Journal of Computers Vol. 29 No. 1, 2018, pp. 21-39
doi:10.3966/199115992018012901003

Robot Navigation in Dynamic Environments Based on Fuzzy Controller and the A* Algorithm

Cheng-Hsiung Chiang

Department of Information Management, Hsuan Chuang University, Hsinchu City 407, Taiwan
chchiang@hcu.edu.tw

Received 14 August 2016; Revised 7 February 2017;Accepted 7 February 2017

Abstract. Robot navigation guides a robot moving from a start position to a target position without collision in an unknown environment. This paper proposes a robot navigation approach in a dynamic environment with static and dynamic obstacles. The proposed navigation method comprises static navigation and dynamic path planning (DPP). Static navigation guides the robot to avoid static obstacles by using a fuzzy controller, which comprises four input and two output variables. If the robot detects dynamic obstacles, a trajectory prediction table is generated to predict their moving trajectories. If this table shows that the robot will collide with a dynamic obstacle in the near future, DPP begins determining a safe and short path to avoid the obstacle by using the waiting or detouring strategies. The detouring strategy employs the A* algorithm to determine an optimal adaptable path to avoid a threatening obstacle. If the robot detects only static obstacles, the fuzzy controller guides the robot to avoid them. Simulation and comparison results showed that the proposed method is superior to static navigation.

Keywords: A* algorithm, dynamic path planning, fuzzy controller, robot navigation

1 Introduction

Path planning is essential for mobile robots, which determines an optimal collision-free path from a starting position to a target position in a specific environment according to criteria such as distance, time, or energy [1]. Two main categories of path planning exist: global path planning, which encompasses all the acquired knowledge relating to the whole environment, and local navigation, which is the process of using only the robot's currently sensed information (entire environment information is unknown or partially unknown) [2]. Numerous studies have used several techniques for global path planning, such as particle swarm optimization [3-5], ant colony optimization [6-7], genetic algorithms [8-9], dynamic programming [10], the visibility graph method [11], the free-space method, the artificial potential field method [12], and the A* algorithm [13]. In addition, numerous studies have used techniques such as neural networks [14] and fuzzy logic [15] for local navigation.

Two types of obstacle can be present in an environment: static and dynamic obstacles. To date, the problem related to static obstacles has been thoroughly studied. Therefore, recent studies have focused on dynamic obstacle avoidance [1-2, 16-21]. A mobile robot that performs a task in an indoor or outdoor environment must navigate safely from one place to another; in addition, avoiding moving objects, such as people, pets, and cars, is essential [20].

Phillips and Likhachev [20] developed the concept of safe intervals for path planning in dynamic environments. A safe interval is a period without collisions. A configuration is a set that describes the robot's state such as position, heading, and joint angles. To avoid a moving obstacle, the robot uses the A* algorithm and safe interval to arrange its path. The method reduces the number of states that must be searched, and does not sacrifice the theoretical guarantees on optimality. They showed that their planner could provide the same optimality and completeness guarantees as planning with time as an additional dimension. Experimental results showed that the reduction in state space results in a planner that finds solutions significantly faster than the standard approach.

Raja and Pugazhenthi [1] proposed algorithms for dynamic path planning (DPP), which can be classified into two types: in the first type, static obstacles are avoided using the direction concept, whereas in the second type, dynamic obstacles are avoided using the waiting time concept. The direction concept identifies the maneuverable and nonmaneuverable obstacle edges by using the pass-through and no-pass-through edge concepts. The waiting time concept investigates the required waiting time and stop position for the robot when encountering dynamic obstacles. This method is intuitive and may be easy to implement.

Faisal et al. [16] developed four fuzzy logic modules—goal seeking module, static and dynamic obstacles avoidance module (SDOAM), emergency module (EM), and robot setting module—for robot navigation in a dynamic and unknown indoor environment. The method is implemented using the Powerbot robot. If the laser or ultrasonic sensors detect any obstacle movement near the robot, the EM module is activated; otherwise, the SDOAM module is used. If the laser or ultrasonic sensors detect any static or dynamic obstacle far from the robot, the SDOAM module is used to guide the robot.

In addition, Faisal et al. [22] conducted another related study that developed two fuzzy logic controllers (FLCs)—tracking FLC (TFLC), for driving the robot moving toward its target position, and obstacle-avoiding FLC (OAFLC) for avoiding obstacles—for robot navigation in unknown and dynamic environments. If the robot does not sense any obstacle, the TFLC guides the robot moving toward its target position; otherwise, the OAFLC helps the robot avoid the obstacle. They used the same method, OAFLC, to avoid static and dynamic obstacles. Because OAFLC cannot predict the movement of a dynamic obstacle, the robot is highly likely to collide with the obstacle, as indicated by our simulation results (Section 5).

Kümmerle et al. [18] presented a navigation system for real mobile robots operating in crowded city environments and pedestrian zones. The architecture of a path planner comprises three levels. The top level considers the topology of the environment. The intermediate level applies Dijkstra's algorithm on local maps to calculate way-points, which serve as inputs of the low-level planner. The low-level planner computes velocity commands for the robot and manages the dynamic objects that are present in the robot's vicinity. To detect moving obstacles in the vicinity of the robot, they employed a blob tracker based on two-dimensional range scanner data and a common coordinate frame. The obtained information is used to predict the obstacle's position at a certain time in the planned trajectory. Planning a path from the current location of the robot to a goal location is described as follows. Considering the robot and goal nodes of the map, the high level planner uses the A* algorithm to obtain a list of waypoints toward the goal. However, following the list may result in suboptimal paths. Thus, they used the Dijkstra algorithm in the local map starting from the current location of the robot and selected way-points as intermediate goals for the low-level planner. The proposed navigation system was implemented and demonstrated in a large-scale public field test, and the robot navigated a path longer than 3 km through the crowded city center of Freiburg in Germany.

An intuitive and effective method is to predict the moving trajectories of dynamic obstacles when the robot detects them [2, 14, 18, 20]. The robot must plan short and safe paths to avoid these moving obstacles. One approach [20] assumes that another system exists that tracks dynamic obstacles to predict their future trajectories and formats them into a general representation. A trajectory is only a list of points, where each point has state variables that specifies its configuration, time, and some measure of the point's uncertainty. Another method [14] uses a neural network to predict only the next movement of dynamic obstacles, but the neural network does not predict the future several movements in a period. Inspired by the method proposed by Phillips and Likhachev [20], we presented a trajectory prediction (TP) table in our previous study [23], which provides information about the predicted trajectories of dynamic obstacles and robot and collision judgment, for DPP to plan an optimal avoidance path.

This paper proposes dynamic navigation (DN) for a robot in a dynamic and unknown environment, which is an extension of the method used in our previous study [23]. The proposed DN method includes two parts: static navigation and DPP. Static navigation, inspired by Singh et al. [24], adopts a FLC to avoid unknown and stationary obstacles. The DPP approach incorporated the A* algorithm to plan an optimal adaptable path to avoid dynamic obstacles. Compared with our previous study [23], we made four major improvements to the DPP method. (1) We added an additional objective function, the danger degree, for each movement of the adaptive path such that the adaptive path can be relatively distant from the obstacles. (2) We improved the original waiting strategy (the robot waits for the dynamic obstacle) so that the robot considers different situations more carefully. (3) The direction and speed of each dynamic

obstacle is changeable with a small probability. If the detected dynamic obstacle changes its moving way and threatens the robot, the DPP method is restarted to generate another adaptive path. (4) We used more precise mathematics to revise the DPP algorithm.

2 The Proposed Dynamic Navigation Method

This paper proposes a DN method including the robot navigation approach for avoiding static obstacles and DPP for avoiding dynamic obstacles, as shown in Algorithm 1. The robot must move from a start position to a target position without collision. We assumed that the static and dynamic obstacles are unknown. In the simulated experiment of the robot navigation problem, we assumed that the robot had two types of sensor. Sonar sensors with a maximum sensing range of 130 cm detect the stationary obstacles on the left, front, and right sides of the robot, as shown in Fig. 1. A laser range finder with a maximum sensing range of 250 cm is used to detect omnidirectional moving obstacles. We assumed that the robot has a circular structure with a 10-cm radius.

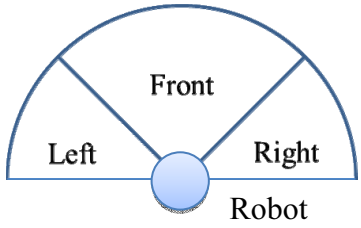


Fig. 1. The illustration of the sensing range of sonar sensors

Algorithm 1: *Algorithm for the DN Method*

1: Set initial parameters: s, g, ε, and v; // s and g: start and target positions of the robot
2: $t = 1$; // initial time step
3: $path(t) = s$; // the robot's initial location $path(t) = (x_t, y_t)$ is at s
4: **while** $Distance(path(t), g) > \varepsilon$ **do**
5: **if** dynamic obstacle is detected **then**
6: $trajTable \leftarrow$ Generate the trajectory prediction table;
7: **end if**
8: **if** threatening dynamic obstacle is not detected around the robot **then**
9: Calculate θ; // included angle between robot's current position and target position
10: **if** no static obstacle is detected around the robot **then**
11: $\Delta\theta = 0$; // $\Delta\theta$: the increment of the robot's steering angle
12: **else**
13: $[\Delta\theta, v]$ = **FuzzyController**(*Left*, *Front*, *Right*, θ); // introduced in Section 3
14: **end if**
15: $t = t + 1$; // time step adds one
16: $\theta = \theta + \Delta\theta$; // the robot turns by $\Delta\theta$ degrees
17: $x_t = x_{t-1} + v \cdot \cos(\theta)$; // calculate the x coordinate of the robot's next position
18: $y_t = y_{t-1} + v \cdot \sin(\theta)$; // calculate the y coordinate of the robot's next position
19: $path(t) = (x_t, y_t)$; // add the new position to *path*
20: Move the robot to the $path(t)$ location;
21: **else**
22: *adaptedPath* = **DynamicPlanning**(*trajTable*); //to DPP algorithm
23: $path = path \cup adaptedPath$;
24: $t = t +$ (the number of positions in *adaptedPath*);
25: **end if**
26: **end while**

In line 1 of Algorithm 1, we first set the initial parameters s, g, ε, and v, where ε is the tolerable error between the robot's current position (ε is set to 20 cm, twice as high as the robot's radius) and target

position and v is the robot's moving speed. The robot's speed was within 20-30 cm/time-step in this study. The initial time step t is set to 1 (line 2). The robot's moving path is recorded using the variable *path*, and its initial value is $path(1) = s = (x_1, y_1)$. The (x, y) coordinate of the robot indicates its center. The **while loop** (lines 4-26) is the primary part of Algorithm 1. In line 4, the condition for executing the **while loop** is $Distance(path(t), g)$, which indicates the distance between the robot's current position at time step t and target position.

In the **while loop**, the robot first detects whether any dynamic obstacle exists around it. If the robot detects a dynamic obstacle within the sensing range of 250 cm, the TP table (Table 2) is generated to predict moving trajectories of the dynamic obstacles (lines 5-7). Numerous studies have used the DPP and predicted the movements of dynamic obstacles for planning a safe path [2, 14, 18, 20]. We assumed that the robot can determine the moving direction and speed of the detected dynamic obstacle, and therefore, we can directly show the moving positions of the dynamic obstacle in the TP table. For example, Table 2 provides the predicted information for the future positions of the robot and detected obstacles from the current time step (e.g., t) to the next several time steps (e.g., $t + 9$). In addition, it evaluates whether the robot collides at each time step. This table was used for planning an adaptive path to avoid the moving obstacle, as described in Section 4.

Lines 8-25 describe the steps for guiding the robot's movement, and there are three types of movement:

(1) The robot moves directly toward the target position (lines 11 and 15-20).

(2) The fuzzy controller is used to guide the robot to avoid the static obstacles (lines 13 and 15-20).

(3) The DPP algorithm is used to obtain a safe and short adaptive path for avoiding the dynamic obstacles (lines 22-24).

The threatening dynamic obstacle in line 8 means that the dynamic obstacle will collide with the robot in the near future; the colliding information is present in the TP table. In line 9, θ is the included angle between the robot's current and target positions, and the robot's orientation is set to θ. In line 11, $\Delta\theta$ is the increment in the robot's steering angle and $\Delta\theta = 0$ indicates that the robot does not turn and directly moves toward the target position. In line 13, $[\Delta\theta, v]$ = **FuzzyController**(*Left*, *Front*, *Right*, θ) obtains the $\Delta\theta$ and v to guide the robot's movement by using the fuzzy controller, as introduced in Section 3, where v is the velocity of the robot. The input parameters, *Left*, *Front*, and *Right*, are the shortest distances between the sensed static obstacle and the robot's left, front, and right sides, respectively. Lines 15-19 calculate the robot's next position denoted using $path(t)$. We can then move the robot to the position $path(t)$.

In line 22, *adaptedPath* = **DynamicPlanning**(*trajTable*) obtains the adaptive path for avoiding dynamic obstacles by using the DPP algorithm, as introduced in Algorithm 2. The number of positions in *adaptedPath* is calculated using the minimal number of movements that the robot can make to exceed its sensing range (250 cm) from its current position. The **while loop** is repeated; the robot can eventually reach or approach the target position, and the robot's trajectory is recorded using the *path* variable.

3 Fuzzy Controller for Static Navigation

Fig. 1 illustrates the sensing range of sonar sensors of the robot [25]. The sensing range is divided into three sectors: *left*, *front*, and *right*. We applied the fuzzy controller presented in our previous study [23] to guide the robot in avoiding the static obstacles. The input variables are the shortest distances between the static obstacle and the **left**, **front**, and **right** sides of the robot (0-130 cm), and the **included angle** between the robot and its target position (0°-180°) represented by θ. The output variables $\Delta\theta$ and v denote the increment in the robot's steering angle (−120°-120°) and the velocity of the robot (10-40 cm/time-step).

The membership functions (Fig. 2) of the input variables *Left*, *Front*, and *Right* are the same. We can use the crisp membership function for input variable θ (Fig. 3). If the value of the included angle θ is less than 90°, it belongs to the right linguistic term; otherwise, it belongs to the left linguistic term. The membership functions of v and $\Delta\theta$ are shown in Figs. 4 and 5, respectively. The membership functions shown in Fig. 2, Fig. 4, and Fig. 5 were first developed in our previous study [23]. We used the MATLAB software, version R2013b, and its fuzzy logic toolbox to construct the Mamdani-type fuzzy inference system [26]. The 28 proposed fuzzy rules are listed in Table 1 [23]. In the fourth column, from rules 2-27, the linguistic terms of the variable θ are "-" which means it can be any value.

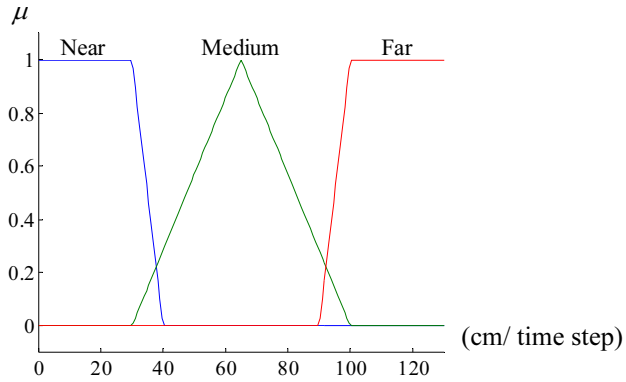


Fig. 2. The membership function of input variable **Left**, **Front**, and **Right** [23]

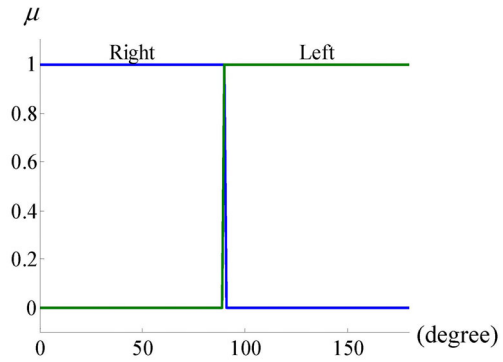


Fig. 3. The membership function of input variable θ

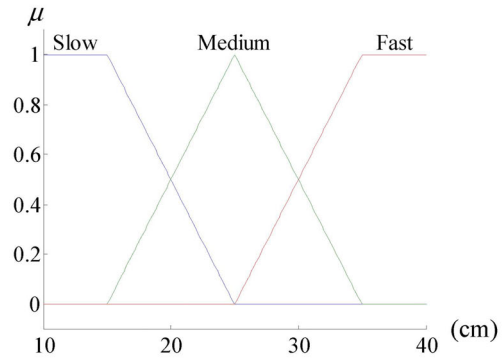


Fig. 4. The membership function of output variable v [23]

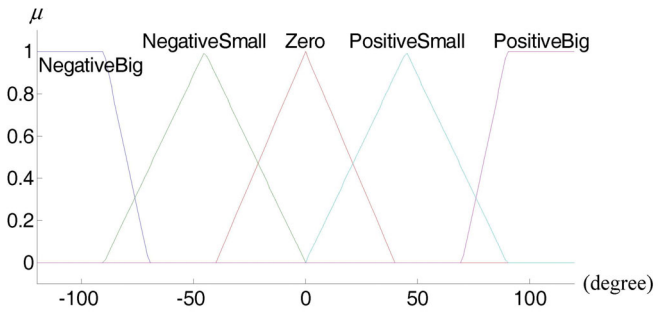


Fig. 5. The membership function of output variable $\Delta\theta$ [23]

Table 1. List of fuzzy rules for static robot navigation [23]

Input variable				Output Variable	
Left	*Front*	*Right*	θ^b	$\Delta\theta^a$	v
Near	Near	Near	Right	NB	Slow
Near	Near	Medium	–	NS	Slow
Near	Near	Far	–	NB	Slow
Near	Medium	Near	–	ZE	Slow
Near	Medium	Medium	–	NS	Medium
Near	Medium	Far	–	NS	Medium
Near	Far	Near	–	ZE	Medium
Near	Far	Medium	–	ZE	Medium
Near	Far	Far	–	NS	Medium
Medium	Near	Near	–	PB	Slow
Medium	Near	Medium	–	NB	Slow
Medium	Near	Far	–	NB	Slow
Medium	Medium	Near	–	PS	Medium
Medium	Medium	Medium	–	ZE	Slow
Medium	Medium	Far	–	NS	Medium
Medium	Far	Near	–	ZE	Medium
Medium	Far	Medium	–	ZE	Fast
Medium	Far	Far	–	ZE	Fast
Far	Near	Near	–	PB	Slow
Far	Near	Medium	–	PB	Slow
Far	Near	Far	–	NB	Slow
Far	Medium	Near	–	PS	Medium
Far	Medium	Medium	–	PS	Medium
Far	Medium	Far	–	NS	Medium
Far	Far	Near	–	PS	Medium
Far	Far	Medium	–	ZE	Fast
Far	Far	Far	–	ZE	Fast
Near	Near	Near	Left	PB	Slow

Note. [a.] NB: Negative Big, NS: Negative Small, ZE: Zero, PS: Positive Small, PB: Positive Big.

The membership functions (Fig. 2) of the input variables Left, Front, and Right are same. We can use the crisp membership function for input variable θ (Fig. 3). If the value of included angle θ is less than 90, it belongs to ”Right” linguistic term; otherwise, it belongs to ”Left”. The membership functions of v and $\Delta\theta$ are shown in Fig. 4 and Fig. 5, respectively. Fig 2, Fig. 4, and Fig. 5 are proposed in the previous work [23]. We use the MATLAB software with version R2013b and its Fuzzy Logic Toolbox to construct the Mamdani-type fuzzy inference system [26]. The proposed 28 fuzzy rules are listed in Table 1 [23]. In the fourth column, from Rules 2 to 27, the linguistic terms of the variable θ are “–” which means it can be any value.

For example, the sixth fuzzy rule shown in Table 1 is as follows.

If *Left* is near, *Front* is medium, and *Right* is far, **then** $\Delta\theta$ is NS and v is medium, **(1)**

where NS denotes negative small.

4 A* Algorithm for Dynamic Path Planning

The A* algorithm is a classic deterministic path-planning method, proposed by Hart et al. [27], which relies on a best-first exploration of the motion graph to determine an optimal path from a starting node to a target node [13]. The A* algorithm uses an evaluation function $f(x, y)$, which is computed using the sum of the cost of the path from the starting node to node x ($g(x)$), the current transition from node x to node y ($c(x, y)$), and the estimated cost ($h(y)$) of the remaining path (from node y to the target node) [28].

$$f(x, y) = g(x) + c(x, y) + h(y) . \quad \textbf{(2)}$$

The A* algorithm uses an ordered list called **Open** which contains all the nodes to be explored. For each iteration, the first node x in **Open** is removed from the list and for each node y (a successor of the node x in the path), $f(x, y)$ is computed and the new y is added to **Open**. The algorithm stops if the top node is a target node. The algorithm always terminates with an optimal path from the start to the target nodes [28]. The proposed DPP algorithm is introduced in Section 4.1, and its procedure, FindNwPath, which employs the A* algorithm to determine a safe and short path for avoiding obstacles, is described in Section 4.2.

4.1 DPP Algorithm

We made four major improvements to the DPP method, which was first proposed in our previous study [23], as described in Section 1. Furthermore, we revised some major parts of the DPP algorithm, as shown in Algorithms 1, 2, and 3, to enhance its readability and replicability. The DPP algorithm has two strategies, waiting or detouring around the dynamic obstacle, for avoiding dynamic obstacles.

The main procedure of the DPP algorithm is shown in Algorithm 2, and an illustration is shown in Fig. 6. As shown in the figure, the robot detects two dynamic obstacles at time step t = 1, and the corresponding TP table (Table 2) is then generated to predict the obstacles' future trajectories (lines 5-7, Algorithm 1). In Fig. 6, the numbers inside the circles and rectangles indicate the time steps. Table 2 shows the predicted positions of the robot from time steps 1 to 9. The first position of the robot $(r_x, r_y)^{(1)}$ in Table 2 is its current position, and the robot first detects the dynamic obstacles in that position. For convenience, we set the first time step in the TP table (Table 2) as time step 1. The number of time steps, $nTraj$, is calculated according to the minimal movements that the robot can exceed in its detecting range of $(r_x, r_y)^{(1)}$, and it is the number of records in the TP table. In Table 2, $nTraj$ = 9 because the table has nine records ($Distance((r_x, r_y)^{(1)}, (r_x, r_y)^{(9)}) > 250$ cm).

Table 2. An example of trajectory prediction table

Time Step	(x, y) Coordinates			Colliding Detection	
	Robot	Obstacle 1	Obstacle 2	Collision 1 [a]	Collision 2 [a]
1	$(r_x, r_y)^{(1)}$	$(o_{1x}, o_{1y})^{(1)}$	$(o_{2x}, o_{2y})^{(1)}$	No	No
2	$(r_x, r_y)^{(2)}$	$(o_{1x}, o_{1y})^{(2)}$	$(o_{2x}, o_{2y})^{(2)}$	No	No
3	$(r_x, r_y)^{(3)}$	$(o_{1x}, o_{1y})^{(3)}$	$(o_{2x}, o_{2y})^{(3)}$	No	No
4	$(r_x, r_y)^{(4)}$	$(o_{1x}, o_{1y})^{(4)}$	$(o_{2x}, o_{2y})^{(4)}$	No	No
5	$(r_x, r_y)^{(5)}$	$(o_{1x}, o_{1y})^{(5)}$	$(o_{2x}, o_{2y})^{(5)}$	Yes	No
6	$(r_x, r_y)^{(6)}$	$(o_{1x}, o_{1y})^{(6)}$	$(o_{2x}, o_{2y})^{(6)}$	Yes	No
7	$(r_x, r_y)^{(7)}$	$(o_{1x}, o_{1y})^{(7)}$	$(o_{2x}, o_{2y})^{(7)}$	No	No
8	$(r_x, r_y)^{(8)}$	$(o_{1x}, o_{1y})^{(8)}$	$(o_{2x}, o_{2y})^{(8)}$	No	No
9	$(r_x, r_y)^{(9)}$	$(o_{1x}, o_{1y})^{(9)}$	$(o_{2x}, o_{2y})^{(9)}$	No	No

Note. [a] Collision 1(2) indicates whether a collision occurs with obstacle 1(2) at specific time step t.

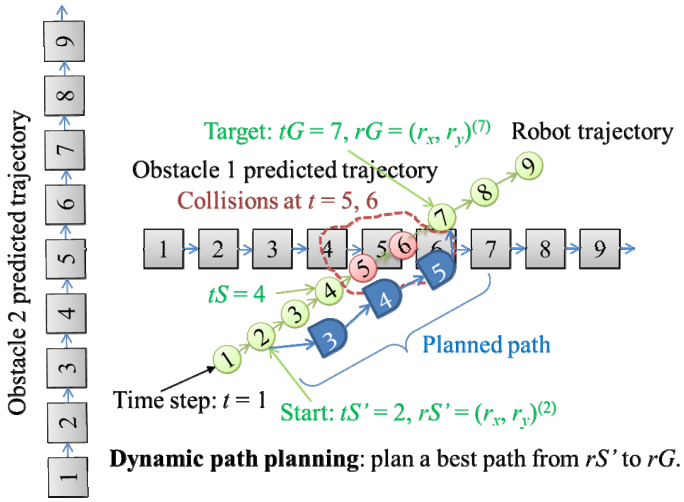


Fig. 6. An illustration of dynamic path planning

Lines 3-10 in Algorithm 2 determine the obstacle that collides with the robot and the time step that causes the collision. According to Table 2 and Fig. 6, the robot collides with obstacle 1 at time steps 5 and 6. Therefore, in line 4 of Algorithm 2, $i = 5$. In line 5, $tc = 5$, and therefore we let the first time step of path planning $tS = 5 - 1 = 4$. The final time step of collision te is 6. In line 7, tG = **iff**($tc < nTraj$, $te + 1$, tS) means

$$\textbf{if } tc < nTraj, \textbf{ then } tG = te + 1, \textbf{ else } tG = tS\ . \tag{3}$$

Algorithm 2: *Dynamic Path Planning*

Procedure *adaptedPath* = **DynamicPlanning**(*trajTable*)

1: *adaptedPath* ← null; // the initial adaptive path is set as empty
2: **for** $i = 1$ to *nTraj*, // *nTraj*: the number of time steps in the TP table
3: **for** each detected dynamic obstacle,
4: **if** the collisions begin from i **then**
5: $tc = i$ and $tS = tc - 1$; // tS: the start time for path planning
6: te ← the final time step of these collisions
7: tG = **iff**($tc < nTraj$, $te + 1$, tS);
8: *plannedPath* = **FindNewPath**(*trajTable*, tS, tG); // to Algorithm 3
9: break; // exit the inner for loop
10: **end if**
11: **end for**
12: **end for**
13: $plannedPath \leftarrow \cup_{k=1}^{tS-2-1}(r_x, r_y)^{(k)} \cup plannedPath$;
14: **for** $j = 2$ to the number of positions in *plannedPath*,
15: $adaptedPath \leftarrow adaptedPath \cup plannedPath(j)$;
16: Move the robot to *plannedPath*(j);
17: **if** the detected obstacle changes its moving way and threatens the robot **then**
18: *trajTable* ← Generate the trajectory prediction table;
19: go to line 2;
20: **end if**
21: **end for**
22: **return** *adaptedPath*; // return the planned adaptive path

According to Table 2 and Fig. 6, $tG = te + 1 = 6 + 1 = 7$ because $tc = 5 < nTraj$ and rS and rG are the start and target positions at time steps tS and tG, respectively. To obtain a wider moving space for the robot, we push the tS back by two time steps; in other words, tS' (the modified tS) $= tS - 2 = 4 - 2 = 2$ (line 9, Algorithm 3). The purpose of DPP is to determine a short and safe path from rS' to rG. In line 8, *plannedPath* = **FindNewPath**(*trajTable*, tS, tG) uses *trajTable* (TP table), tS, and tG as input parameters for the FindNewPath procedure to determine the optimal path, and stores this path in *plannedPath*. After obtaining the adaptive path, the procedure exits the inner **for loop** and proceeds to line 12. In line 13, $plannedPath \leftarrow \cup_{k=1}^{tS-2-1}(r_x, r_y)^{(k)} \cup plannedPath$ indicates a combination of the initial positions from $(r_x, r_y)^{(1)}$ to $(r_x, r_y)^{(tS-2-1)}$ with the adaptive positions of *plannedPath* determined using the FindNewPath procedure. For example, in Fig. 6, $\cup_{k=1}^{tS-2-1}(r_x, r_y)^{(k)} = (r_x, r_y)^{(1)}$ because $tS = 4$, and *plannedPath* = $\{(r_x, r_y)^{(2)}, (r'_x, r'_y)^{(3)}, (r'_x, r'_y)^{(4)}, (r'_x, r'_y)^{(5)}, (r_x, r_y)^{(7)}\}$, where $(r'_x, r'_y)^{(t)}$ denotes the (x, y) coordinate of the adaptive position of *plannedPath* at time step t. The $(r'_x, r'_y)^{(t)}$ in *plannedPath* is labeled using the flowchart symbol *delay,* as shown in Fig. 6.

Lines 12-17 denote the replanning process. If a detected dynamic obstacle, which is recorded in the TP table, changes its moving direction or speed and threatens the robot, the DPP algorithm restarts and plans another adaptive path to avoid a collision in the near future. In line 14, the index of **for loop** j starts from 2 because the robot initially stays at the first position of *plannedPath*.

4.2 A* Algorithm for Determining the Optimal Path

The most crucial part of the DPP algorithm is the FindNewPath procedure, as shown in Algorithm 3. First, the robot must select the avoiding strategy, **waiting strategy** (lines 3-8), **or detouring strategy** (lines 9-29). For the waiting strategy, if $tS = tG$ in line 3, the robot collides with a dynamic obstacle at the final time step in the TP table (tS and tG are calculated in line 7 in Algorithm 2). In line 23, $tWait$ = **iff**($tS > 3$, $tS - 3$, tS) is equivalent to

$$\textbf{if } tS > 3, \textbf{ then } tWait = tS - 3, \textbf{ else } tWait = tS . \quad (4)$$

Line 5 denotes that the robot moves from $rTraj(1)$ to $rTraj(tWait)$ and waits at $rTraj(tWait)$ for three time steps. For example, in Fig. 6, $tWait = tS - 3 = 1$ because $tS = 4 > 3$; $plannedPath = \{rTraj(1)\} \cup_{i=1}^{3} rTraj(1) = \{ rTraj(1), rTraj(1), rTraj(1), rTraj(1)\}$. In lines 6 and 7, if $waitTime < hWait$ (a tolerable threshold for waiting is $hWait = 2$ in this study) and there is no collision during waiting at $rTraj(tS)$, the robot moves from $rTraj(1)$ to $rTraj(tS)$ and waits at $rTraj(tS)$ for $waitTime$ time steps.

Lines 8-28 of Algorithm 3 determine the optimal path to avoid the dynamic obstacle. In line 10, the variables rS' and rG denote the starting and target positions, respectively. For example, in Table 2 and Fig. 6, $rS' = (r_x, r_y)^{(2)}$ and $rG = (r_x, r_y)^{(7)}$. The initial $plannedPath$ is set to $\{rTraj(1), rTraj(2), \ldots, rTraj(tS')\}$. By applying the A* algorithm, lines 13-27 (**while loop**) obtain an optimal path from rS' to rG without colliding with the obstacle according to the two objective functions f and D shown in Eq. (2). According to the A* algorithm, the robot selects the next position $rNext$ to move with minimal total cost C.

Algorithm 3: *FindNewPath Procedure*

procedure *plannedPath* = **FindNewPath**(*trajTable*, *tS*, *tG*)

1: $rTraj \leftarrow$ robot's (x, y) coordinates in *trajTable*;
2: $waitTime = tG - tS - 1$; // number of time steps the robot should wait
3: **if** $tS = tG$ **then**
4: $tWait$ = **iff**($tS > 3$, $tS - 3$, tS); // the time step for waiting
5: $plannedPath = \{rTraj(1), rTraj(2), \ldots, rTraj(tWait)\} \cup_{t=1}^{3} rTraj(tWait)$;
6: **else if** ($waitTime < hWait$) **and** (no collision during waiting at $rTraj(tS)$) **then**
7: $plannedPath = \{rTraj(1), rTraj(2), \ldots, rTraj(tS)\} \cup_{t=1}^{waitTime} rTraj(tS)$;
8: **else**
9: **if** $tS > 2$ **then** $tS' = tS - 2$; **end if**
10: $rS' = rTraj(tS')$; $rG = rTraj(tG)$;
11: $t_P = tS'$; // the current time step
12: $plannedPath = \{rTraj(1), rTraj(2), \ldots, rTraj(tS')\}$;
13: **while** $Distance(plannedPath(t_p), rG) > \varepsilon$, // apply the A* algorithm
14: $rNext$ = **GetSuccessors**($plannedPath(t_p)$, rG);
15: **for** each $rNext$,
16: **if** a collision occurs when moving to $rNext$ **then**
17: $g(rNext) = \infty$; $f(rNext) = \infty$;
18: **else**
19: $g(rNext) \leftarrow$ distance from $plannedPath(t_p)$ to $rNext$;
20: $h(rNext) \leftarrow$ distance from $rNext$ to rG;
21: $f(rNext) = g(rNext) + h(rNext)$;
22: **end if**
23: $D(rNext) \leftarrow$ the danger degree of $rNext$; // use Eq. (7) to calculate
24: $C(rNext) = f(rNext) \cdot e^{\alpha \cdot D(rNext)}$;
25: **end for**
26: $t_p = t_p + 1$; $plannedPath(t_p) \leftarrow$ the $rNext$ with minimal C;
27: **end while**
28: **end if**
29: **return** *plannedPath*;

$$C(rNext) = f(rNext) \cdot e^{\alpha \cdot D(rNext)}, \quad \textbf{(5)}$$

where $f(rNext) = g(rNext) + h(rNext)$ is the objective function for path length, from initial position (r_x, r_y)[(1)] to target position *rG,* passing through *rNext*; $\alpha \geq 0$ is a coefficient of the danger degree; and $D(rNext)$ is the objective function of the danger degree of *rNext*, and it measures how dangerous the position *rNext* is. Subsequently, $D(rNext)$ is calculated according to the distances of the surrounding obstacles of *rNext*. We suggest that α is set to smaller than 3. If $\alpha = 0$, $C(rNext) = f(rNext)$, which is the same objective function, which was presented in our previous study [23]. It only considers the path-length objective function, and $C(rNext) = f(rNext)$ is a special case ($\alpha = 0$) of Eq. (5).

Gong et al. (2011) proposed a computing method for the danger degree of a path. We used this method to calculate a position's danger degree. The threatening degree of DS_j to *rNext*, T_j, is calculated as

$$T_j = \begin{cases} 1, & d(DS_j) \leq d(DS_j)_{\min} \\ e^{-\beta \frac{d(DS_j) - d(DS_j)_{\min}}{d(DS_j)_{\max} - d(DS_j)_{\min}}}, & d(DS_j)_{\min} < d(DS_j) < d(DS_j)_{\max} \\ 0, & d(DS_j) \geq d(DS_j)_{\max} \end{cases} . \quad \textbf{(6)}$$

where DS_j (j = 1, 2, …, *NS*) is the jth danger source; in other words, the static and dynamic obstacles detected around the robot by using the laser range finder; $d(DS_j)$ represents the minimal distance between *rNext* and DS_j; and $d(DS_j)_{\min}$ and $d(DS_j)_{\max}$ are the lower and upper bounds for $d(DS_j)$, respectively. If $d(DS_j) \geq d(DS_j)_{\max}$, the position *rNext* is absolutely safe related to DS_j. Conversely, if $d(DS_j) \leq d(DS_j)_{\min}$, the position *rNext* is highly dangerous related to DS_j. The term β is a constant positive value. For all n danger sources, the danger degree of *rNext*, $D(rNext)$, is formulated as

$$D(rNext) = \max_{j=1,2,\ldots,n} \{T_j\} . \quad \textbf{(7)}$$

In this paper, $d(DS_j)_{\min}$ and $d(DS_j)_{\max}$ are set as the robot's radius and 150 (five *times* the maximal moving speed of 30 cm/time-step: 5 × 30 = 150), and β is 3 according to our prior experiments.

The initial position of the robot is *rS'*. For each *iteration* in the **while loop** (lines 13-27), the robot determines the ideal next position, *rNext*, according to the objective function given in Eq. (5), which is then stored in *plannedPath*. Repeat the procedure until the robot reaches or approaches the target position *rG*. In line 14, *rNext* = **GetSuccessors**(*plannedPath*(t_p), *rG*) indicates that the FindNewPath Procedure applies the **GetSuccessors** procedure to calculate all possible next positions, considering the robot's current position *plannedPath*(t_p) and the target position's coordinate. The **for loop** (lines 15-25) calculates the cost function $C(rNext)$ of each *rNext*. In line 26, first, the t_p value is incremented by one, and then the *rNext* with minimal $C(rNext)$ is determined and stored in *plannedPath*(t_p). Line 29 returns the produced path, *plannedPath*, to line 8 in Algorithm 2.

The **GetSuccessors** procedure is shown in Algorithm 4. Lines 3-7 obtain the ϕ_{Low}, which is the smallest steering angle of the robot among all angles and is parallel to the orientation of the threatening obstacle. Line 4 can be expressed as

$$\textbf{if } \theta < \phi + 180°, \textbf{ then } \phi_{Low} = \phi, \textbf{ else } \phi_{Low} = \phi + 180° . \quad \textbf{(8)}$$

In addition, line 6 can be expressed as

$$\textbf{if } \theta < \phi, \textbf{ then } \phi_{Low} = \phi - 180°, \textbf{ else } \phi_{Low} = \phi . \quad \textbf{(9)}$$

For example, in Fig. 6, the orientation of obstacle 1 is $\phi = 0°$.

To obtain the robot's next feasible position *rNext*, we first established the three possible velocities, v_{Slow}, v_{Medium}, and v_{Fast}, which were set to 20, 30, and 40 cm/time-step, respectively, according to the previous experiments. We then computed 19 steering angles $\{\phi_{Low}, \phi_{Low} + 10°, \phi_{Low} + 20°, \ldots, \phi_{Low} + 180°\}$ of the robot. Different speeds combined with different steering angles can obtain different next positions. There are 57 (3 × 19) combinations, and different next positions are calculated using lines 11-17 of Algorithm 4. These 57 coordinates are stored in *rNext* and returned to line 14 of Algorithm 3.

In the next section, we present the results of two simulations to demonstrate the proposed DN method.

Algorithm 4: *GetSuccessors Procedure*

procedure *rNext* = GetSuccessors(*r*, *rG*)

1: $\theta \leftarrow$ included angle between the robot's current position *r* and the target position *rG*;
2: $\phi \leftarrow$ the orientation of the threatening obstacle;
3: **if** $\phi < 180°$ **then**
4: ϕ_{Low} = **iff**($\theta < \phi + 180°$, ϕ, $\phi + 180°$);
5: **else**
6: ϕ_{Low} = **iff**($\theta < \phi$, $\phi - 180°$, ϕ);
7: **end if**
8: $\boldsymbol{v}_{suc} = \{v_{Slow}, v_{Medium}, v_{Fast}\}$; // robot's different velocities
9: $\boldsymbol{\varphi} = \{\phi_{Low}, \phi_{Low} + 10°, \phi_{Low} + 20°, \ldots, \phi_{Low} + 180°\}$; // robot's steering angle
10: *rNext* $\leftarrow$ null; $r = (r_x, r_y)$;
11: **for** *i* = 1 to 3,
12: **for** *j* = 1 to 19,
13: $r_x' = r_x + \boldsymbol{v}_{suc}(i)\cos(\boldsymbol{\varphi}(j))$; // *x* coordinate of *rNext*
14: $r_y' = r_y + \boldsymbol{v}_{suc}(i)\sin(\boldsymbol{\varphi}(j))$; // *y* coordinate of *rNext*
15: $rNext = rNext \cup (r_x', r_y')$;
16: **end for**
17: **end for**
18: **return** *rNext*;

5 Simulation Results

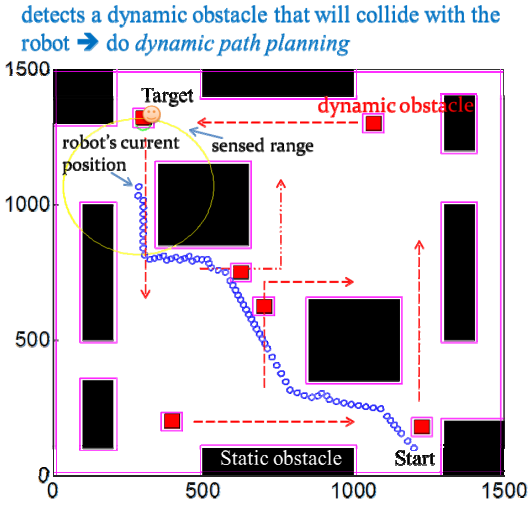
In this section, we first demonstrate our proposed DN method by presenting and analyzing the simulation results of different coefficients of danger degree α. Next, we present the comparison results of the DN method and static navigation. The proposed algorithms are programmed using MATLAB (version: R2013b), and the fuzzy controller described in Section 3 is implemented using the Fuzzy Logic Toolbox in MATLAB.

5.1 Simulation Analysis for the Proposed DN Method

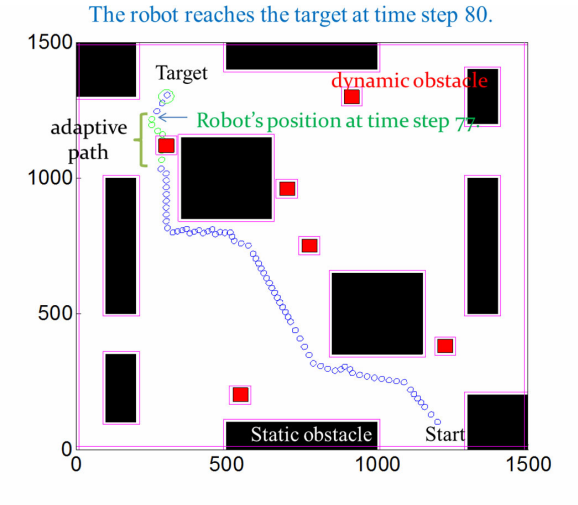
The coefficient of the danger degree α is presented in Eq. (5), which is the cost function $C(rNext)$ for determining an adaptive path to avoid the threatening obstacle, has a high influence on $C(rNext)$, such that it affects the searching result of the adaptive path. The smaller the α is, the more crucial the path-length objective $f(rNext)$ is. Conversely, the greater the α is, the greater the emphasis placed on the danger-degree objective $D(rNext)$ is. Therefore, we investigated the influences of different α values.

Fig. 7 shows the simulation results of different α (α = 0, 1, and 2). In the figure, 10 static obstacles are represented by black rectangles and six dynamic obstacles are represented by red squares. The start and target positions are (1200, 100) and (300, 1300), respectively. The environment is square shaped with an area of 1500 × 1500 cm. The dynamic obstacle moves back and forth along the dashed line. The rectangular frames outside the obstacles denote the safe range, and they are used for attracting attention. Figs. 7 (a) and (b) are the results of $\alpha = 0$ (the DPP method only considers the path-length objective). Fig. 7 (a) shows that the robot detects a dynamic obstacle at time step 70, and thus the DPP method starts determining the shortest path without collisions. The path without collision, comprising seven movements from time steps 71 to 77, is then generated using the DPP method, as shown in Fig. 7 (b). On this path, the robot first moves toward the target position and then left to avoid the dynamic obstacle. At time step 77, the position of the dynamic obstacle locates behind the robot. After time step 78, the robot moves toward the target position and reaches it at time step 80. The robot's paths, from time steps 1 to 70, shown in Figs. 7 (b)–(d) are the same, but the paths after time step 71 are different.

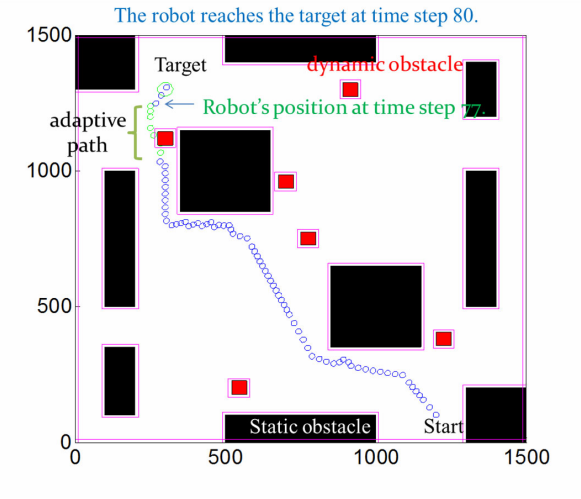
Fig. 7 (c) shows the simulation result of $\alpha = 1$ (the objective function of the DPP method is $C(rNext) = f(rNext) \cdot e^{D(rNext)}$, and it considers both path length and safety objectives). From time steps 71 to 76, the robot first moves to the upper left (stopping at (251, 1239)) to avoid the dynamic obstacle, and then



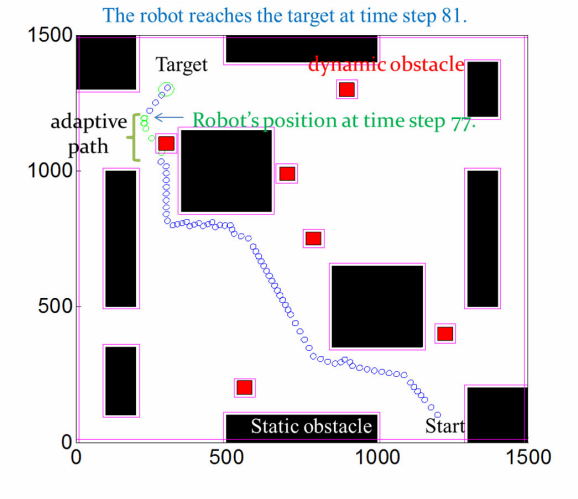
(a) $\alpha = 0$, at time step = 70



(b) $\alpha = 0$, at time step = 80



(c) $\alpha = 1$, at time step = 80



(d) $\alpha = 2$, at time step = 81

Fig. 7. Comparison results of different coefficients of the danger degree, $\alpha = 0$, 1, and 2. All subfigures have the same path from time steps 1 to 70, but the adaptive paths from time steps 71 to 77 obtained using the DPP algorithm with $\alpha = 0$, 1, and 2 are different

moves back to the previous position (251, 1219) at time step 77. The leftward arc of the adaptive path of the robot in Fig. 7 (c) is greater than that in Fig. 7 (b) because its α value is higher.

For the simulation result of $\alpha = 2$, as shown in Fig. 7 (d), the adaptive path from time steps 71 to 77 is similar to that shown in Fig. 7 (c). However, the leftward arc of the adaptive path in Fig. 7 (d) is greater than that in Fig. 7 (c) because its α value is higher. After time step 78, the robot moves toward the target position directly and reaches it at time step 81.

The results of numerical computation are summarized in Table 3. The number of movements of the three adaptive paths with $\alpha = 0$, 1, and 2 are the same (i.e., seven steps). The length of the adaptive path with $\alpha = 0$ is the shortest (because it only considers the objective of path length), that is, 130 cm, and $\alpha = 1$ results in the longest adaptive path (i.e., 170 cm). The larger the α value is, the farther the obstacle is from the adaptive path, and therefore the robot can move more safely. All three settings of the α values allow the robot to reach the target position successfully. For $\alpha = 0$ and 1, the distances between the final

position of the adaptive path and the target position are approximately equal. According to Table 3, the robot's total path length for $\alpha = 0$ is the shortest and for $\alpha = 2$ is the longest (all path lengths before the adaptive paths are identical).

In the simulation, if we do not consider the distance from the obstacle (not considering safety), $\alpha = 0$ is the ideal choice. If we consider safety (adopting a more conservative mode), $\alpha = 1$ is the ideal choice.

Table 3. Computational results of adaptive paths under different coefficients of danger degree, α = 0, 1, and 2

Performance Measure	Coefficient of Danger Degree α		
	0	1	2
No.[a] of moving steps in adaptive path	7	7	7
Length of adaptive path (cm)	**130**	170	160
Distance between last position of adaptive path and target (cm)	102.8	**102.78**	136.23
Average moving speed in adaptive path (cm/time-step)	18.571	**24.286**	22.857

Note. [a] No. is an abbreviation of number.

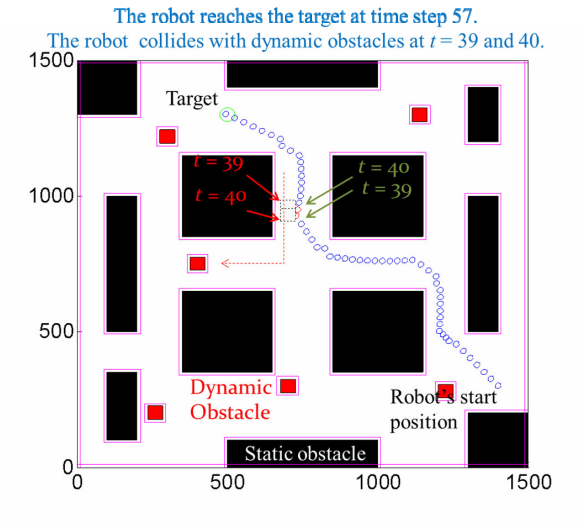
5.2 Comparison Results

For avoiding dynamic obstacles, some studies have used static navigation methods, which regard the dynamic obstacles as static obstacles. However, we proposed the DPP algorithm, which predicts the future moving trajectories of dynamic obstacles for obtaining a more effective strategy for avoiding moving obstacles. Several studies have adopted static navigation [16, 22, 25]. According to our simulation results, static navigation is not suitable for avoiding dynamic obstacles.

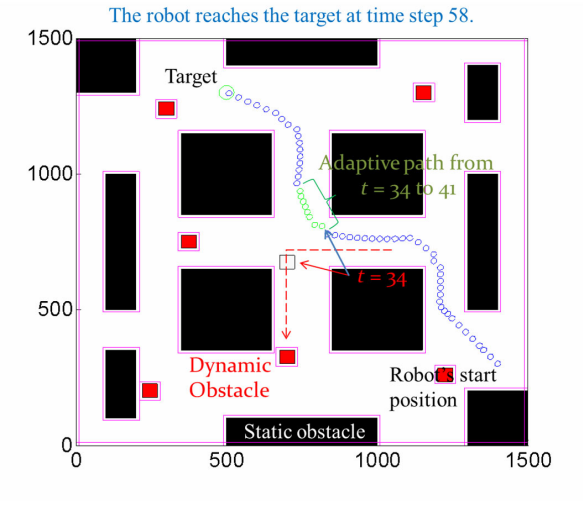
To construct static navigation for comparison, we used only the fuzzy controller described in Section 3 to steer the robot. The robot has sonar sensors with a maximum sensing range of only 130 cm to detect the static and dynamic obstacles, as shown in Fig. 1. If the robot does not detect any obstacle, it directly moves toward the target position. Otherwise, the fuzzy controller is activated to control the robot's steering angle and moving speed to guide the robot's movement. The fuzzy rules to infer the control outputs are shown in Table 1. We introduce two comparison results as follows:

Case 1: Moving from (1400, 300) to (500, 1300). The setting of the simulation environment is similar to that described in Section 5.1; however, the environment in this section adds two more static obstacles, as shown in Fig. 8. The starting and target positions are (140, 300) and (500, 1300), respectively. The simulation and numerical computation results are shown in Fig. 8 and Table 4, respectively. According to Fig. 8, static navigation results in two collisions at time steps 39 and 40; however, the other three DN methods with different settings for coefficients of the danger degree can successfully guide the robot to reach the target position without collision. To facilitate the explanation of the simulation results, we used the terms $DN_{\alpha=0}$, $DN_{\alpha=0.5}$, $DN_{\alpha=1}$, and $DN_{\alpha=2}$ to express the DN methods whose coefficients of the danger degree are set to $\alpha = 0$, 0.5, 1, and 2, respectively. In the simulation process, we first let the robot move and then let the dynamic obstacles move. Therefore, in static navigation, the robot does not know the position of a dynamic obstacle at the next time step. In addition, the robot controlled using static navigation cannot predict the future path of a dynamic obstacle. Therefore, the robot has a higher likelihood of colliding with dynamic obstacles when it is close to them.

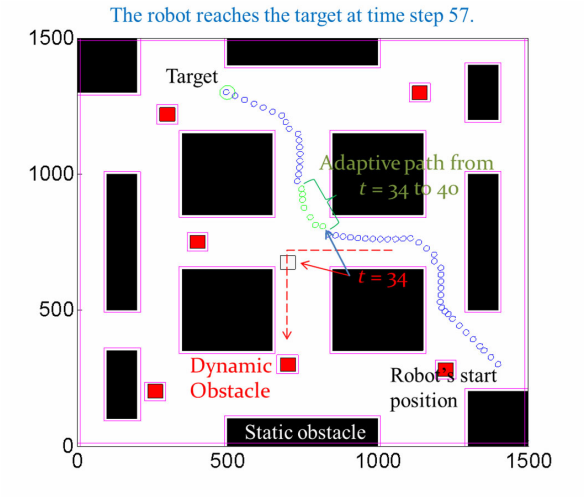
Figs. 8 (c) and (d) show the simulation results of the DN method with different settings of α. Because the time step is 34 ($t = 34$), the robot detects a dynamic obstacle at its lower-left corner. The DPP method begins searching an adaptive path based on the objective function shown in Eq. (2). The paths of the three robots are similar, as shown in Figs. 8 (c) and (d). According to Table 4, the $DN_{\alpha=0}$ method obtains the shortest path (1545.8 cm), and static navigation results in the longest path (1576.9 cm). Therefore, the setting of $\alpha = 0$ can result in the shortest adaptive path (from time steps 34 to 41). Except for time steps 34 to 41, the rest of the robot's paths in the different methods, as shown in Fig. 8, are highly similar. In addition, the length of the adaptive path generated using the $DN_{\alpha=0}$ method is the shortest (158.6 cm), and the $DN_{\alpha=2}$ method has the longest adaptive path (178.6 cm).



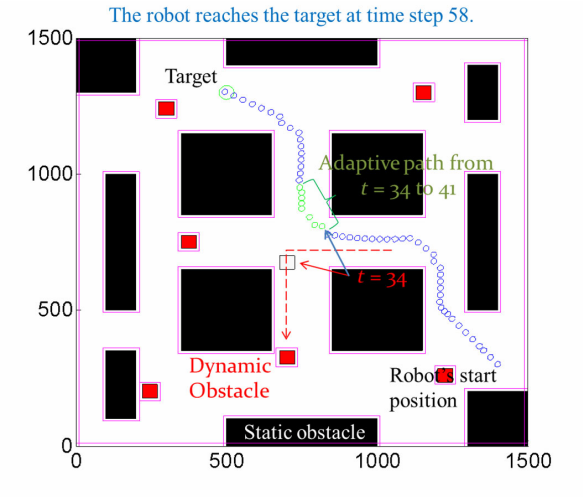
(a) Static navigation method, at $t = 57$



(b) $DN_{\alpha=0}$ method, at $t = 58$



(c) $DN_{\alpha=1}$ method, at $t = 57$



(d) $DN_{\alpha=2}$ method, at $t = 58$

Fig. 8. Comparison Results I: Comparison of static navigation and DN method with different settings of α. The symbol "t" indicates the time step. Static navigation cannot successfully guide the robot to reach the target position; however, the proposed DN method can guide the robot without collision

Table 4. Comparison results I for static navigation method and DN method with different settings of α

Performance Measure	Static Navigation Method	DN Method with Different Settings of α [a]		
		0	1	2
No. [b] of time steps to reach target	**57**	58	**57**	58
Path Length (cm)	1576.9	**1545.8**	1560.4	1569.2
Average moving speed (cm/time-step)	27.7	26.7	27.4	27.1
No. of collisions	2	**0**	**0**	**0**
No. of executions for dynamic path planning	–	1	1	1
Total no. of moving steps for all adaptive paths	–	8	**7**	8
Total length of all adaptive paths (cm)	–	**158.6**	168.6	178.6

Note. [a] α is the coefficient of danger degree. [b] No. is an abbreviation of number.

According to the preceding description, the $DN_{\alpha=0}$ method exhibits superior performance, and the performance of static navigation is inferior.

Case 2: Moving from (750, 200) to (1200, 1300). The simulation environment of Case 2 is the same as that in Case 1, but the starting and target positions are set to (750, 200) and (1200, 1300), respectively. According to Fig. 9, the robot's path obtained using static navigation includes three collisions; however, the robot's paths obtained using the DN approach with different settings of α have no collisions. According to Fig. 9 (a), the robot collides with the oncoming dynamic obstacle at time steps 64, 65, and 66 because static navigation does not predict the movement of dynamic obstacles. In Case 2, the robot's paths using these four methods are slightly different, where the $DN_{\alpha=0}$ method obtains the shortest and smoothest path.

At time step 14, the robot detects a threatening dynamic obstacle. Thus, the DPP algorithm within the DN method begins determining the optimal adaptive path. The adaptive paths of the three different settings of α values (i.e., α = 0, 0.5, and 1) of DN methods have nine movements, from time steps 14 to 23; however, the three adaptive paths are not the same. The $DN_{\alpha=0}$ method steers the robot moving behind the threatening obstacle. The $DN_{\alpha=0.5}$ and $DN_{\alpha=1}$ methods, as shown in Figs. 9 (c) *and* (d), respectively, steer the robot moving in front of the threatening obstacle, and thus the robot has to move farther and spend more time to detour around the obstacle.

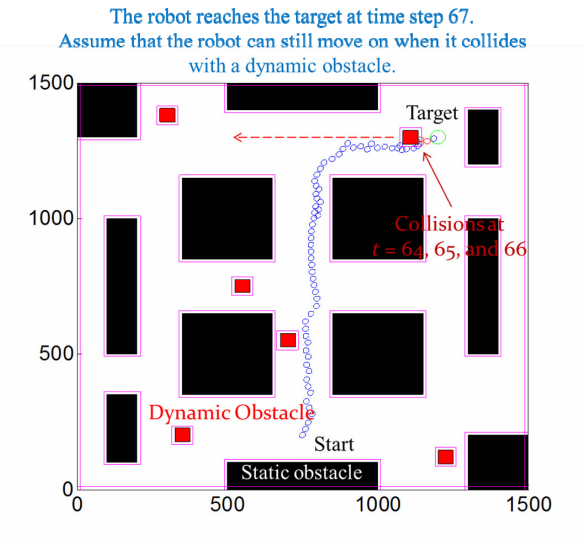
After the robot moves according to the *first* adaptive path, it detects an extremely close threatening dynamic obstacle at time step 23 (Figs. 9 (c) and (d)). Therefore, the $DN_{\alpha=0.5}$ and $DN_{\alpha=1}$ methods perform the DPP algorithm for a second time to avoid the oncoming obstacle. Compared with the $DN_{\alpha=1}$ method, the $DN_{\alpha=0.5}$ method can obtain a shorter adaptive path, from time steps 24 to 26. After performing the DPP algorithm for a second time, the robot controlled using $DN_{\alpha=0.5}$ and $DN_{\alpha=1}$ methods continues to move toward the target position, and eventually reaches the target position successfully.

At time step 14, the robot detects a threatening dynamic obstacle. Thus, the DPP algorithm within the DN method begins determining the optimal adaptive path. The adaptive paths of the three different settings of α values (i.e., α = 0, 0.5, and 1) of DN methods have nine movements, from time steps 14 to 23; however, the three adaptive paths are not the same. The $DN_{\alpha=0}$ method steers the robot moving behind the threatening obstacle. The $DN_{\alpha=0.5}$ and $DN_{\alpha=1}$ methods, as shown in Figs. 9 (c) and (d), respectively, steer the robot moving in front of the threatening obstacle, and thus the robot has to move farther and spend more time to detour around the obstacle.

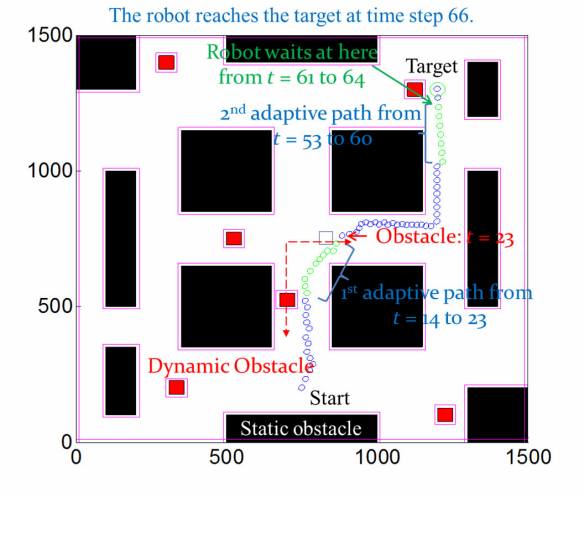
After the robot moves according to the first adaptive path, it detects an extremely close threatening dynamic obstacle at time step 23 (Figs. 9 (c) and (d)). Therefore, the $DN_{\alpha=0.5}$ and $DN_{\alpha=1}$ methods perform the DPP algorithm for a second time to avoid the oncoming obstacle. Compared with the $DN_{\alpha=1}$ method, the $DN_{\alpha=0.5}$ method can obtain a shorter adaptive path, from time steps 24 to 26. After performing the DPP algorithm for a second time, the robot controlled using $DN_{\alpha=0.5}$ and $DN_{\alpha=1}$ methods continues to move toward the target position, and eventually reaches the target position successfully.

For the $DN_{\alpha=0}$ method, the robot detects a threatening dynamic obstacle near the target position at time step 53, and therefore, it uses the DPP algorithm for a second time. There are six movements in the second adaptive path, and the robot stops at (1204.05, 1235.17) from time steps 59 to 60 to wait for the dynamic obstacle. At time step 60, the dynamic obstacle is extremely close to the robot, and the robot waits from time steps 60 to 61. Similarly, at time step 62, the robot waits for the obstacle from time steps 62 to 63. After performing the DPP algorithm for a second time, the robot performs two successive DPP algorithms at time steps 60 and 62. At time step 64, because the dynamic obstacle has moved away from the target position, the robot moves toward the target position by using the fuzzy controller.

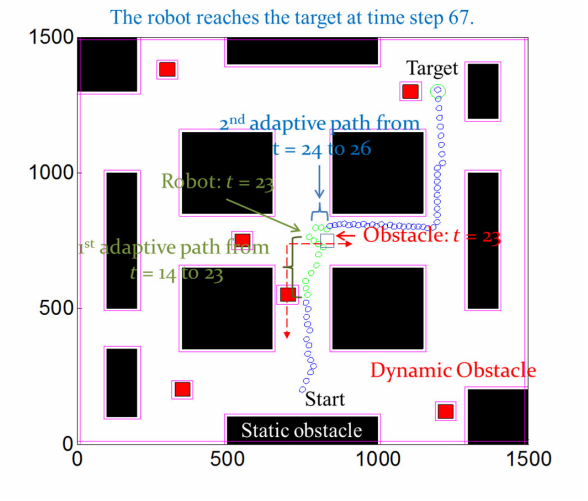
The numerical computation results are summarized in Table 5. The $DN_{\alpha=0}$ method spends the least time, 66 time steps, to steer the robot to reach the target position without any collision, and the total path length is the shortest at 1481.8 cm. Conversely, the $DN_{\alpha=1}$ method spends the most time, 72 time steps, to steer the robot to reach the target position without any collision, and the entire path length is the longest at 1689.6 cm. For the adaptive behavior of the DPP algorithm, the proposed DN methods, for example, $DN_{\alpha=0}$, $DN_{\alpha=0.5}$, and $DN_{\alpha=1}$ in this simulation case, can successfully steer the robot to avoid the dynamic obstacles. The $DN_{\alpha=0}$ method is used to determine the shortest adaptive path. However, the total length of the adaptive paths of the $DN_{\alpha=0}$ method (432.4 cm) is not the shortest, because it contains an adaptive path that is relatively long from time steps 53 to 60. The $DN_{\alpha=0}$ method performs the DPP algorithm four times. For the $DN_{\alpha=0.5}$ method, the total number of movements for all adaptive paths is minimal (i.e., 12



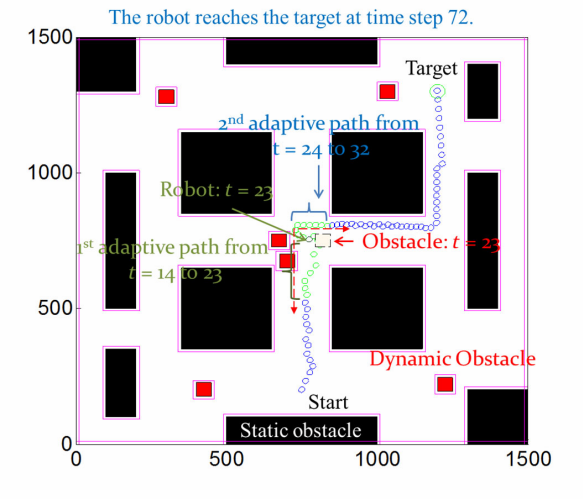
(a) Static navigation method, at time step = 67



(b) $DN_{\alpha=0}$ method, at $t = 66$



(c) $DN_{\alpha=0.5}$ method, at $t = 67$



(d) $DN_{\alpha=1}$ method, at $t = 67$

Fig. 9. Comparison Results II: Comparison of static navigation method and dynamic navigation method (DN method) with different settings of α. The symbol "t" indicates the time step. The proposed DN method can successfully drive the robot to reach the target position without collision.

Table 5. Comparison results II for static navigation method and DN method with different settings of α

Performance Measure	Static Navigation Method	DN Method with Different Settings of α		
		0	0.5	1
No.[a] of time steps to reach target	67	**66**	67	72
Path Length (cm)	1602.9	**1481.8**	1594.1	1689.6
Average moving speed (cm/time-step)	23.9	22.5	23.8	23.5
No. of collisions	3	**0**	**0**	**0**
No. of executions of dynamic path planning	–	4	**2**	**2**
Total no. of moving steps for all adaptive paths	–	15	**12**	17
Total no. time steps for waiting dynamic obstacle		5	0	1
Total length of all adaptive paths (cm)	–	432.4	**290.0**	440.0

Note. [a] No. is an abbreviation of number.

movements), and the total length of all adaptive paths is the shortest at 290 cm. In addition, the robot controlled using the $DN_{\alpha=0.5}$ method does not wait for the dynamic obstacle. Conversely, the total number of movements for the two adaptive paths of the $DN_{\alpha=1}$ method is the highest at 17 movements, and the total length of the two adaptive paths is the longest at 440 cm.

According to the preceding description, the $DN_{\alpha=0}$ method exhibits the best performance because it allows the robot to move to the target position by using the shortest path without any collision.

However, in some cases, the waiting strategy of the DPP algorithm fails. For example, in the $DN_{\alpha=0}$ method, the starting and target positions of the robot are set to (750, 280) and (1200, 1300), respectively. When the time step is 45, the robot waits endlessly.

6 Conclusions

This paper presented a navigation method in the dynamic environment called the DN method with static and dynamic obstacles. The proposed DN method comprises static navigation and DPP. When the robot detects a dynamic obstacle that will collide with the robot in the near future, the DPP algorithm starts to determine the optimal adaptive path to avoid the obstacle according to the two objective functions of the path length and distances between the adaptive path and the surrounding obstacles. For the DPP algorithm, two strategies are used to avoid dynamic obstacles: one is to detour around the obstacle, and the other is to wait until the robot can move safely. If the robot only detects the static obstacles around it, the fuzzy controller guides the robot's movement to avoid these obstacles.

This study is an extension of our previous study [23]; we made four major improvements to the DPP algorithm. In particular, we added another objective function, the danger degree of each movement in the adaptive path, and improved the waiting strategy for considering more situations. Moreover, the movements of dynamic obstacles are modified so that they are changeable. If a dynamic obstacle in the sensing range changes its moving way, the DPP algorithm restarts to determine the optimal adaptable path. Moreover, we revised the DPP algorithm by using mathematics that is more precise. For the two simulation cases, we compared static navigation (which treats dynamic obstacles as static obstacles) with the proposed DN method by using different settings of α. Static navigation fails to avoid the dynamic obstacles in both cases. The proposed $DN_{\alpha=0}$ method exhibits the best performance because it can obtain the shortest path from the start to the target positions without collision.

Acknowledgments

The author would like to thank Chiehyi Ding for his valuable suggestions to this paper, e.g. adding the additional safety objective function for dynamic path planning. The author also wants to thank Wallace Academic Editing for editing the English of this paper. This work was supported by Hsuan Chuang University in Taiwan, Republic of China, with Grant Number HCU-102-B1-07.

References

[1] P. Raja, S. Pugazhenthi, Path planning for a mobile robot in dynamic environments, International Journal of the Physical Sciences 6(20)(2011) 4721-4731.

[2] Z. Wu, L. Feng, Obstacle prediction-based dynamic path planning for a mobile robot, International Journal of Advancements in Computing Technology 4(3)(2012) 118-124.

[3] D.-W. Gong, J.-H. Zhang, Y. Zhang, Multi-objective particle swarm optimization for robot path planning in environment with danger sources, Journal of Computers 6(8)(2011) 1554-1561.

[4] A. Kaplan, P. Uhing, N. Kingry, R.D. Adam, Integrated path planning and power management for solar-powered unmanned ground vehicles, in: Proc. 2015 IEEE International Conference on Robotics and Automation (ICRA), 2015.

[5] M. Yarmohamadi, H.H.S. Javadi, H. Erfani, Improvement of robot path planning using particle swarm optimization in

dynamic environments with mobile obstacles and target, Advanced Studies in Biology 3(1)(2011) 43-53.

[6] N. Buniyamin, N. Sariff, W.A.J. Wan Ngah, Z. Mohamad, Robot global path planning overview and a variation of ant colony system algorithm, International Journal of Mathematics and Computers in Simulation 5(1)(2011) 9-16.

[7] X. Deng, L. Zhang, L. Luo, An improved ant colony optimization applied in robot path planning problem, Journal of Computers 8(3)(2013) 585-593.

[8] F. Ahmed, K. Deb, Multi-objective optimal path planning using elitist non-dominated sorting genetic algorithms, Soft Computing 17(7)(2013) 1283-1299, 2013.

[9] A. Tuncer, M. Yildirim, Chromosome coding methods in genetic algorithm for path planning of mobile robots, in: E. Gelenbe, R. Lent, G. Sakellari, A. Sacan, H. Toroslu, A. Yazici (Eds.), Computer and Information Sciences II, Springer, London, UK, 2012, pp. 377-383.

[10] M. Roozegar, M.J. Mahjoob, M. Jahromi, DP-based path planning of a spherical mobile robot in an environment with obstacles, Journal of The Franklin Institute 351(10)(2014) 4923-4938.

[11] J.H. Zhou, H.Y. Lin, A self-localization and path planning technique for mobile robot navigation, in: Proc. 2011 9th World Congress on Intelligent Control and Automation, 2011.

[12] G. Li, Y. Tamura, A. Yamashita, H. Asama, An effective improved artificial potential field-based regression search method for autonomous mobile robot path planning, International Journal of Mechatronics and Automation 3(3)(2013) 141-170.

[13] S.M. Persson, I. Sharf, Sampling-based A* algorithm for robot path-planning, The International Journal of Robotics Research 33(13)(2014) 1683-1708.

[14] L. Sun, Y. Luo, X. Ding, L. Wu, Path planning and obstacle avoidance for mobile robots in a dynamic environment, The Open Automation and Control Systems Journal 6(2014) 77-83.

[15] P. Benavidez, M. Jamshidi, Mobile robot navigation and target tracking system, in: Proc. 2011 6th International Conference on System of Systems Engineering, 2011.

[16] M. Faisal, K, Al-Mutib, R. Hedjar, H. Mathkour, M. Alsulaiman, E. Mattar, Multi modules fuzzy logic for mobile robots navigation and obstacle avoidance in unknown indoor dynamic environment, in: Proc. 2013 International Conference on Systems, Control and Informatics, 2013.

[17] R. Kala, A. Shukla, R. Tiwari, Dynamic environment robot path planning using hierarchical evolutionary algorithms, Cybernetics and Systems: An International Journal 41(6)(2010) 435-454.

[18] R. Kümmerle, M. Ruhnke, B. Steder, C. Stachniss, W. Burgard, Autonomous robot navigation in highly populated pedestrian zones, Journal of Field Robotics 32(4)(2015) 565-589.

[19] Y. Lu, X. Huo, O. Arslan, P. Tsiotras, Incremental multi-scale search algorithm for dynamic path planning with low worst-case complexity, IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics 41(6)(2011) 1556-1570.

[20] M. Phillips, M. Likhachev, SIPP: Safe interval path planning for dynamic environments, in: Proc. 2011 IEEE International Conference on Robotics and Automation (ICRA), 2011.

[21] N.T. Thanh, N.V. Afzulpurkar, Dynamic path planning for a mobile robot using image processing, Journal of Computer Science and Cybernetics 24(4)(2008) 358-373.

[22] M. Faisal, R. Hedjar, M. Al Sulaiman, K. Al-Mutib, Fuzzy logic navigation and obstacle avoidance by a mobile robot in an unknown dynamic environment, International Journal of Advanced Robotic Systems 10(37)(2013) 1-7.

[23] C.-H. Chiang, C. Ding, Robot navigation in dynamic environments using fuzzy logic and trajectory prediction table, in: Proc. 2014 International Conference on Fuzzy Theory and its Applications, 2014.

[24] M.K. Singh, D.R. Parhi, S. Bhowmik, S.K. Kashyap, Intelligent controller for mobile robot: Fuzzy logic approach, in Proc. 12th International Conference of International Association for Computer Methods and Advances in Geomechanics, 2008.

[25] M. Wang, J.N.K. Liu, Fuzzy logic-based real-time robot navigation in unknown environment with dead ends, Robotics and Autonomous Systems 56(2008) 625-643.

[26] S.N. Sivanandam, S. Sumathi, S.N. Deepa, Introduction to Fuzzy Logic Using MATLAB, Springer, Berlin, Germany, 2007.

[27] P.E. Hart, N.J. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, IEEE Transactions on Systems Science and Cybernetics 4(2)(1968) 100-107.

[28] P. Nocera, G. Linares, D. Massonié, L. Lefort, Phoneme lattice based A* search algorithm for speech recognition, in: P. Sojka, I. Kopeček, K. Pala (Eds.), TSD 2002, Lecture Notes in Artificial Intelligence 2448, Springer, London, UK, 2002, pp. 301-308.