AN ENHANCED PATH-OPTIMIZATION ALGORITHM FOR REAL-TIME INTELLIGENT TRANSPORTATION SYSTEMS

Huynh Ngoc Hoi, Nguyen Thai Ngoc Khuong, Vo Thanh Chau, Dang Xuan Ba
Ho Chi Minh City University of Technology and Education, Vietnam

Received 21/01/2020, Peer reviewed 13/02/2020, Accepted for publication 27/02/2020.

ABSTRACT

Development of intelligent robot systems that could efficiently transport goods with the minimal time and high accuracy is an important demand for increasing the productivity of the factory. This paper reports new technologies to improve performance of the Dijkstra algorithm for the path-finding application of the transportation systems. Drawbacks of the conventional searching method are first studied for real-time situations. Two important hypotheses are then considered for the practical applications at which number of the passing nodes and the shape of the passing routine are sufficient conditions for an optimal path. Implementation of the two new features are conducted based on analyses of the conventional one and noting additional optimal objectives. The modified algorithm was successfully deployed and verified both in simulation and experiments. The testing results confirm that the proposed method could reduce the routing time of the robot system down to 14.5% as compared to the classical algorithm.

Keywords*: Dijkstra algorithm; Path-optimization; Intelligent transportation system; robot system; real-time path-finding algorithm.*

1. INTRODUCTION AND MAP OF CONTROL

In practical manufacturing, to achieve the maximum capacity of plants, speed of production circulation needs to be optimized in all stages: from feeding, processing to sorting and warehouse arrangement. If the transportation process is mainly operated by workers, it requires large workspaces and is difficult to reach the factory's goal. Therefore, it is necessary to design an intelligent transport system that can optimize the robot's orbit with the smallest cost. A basic task in this field is to find the shortest path from a current position to a given destination (within a graph of nodes). For this problem, a number of algorithms have been developed.

The Dijkstra algorithm [1] finds the shortest path from a given node i to a single destination node or all other nodes inside a graph. It also supports a bi-directional searching feature. However, the algorithm could only apply to the optimal problem with nonnegative travel cost. The A* Algorithm [2] is a more general approach as compared to Dijkstra's algorithm in solving the minimal path between two nodes in a graph. The order that nodes are selected in the optimization process could be determined by the A* algorithm using a heuristic technology [3]. In fact, Dijkstra's algorithm is a special case of the A* algorithm [4]. An obstacle of the A* algorithm in real-time applications is long processing time spending for the estimation of distances from different nodes.

The Floyd-Warshall's approach [5],[6] is an independent version of the shortest path algorithm. The algorithm could compute all shortest paths from each node to all others within a single loop. It means that the system does not need to re-find the best solution when the destination is changed. However, slow computation and storage ability are main drawbacks. To increase the computation performance, the Bellmann-

Ford algorithm [7]-[8] was developed in which the shortest paths to all nodes only from a starting node are calculated at a single iteration. Although its execution time is still larger than that of Dijkstra's algorithm, it is possible to apply for negative-weight-cost problems.

Among the advanced searching methods, Dijkstra's algorithm could be a proper solution for applications that require the fast computation. However, the algorithm needs to be modified for real-time situations. In this paper, we propose new approaches for the shortest path that is represented in terms of the minimum travel time. To realize this objective, improvements are proposed to select the really shortest routine constructed with minimum nodes passing or with maximum number of the straight lines. The new features have been successfully implemented and validated both in simulation and real-time experiments. The verification results show that the travel time using the proposed algorithm was effectively reduced to 14.5% as compared to the previous one.

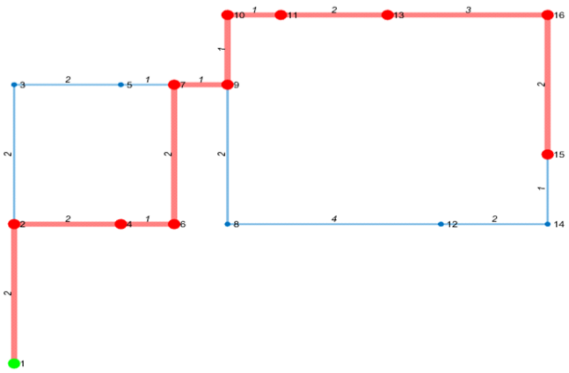
The remaining of the paper is organized as follows: Section 2 presents new features of the shortest-path planning algorithm, Section 3 discusses the validation results of the improvement proposed, and the paper is then concluded in Section 4.

2. OPTIMAL ALGORITHM OF THE SHORTEST-PATH APPLICATIONS

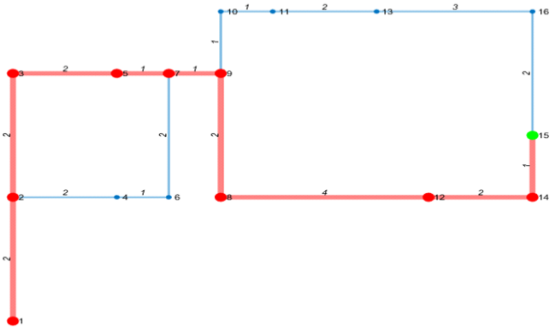
In this research, the Dijkstra algorithm is selected to apply for the shortest-path application thanks to the high computation speed. It has been widely adopted in (=to) navigation and data transmission systems.

2.1 The conventional Dijkstra algorithm

The key idea of the Dijkstra method is begun with a set of unvisited nodes and computation of the tentative distances from node *s* (*the starting node*) to all other nodes *k*. Furthermore, a list of the previous nodes on the path from node *s* to a certain node *j* is structured as *Prev[j]*. Note that, initial values of all distances are set to be infinity [1], [9]. The searching process is started from a current node *i* belonging to the unvisited set, which has the smallest distance from the starting node *s*. All of its unvisited neighbors (connected by an edge) are next scanned and their tentative distances to the node *i* are calculated. After one iteration, the current node is deleted from the set of unvisited nodes. The node that has the shortest distance to node *i* (i.e., the node which has the minimum value in *Dist[k]*) is considered as the next current one. Once the destination node *e* has been removed from the unvisited set, or all nodes have been considered and the unvisited set is empty, the algorithm will be stopped. The tentative distance of investigating node *k* is updated if the current stored distance from the starting node *s* to the node *k* (given by *Dist[k]*) is greater than the term (*Dist[i] + d(k, i)*). In this case, *Dist[k]* is overwritten by *Dist[i] + d(k, i)* and *Prev[k]* is set to node *i*.



a) Forward optimal path



b) Backward optimal path

***Figure 1**. Performance evaluation of the conventional Dijkstra algorithm*

Performance of the conventional Dijkstra algorithm has been evaluated through simulation. A graph of total 15 nodes was set up as depicted in **Fig. 1**(a) and (b). The first test was conducted with the starting node (***1***) and the ending node (***15***). The testing result is plotted in Fig. 1(a). As seen in the figure, the minimum distance in this case is 17 along with the optimal list visited of (***1, 2, 4, 6, 7, 9, 10, 11, 13, 16, 15***). A reverse routine of the first test was chosen for the second test at which the starting node was (***15***) and the ending node is (***1***). The optimal result is illustrated in **Fig. 1**(b) in which the same minimum distance of 17 was found, but interestingly the visited list was different, in detail, of (***15, 14, 12, 8, 9, 7, 6, 3, 2, 1***). In fact, this example had multiple optimal solutions with the criterion of the minimum travel distance. The conventional Dijkstra algorithm only searches for the first optimal one. Note that in real-time situations, different shapes and structures of optimal routines will result in different travel times. To apply this algorithm in real-time path-finding applications with the optimal passing time, the following hypotheses hence need to be taken into account [**10**]-[**12**]:

Hypothesis 1: with the same travel distance, the less nodes passing will result in the shorter travelling time.

Hypothesis 2: with the same travel distance and same number of passing nodes, the larger number of the straight lines passing will result in the shorter travelling time.

Integrating the new features proposed to the conventional searching method is not trivial work. In the next sections, two improvements for this method are studied.

2.2 Improvement with the minimal-passing nodes

To ease the understanding of the proposed idea, an example of a routine map is considered as plotted in **Fig. 2**. It can be seen in the figure that to go to node ***H*** from node ***A***, there are two possible shortest ways, as follows: The first optimal way is the list of passing nodes (***A, D, F, E***, and ***H***), and the second one is of (***A, B, C, F, E***, and ***H***). Both cases have the same length of 10, but the first case passes only 5 nodes while the second path goes through 6 ones. The reason is that in the length-update step of Dijkstra's algorithm, if the previous length is equal to the current length considered (although passing more nodes), this algorithm will not update again.

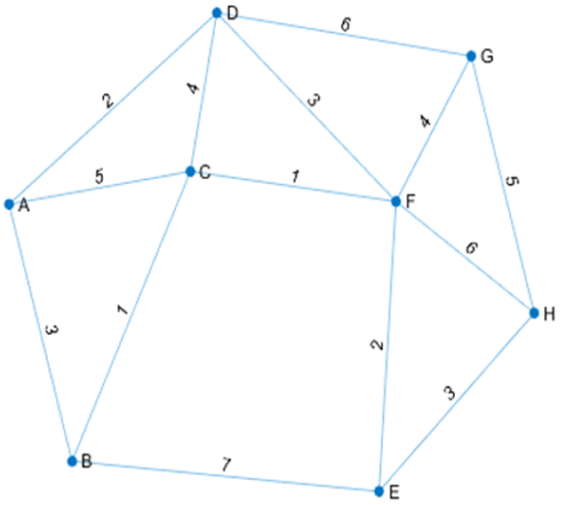


Figure 2*. An example of the searching map*

As a result, to solve the aforenoted problem, the proposed solution is as follows. In general, for each iteration, the algorithm only produces a single candidate routine. In order to find the shortest path with the best routine, it needs to find all the paths from the starting node ***s*** to the nodes (near the end node ***e***. The best result is obtained by comparing the two input values of the number of nodes passed and the shortest length. The idea is illustrated more in detail in **Fig. 3** and **Table 1**.

*Comment 1***:** With the new feature, the algorithm can not only find the smallest path, but also optimize the number of nodes passing through. The algorithm will search all possible paths to the destination with the minimum value and then compare to make the optimal choice. It means that an intensive additional option to improve the performance of the searching algorithm has been supported for users.

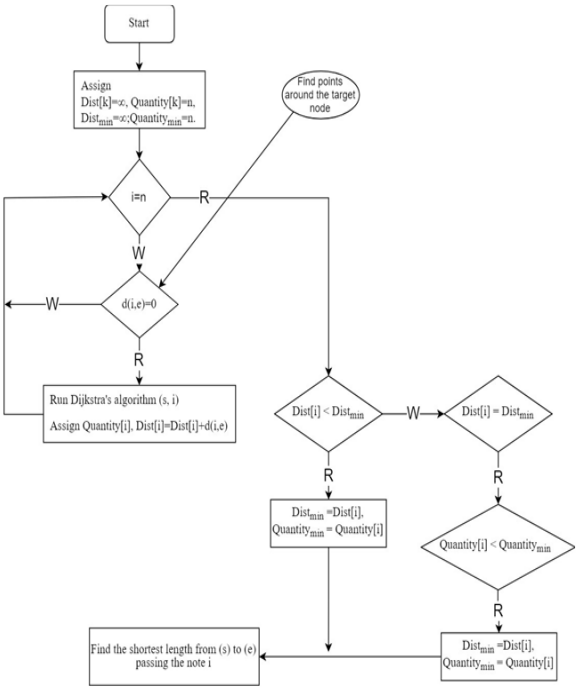


***Figure 3**. Flowchart of the shortest-path-finding algorithm with minimal passing node*

***Table 1.** Implementation procedure for the shortest-path-finding algorithm integrated with the minimal-passing-node feature.*

Algorithm:

T is the set of points around the target node.

Quantity is the value representing the number of nodes to pass.

Step 1: Assign Dist[k]=∞, Quantity[k]=n, $\forall k \in N$

$Dist_{min}$=∞; $Quantity_{min}$=n.

Step 2: $\forall i \in T$

- Run Dijkstra's algorithm (s, i)
- Assign Quantity[i], Dist[i]=Dist[i]+d(i,e);

Step 3:

Find min of Dist[]

If Dist[i] < $Dist_{min}$, $Dist_{min}$ =Dist[i], $Quantity_{min}$ = Quantity[i]

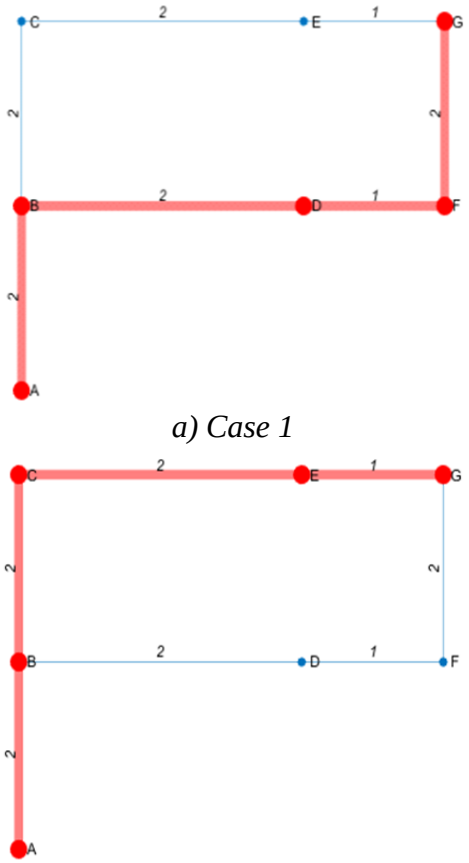
Elseif Dist[i] = $Dist_{min}$ compare Quantity[i] with $Quantity_{min}$

If Quantity[i] < $Quantity_{min}$ select Dist[i] = min

$Quantity_{min}$ = Quantity[i].

Step 4: Find the shortest length from (s) to (e) passing the note in **Step 3.**

2.3 Improvement with straight-line optimization



a) Case 1

b) Case 2

***Figure 4**. Two optimal cases of the Dijkstra's algorithm*

To intuitively study the straight-line optimization for the Dijkstra algorithm, another example of routine map is considered as plotted in **Fig. 4**. It can be seen in the figure that to go from node ***A*** to node ***G***, there are also two possible shortest ways, as follows. The first optimal way is the list of passing nodes (***A, B, D, F***, and G), and the second one is of (***A, B, C, E***, and ***G***). Both cases have the same length of 7, but the first optimal case contains three straight lines while the second path has only two lines. According to the *Hypothesis 2*, the real-time passing time of the second optimal routine would be shorter than the other.

The core idea for this improvement is (the) adoption of a new variable which is assigned to comparative paths. The detailed implementation procedure of the new feature is expressed in **Fig. 5** and **Table 2**.

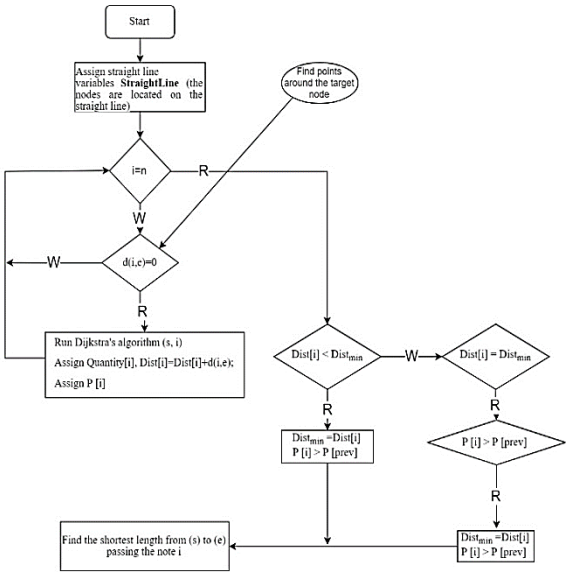


Figure 5*. Flowchart of the shortest-path-finding algorithm with optimal straight lines*

Table 2*. Implementation procedure for the shortest-path-finding algorithm integrated with optimal straight lines.*

Algorithm:

T is the set of points around the target node.

StraightLine is the variable used to identify sets of three consecutive nodes that create a line.

P[] is the number of **StraightLine** per routine

Step 1: Assign straight-line variables **StraightLine** (the nodes are located on the straight line), $Dist_{min}=\infty$;

Step 2: $\forall i \in T$

- Run Dijkstra's algorithm (s, i)
- Assign Quantity[i], Dist[i]=Dist[i]+d(i,e);
- Assign P [i]

Step3: Find min of Dist[]

- If Dist [i] = $Dist_{min}$, compare P [i] with P_{min}
- If P [i] > P_{min} then select Dist[i] = min, P [i]=P_{min}

Step 4: Find the shortest length from (s) to (e) passing the note in **Step 3**

Comment 2: This feature is the second optimal criterion for the path-finding algorithm. By adding "StraightLine" variable, the algorithm could identify the straight lines in the map, which then supports for the selection of the real-optimal routine. Another interesting additional option has been integrated for users to improve the performance of the searching algorithm.

3. VALIDATION RESULTS

In this section, the improved algorithms are carefully investigated both in simulation and real-time environments by comparative experiments

3.1 Simulation Results

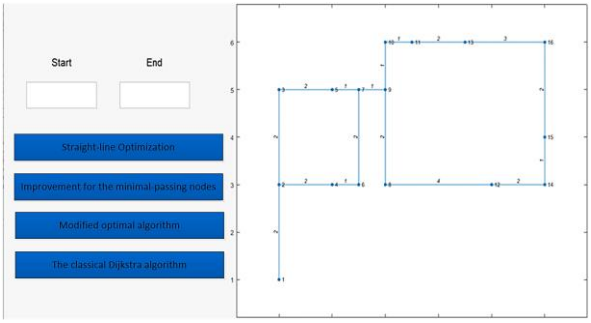


Figure 6*. Simulation interface*

A simulation interface had been built, as shown in **Fig. 6**, to support the interaction of users and clearly observe performance of each algorithm. The interface consists of two parts: the left-hand side part allows users to set desired nodes with four running modes (The classical Dijkstra algorithm, Straight-line optimization, Improvement for minimal passing nodes, and Modified optimal algorithm), while the opposite side is used to present the detailed passing routine.

For representing the real-world behaviors of traffic systems, the aforementioned hypotheses could be applied to the simulation as follows: the average moving velocity of vehicles between two direct-connection nodes are 4/s; the times to passing throughout intersections in a straight direction and in a turn direction are 0.75s and 1.5s, respectively.

In the simulation test, the starting node was chosen to be 1 while the end node is 15. The running results of the algorithms are shown in **Figs. 7** to **10**. By applying the conventional Dijkstra algorithm, as seen in **Fig. 7**, the optimal path was [***1, 2, 4, 6,7, 9,***

10, 11, 13, 16, 15] with the shortest distance of 17. However, in the case of using the Dijkstra method with the minimal-node enhancement, as shown in **Fig. 8**, the shortest of 17 was still obtained, but the number of passing nodes was reduced to [***1, 2, 4, 6,7, 9, 8, 12, 14, 15***]. As a result, the passing time (14.75s) of the minimal passing-node method was faster than that of the classical one (15.5s). On the other hand, the Dijkstra algorithm was also updated by using the second optimal criterion of the passing straight lines. The testing result on the same map is displayed in **Fig. 9**, in which the shortest path of 17 was also found along another optimal path of [***1, 3, 5, 6,7, 9, 10, 11, 13, 16, 15***]. Even though the three optimal routines have the same distance, travelling time is different: Applying the straight-line optimization also results in the smaller moving time of 14s as compared to the conventional method. Another interesting approach is the combination of the studied solutions. Its simulation result is depicted in **Fig. 10**. This approach would consider both the second-level optimal criterions (minimal passing node and straight line). From **Fig. 9**, a new optimal routine of [***1, 3, 5, 6,7, 9, 8, 12, 14, 15***] was found. The total length of the new path is still 17, but the travelling time is the smallest one of only 13.25s. The simulation results are summarized in **Table 3**. The table reveals that using the improvements for the Dijkstra algorithm could reduce the traveling time (14.5% as compared to the conventional method) and lead to low operating costs while maintaining stability of the vehicles.

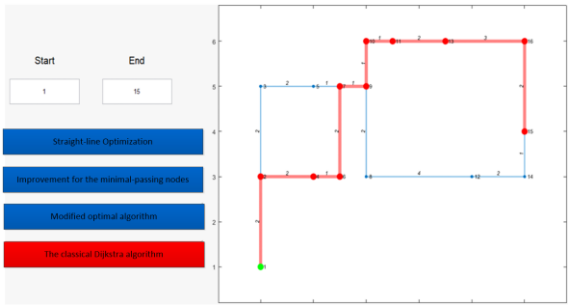


Figure 7. *Searching result of the classical Dijkstra' algorithm*

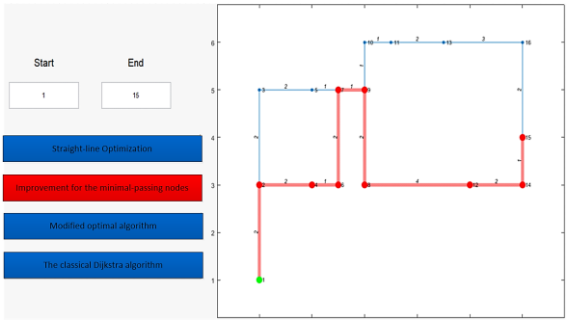


Figure 8. *Searching result of Dijkstra' algorithm improved by a minimal passing-node feature*

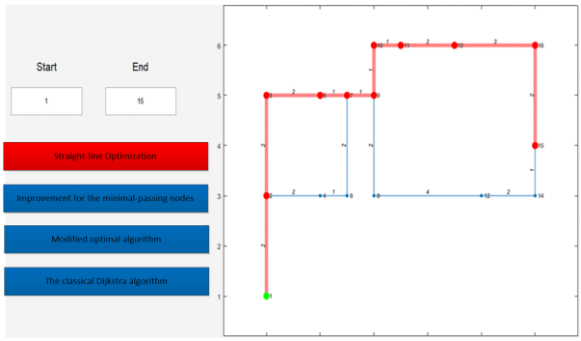


Figure 9. *Searching result of Dijkstra' algorithm improved by optimal straight lines*

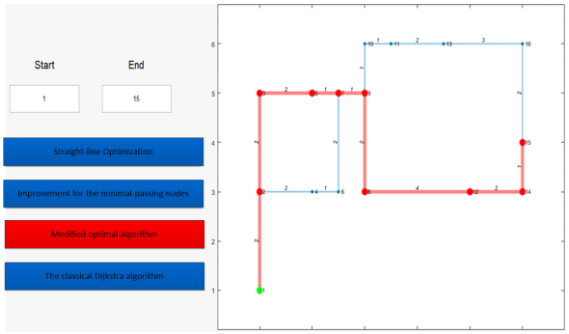


Figure 10. *Searching result of Dijkstra' algorithm improved by optimal straight lines and the minimal passing node*

Table 3. *Performance of the three searching methods for a certain case*

Optimization methods	Moving distance	Passing nodes	Travelling time
The classical Dijkstra's algorithm	17	[***1, 2, 4, 6,7, 9, 10, 11, 13, 16, 15***]	15.5s
Improvement with the minimal passing node	17	[***1, 2, 4, 6,7, 9, 8, 12, 14, 15***]	14.75s

Optimization methods	Moving distance	Passing nodes	Travelling time
Improvement with optimal straight lines	17	[***1, 3, 5, 6,7, 9, 10, 11, 13, 16, 15***]	14s
Improvements with both features	17	[***1, 3, 5, 6,7, 9, 8, 12, 14, 15***]	13.25s

3.2 Real-time Experiments

a) Setup

To verify the performance of the optimal searching methods in a real-time system, a line-tracking robot was fabricated. Motion of the robot was controlled by PID controllers applied to two GA25 motors according to the feedback signals of 7-line detectors. Whole the optimal algorithms and controllers were implemented on an Arduino Mega 2560 processor.

b) Results

The travelling map, starting node and end node of the experiments were set up as same as those presented in the simulation. Travelling time of the four searching methods in 10 running times were collected and summarized in **Fig. 11**. As seen in the figure, the average travelling time of the robot using the conventional Dijkstra algorithm, the minimal-passing-node, optimal-straight-line improvements and their combination were 12.75s, 12.5s, 11.25s, and 11.2s, respectively. Especially, the moving time of the combination method was significantly reduced from 12% to 15% as compared to that of the conventional one. The obtained results confirm the effectiveness of the proposed features in the real-time situations. Other images captured in the experiments are depicted in **Fig. 12**.

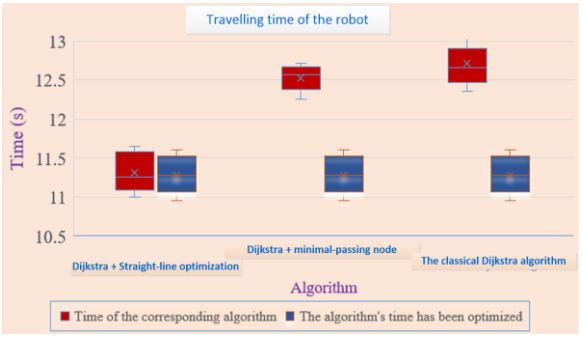


***Figure 11**. Comparative performance of the robot with respect to the searching methods*

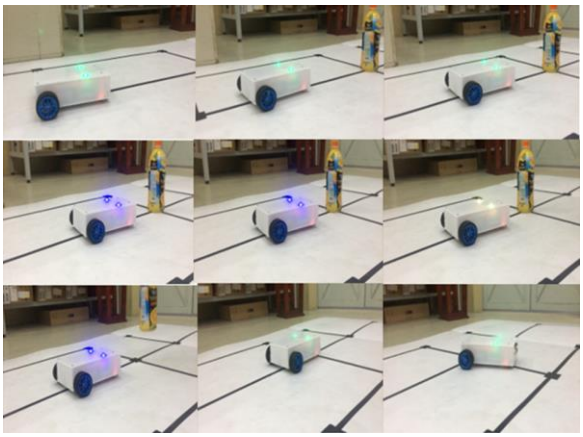


***Figure 12**. Photographs of the real-time implementation system*

4. CONCLUSION

In this paper, improvements have been studied to improve performance of the famous searching method - Dijkstra's algorithm. Shortcomings of the conventional method in real-time aspects were carefully analyzed. To overcome these drawbacks, minimal-passing-node and optimal-straight-line features were proposed as second-level optimal criterions of the searching method. These techniques were successfully implemented and tested both in simulation and real-time experiments. The results achieved show the effectiveness and feasibility of the proposed ideas.

REFERENCES

[1] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik 1*, pp. 269-271, 1959.

[2] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, pp. 100–107, 1968.

[3] P. E. Hart, N. J. Nilsson, and B. Raphael, "Correction to 'A Formal Basis for the Heuristic
Determination of Minimum Cost Paths," SIGART Newsletter, vol. 37, pp. 28–29, 1972.
[4] N. J. Nilsson. Principles of Artificial Intelligence. Palo Alto: Tioga Publishing Company, 1980.
[5] R. W. Floyd, "Algorithm 97: Shortest path," Communications of the ACM, pp. 3-345, 1962
[6] S. Warshall, "A theorem on boolean matrices". Journal of the ACM, vol. 9, pp.11–12, 1962.
[7] R. Bellman, "On a routing problem," Quarterly of Applied Mathematics, vol. 16. pp. 87-90, 1958.
[8] S. Knopp, P. Sanders, D. Schultes, F. Schulz, and D. Wagner, "Computing many-to-many shortest paths using highway hierarchies," In Proceedings of the Workshop on Algorithm Engineering and Experiments, 36–45. New Orleans, Louisiana. .2007, January6.
[9] Y. Huang, Q. Yi, M. Shi, "An improved Dijkstra shortest path algorithm," in The 2nd International Conference on Computer Science and Electronics Engineering, 2013.
[10] F. B. Zhan and C. Noon. 1998. "Shortest path algorithms: An evaluation using real road networks," Transportation Science, vol. 32, no. 1, pp. 65-73, 1998.
[11] T. Willhalm, "Engineering Shortest Paths and Layout Algorithms for Large Graphs," Ph. D. Thesis, Karlsruhe University, 2005.
[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Introduction to Algorithms, 2nd Ed. New York: MIT Press and McGraw-Hill, 2001.

Corresponding author:
Dang Xuan Ba
Ho Chi Minh City University of Technology and Education
Email: badx@hcmute.edu.vn