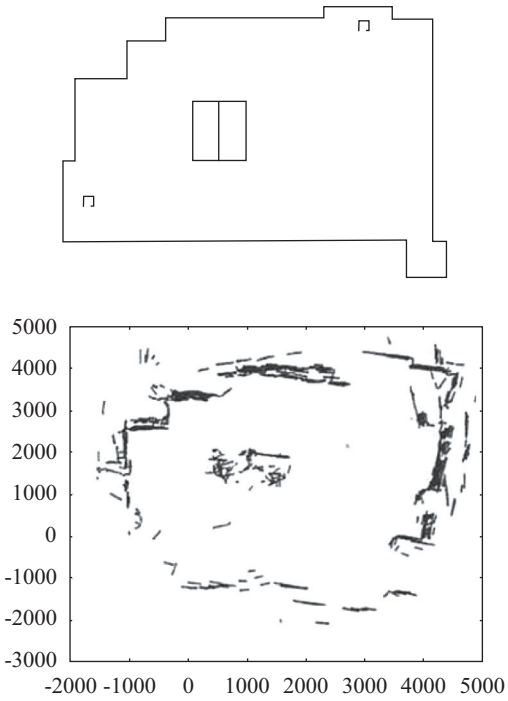


Fig. 24. Map building result with EKF SLAM.

Now, the robot found an uncertain cell on the right, but it couldn't find a valid path to reach it, as shown in Fig. 22. Therefore, the flood fill was used to mark the whole isolated area, and no more cost was spent on this area ever since.

In the final step, there was a small area on the center that was out-of-reach too, as shown in Fig. 23. Flood fill algorithm was employed again to mark this area as well, thus completing the construction of the 2-D map.

VI. DISCUSSION AND CONCLUSION

In this paper, integration of different algorithms was carried out for building an autonomous robot for navigating an unknown environment. First of all, ICP algorithm was used for map building and localization. And then, a map exploration method was proposed, it uses the flood fill algorithm to eliminate unreachable area for reducing the computational time. Furthermore, the conventional A* algorithm was improved by using lazy evaluation.

Without flood fill and lazy evaluation strategies, the experiment took more than 30 minutes to finish. With flood fill and lazy evaluation, the worst case was blocked out and the process was accelerated. In this experiment, it took less than 3 minutes to do the same job.

By comparing Fig. 16 and Fig. 23, the room contour is depicted well by the robot, except for some corners that are too narrow for the robot to get through. Compare this study with another similar one (Bao, 2007), which uses EKF SLAM (see Fig. 24).

The room size is similar to ours. The difficulties of using EKF SLAM come from the existence of uncertainties in a real environment such as sensor noises. Our study uses another approach-Iterative Closest Point (ICP) for building map and localization. Together with efficient path finding and map exploration algorithms, an autonomous robot capable of navigating an unknown indoor environment with random obstacles can be developed.

ACKNOWLEDGEMENTS

This work was supported by Minister of Science and Technology, Taiwan, under grant number NSC 102-2221-E-027-039.

REFERENCES

Arun, K., T. Huang and S. Blostein (1987). Least-squares fitting of two 3-D point sets. IEEE Trans. PAMI 9(5), 698-700.

Bandera, A., C. Urdiales and F. Sandoval (2001). An hierarchical approach to grid based and topological maps integration for autonomous indoor navigation, in Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst., 883-888.

Bao, J. (2007). Study on geometric map building and robot localization in unknown indoor environments, Master Thesis, Tianjin University, China. (in Chinese)

Besl, P. J. and N. D. Mckay (1992). A method for registration of 3-D shapes. IEEE Transactions on Pattern Analysis and Machine Intelligence 14(2), 239-256.

Blais, G. and M. Levine (1995). Registering multiview range data to create 3D computer objects. IEEE Transactions on Pattern Analysis and Machine Intelligence 17, 820-824.

Chen, Q., U. Ozguner and K. Redmill (2004). Ohio state university at the 2004 darpa grand challenge: developing a completely autonomous vehicle. IEEE Intell. Syst. 19(5), 8-11.

Chen, Y. and G. Medioni (1991). Object modeling by registration of multiple range images. IEEE International Conference on Robotics and Automation 3, 2724-2729.

Elfes, A. (1989). Using occupancy grids for mobile robot perception and navigation, Computer 22(6), 46-57.

Fang, G., G. Dissanayake, N. M. Kwok and S. Huang (2005). Near minimum time path planning for bearing-only localization and mapping. In Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst., 850-855.

Forlizzi, J. and C. DiSalvo (2006). Service robots in the domestic environment: a study of the roomba vacuum in the home. In Proc. the 1st ACM SIGCHI/SIGART Human-robot interact., 258-265.

Law, G. (2013). Quantitative comparison of flood fill and modified flood fill algorithms. International Journal of Computer Theory and Engineering 5(3), 503-508.

Guivant, J. E., F. R. Masson and E. M. Nebot (2002). Simultaneous localization and map building using natural features and absolute information. J. Robot. and Auton. System. 40(31), 79-90.

Horn, B. (1987). Closed-form solution of absolute orientation using unit quaternions. JOSA A. 4(4), 629-642.

Horn, B., H. Hilden and S. Negahdaripour (1988). Closed-form solution of absolute orientation using orthonormal matrices. JOSA A. 5(7), 1127-1135.

Johnson, A. and S. Kang (1997). Registration and integration of textured 3-D data. Proc. 3DIM, 234-241.

Kurabayashi, D., J. Ota, T. Arai and E. Yoshida (1996). Cooperative sweeping by multiple mobile robots. In Proc. IEEE Int. Conf. Robot. Autom., 1744-1749.

Koenig, S., B. Szymanski and Y. Liu (2001). Robotic exploration as graph construction. Ann. of Math. and Artif. Intell. 31, 41-76.

Kuipers, B. and Y. T. Byun (1991). A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. J. Robot. and Auton. System. 8, 47-63.

Levinson, J., J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt, M. Sokolsky, G. Stanek, D. Stavens, A. Teichman, M. Werling and S. Thrun (2011). Towards fully autonomous driving: Systems and algorithms. In IEEE Intell. Vehicles Symposium (IV), 163-168.

Masuda, T., K. Sakaue and N. Yokoya (1996). Registration and Integration of Multiple Range Images for 3-D Model Construction, Proc. CVPR 1, 879-883.

Pulli, K. (1999). Multiview registration for large data sets. Second International Conference on 3-D Digital Imaging and Modeling, 160-168.

Rusinkiewicz, S. and M. Levoy (2001). Efficient variants of the ICP algorithm. Proceedings Third International Conference on 3-D Digital Imaging and Modeling, 145-152.

Santoso, L. W., A. Setiawan and A. K. Prajogo (2010). Performance analysis of Dijkstra, A* and ant algorithm for finding optimal path Case Study: Surabaya City Map, MICEEI 2010, 27-10-2010-28-10-2010.

Schroter, D., T. Weber, M. Beetz and B. Radig (2004). Detection and classification of gateways for the acquisition of structured robot maps. In Proc. of 26th Pattern Recognition Symposium (DAGM), 553-561.

Smith, R. C. and P. Cheeseman (1986). On the representation and estimation of spatial uncertainty. The International Journal of Robotics Research 5(4), 56-68.

Smith, R. C. and P. Cheeseman (1986). Estimating uncertain spatial relationships in robotics. Proceedings of the Second Annual Conference on Uncertainty in Artificial Intelligence, 435-461.

Thompson, P., E. Nettleton and H. Durrant-Whyte (2011). Distributed large scale terrain mapping for mining and autonomous systems. Intelligent Robots and Systems, 4236-4241.

Tong, X., T. Furukawa and H. Durrant-Whyte (2013). Computational modeling for parallel grid-based recursive Bayesian estimation: parallel computation using graphics processing unit. Journal of Uncertainty Analysis and Applications, 1-15.

Turk, G. and M. Levoy (1994). Zippered Polygon Meshes from Range Images. Proc. SIGGRAPH, 311-318.

Weik, S. (1997). Registration of 3-D partial surface models using luminance and depth information. International Conference on 3-D Digital Imaging and Modeling, 93-100.

Walker, M., L. Shao and R. Volz (1991). Estimating 3-D location parameters using dual number quaternions, CVGIP: Image Understanding 54(3), 358-367.