Robotics and Autonomous Systems 115 (2019) 143–161

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

Solving the optimal path planning of a mobile robot using improved Q-learning

Ee Soong Low, Pauline Ong *, Kah Chun Cheah

Faculty of Mechanical and Manufacturing Engineering, Universiti Tun Hussein Onn Malaysia (UTHM), 86400 Parit Raja, Batu Pahat, Johor, Malaysia

HIGHLIGHTS

- We propose an improved Q-learning to solve path planning of a mobile robot.
- The flower pollination algorithm is used to initialize the Q-table prior to the implementation of Q-learning.
- Its effectiveness is tested in solving the optimal path in different test cases.
- Performance comparison with classical Q-learning and other modified Q-learning is made.
- The proposed model shows improvement in terms of computational time than others.

ARTICLE INFO

Article history:
Received 14 October 2018
Received in revised form 4 February 2019
Accepted 18 February 2019
Available online 25 February 2019

Keywords:
Flower pollination algorithm
Obstacle avoidance
Path planning
Robot
Q-learning
Robot navigation

ABSTRACT

Q-learning, a type of reinforcement learning, has gained increasing popularity in autonomous mobile robot path planning recently, due to its self-learning ability without requiring a priori model of the environment. Yet, despite such advantage, Q-learning exhibits slow convergence to the optimal solution. In order to address this limitation, the concept of partially guided Q-learning is introduced wherein, the flower pollination algorithm (FPA) is utilized to improve the initialization of Q-learning. Experimental evaluation of the proposed improved Q-learning under the challenging environment with a different layout of obstacles shows that the convergence of Q-learning can be accelerated when Q-values are initialized appropriately using the FPA. Additionally, the effectiveness of the proposed algorithm is validated in a real-world experiment using a three-wheeled mobile robot.

1. Introduction

Autonomous mobile robots are getting more and more interest, due to it can make a decision by its own when executing the actions in a given environment. This behaviour has popularized its utilization in unsupervised tasks, where the robot will learn by its own and execute a series of actions to achieve the predefined goal. To accomplish the given mission, for example, terrestrial exploration, path planning is important for mobile robots such that they know how to move from one place to another, and how to perform the desired task. The mobile robots are not only needed to complete the mission; they must also be able to deal with all kinds of unexpected disruptions that bring adverse consequences performance. Hence, path planning is crucial for mobile robot navigation.

Path planning concerns with how the robot decides to move in an environment to its predefined goal position without hitting any existing obstacles. This involves the computation of a collision-free path between the initial position and the goal position. Path planning is varied according to different environments which the robots have to deal with, such as a structured/ fully known environment, partially known environment or fully unstructured/ fully unknown environment. Among the types of environment, the partially known environment is said to be the most practical where some areas within the environment have already known before the navigation. The path planning can be classified into static path planning and dynamic path planning, depending on the nature of the obstacles and goals. In static path planning, the position and orientation of the obstacles and goals are unchanged with time, while in dynamic path planning, the obstacles and goals are free to move in the environment [1].

Path planning, alternatively, can be classified into local path planning and global path planning. For the local path planning, the data collected from the local sensor during navigation are used to generate a new path in responding to the change of environment. On another hand, global path planning with its algorithm can produce a complete path from the initial start point to the goal point, if only static obstacles exist before the mobile robot starts its exploration. In the navigation of mobile robots,

* Corresponding author.
E-mail address: ongp@uthm.edu.my (P. Ong).

https://doi.org/10.1016/j.robot.2019.02.013
0921-8890/

both localization and path planning are considered. Localization is a process in which the actual position in the real world is matched into a map. This method of navigation is limited in its application as it cannot deal with uncertainty and dynamic environment. On contrary, path planning is a process of searching for the shortest yet collision-free path from the initial position of the robot to the goal position.

The solution to path planning has seen numerous methods, as mentioned in [2–13]. The reinforcement learning was originally used in the disciplines of game theory, information theory, control theory and operation research. Through time, it has been adopted in the navigation of autonomous mobile robot [14–16]. Reinforcement learning can be used in navigating the unknown environment based on a set of rewards and penalties. The mobile robot which is known as the agent receives a reward for collision-free action and receive a penalty when it collides with the obstacles [17].

Q-learning — a type of reinforcement learning, is a famous learning technique related to the principle of reward and penalties, and also the interaction of the robot with the environment. An agent, which is the mobile robot, performs an action in an environment and receives an immediate reward or penalty for the action taken. The Q-value is updated continuously based on the received reward or penalty, and the states with the highest Q-value are considered as the optimal path for the mobile robot. The primary advantage of using Q-learning in autonomous mobile robot enables the autonomous mobile robot to deal with the unstructured environment through self-learning and navigate to the goal position in a collision-free path.

Although Q-learning has shown a successful implementation in obstacle avoidance of mobile robot, it has limitation too. When the size of the learning environment increases, not only a longer computational time to update the matrices of Q-value and larger adaptive memory matrices are required, but there is also a possibility of not involving the right probabilities in the converged memory matrices. In this regard, Arin and Rabadi integrated metaheuristic algorithm for randomized priority search with Q-learning, in order to minimize the search time for the optimal solution [18]. Subsequently, 0–1 multidimensional knapsack problem was used to validate the proposed algorithm. Wang et al. proposed a backward Q-learning, combining the Sarsa algorithm and Q-learning [19]. The advantage of using Sarsa algorithm is that it takes a shorter time in convergence, as compared to Q learning. However, Sarsa algorithm tends to be trapped in the local minimum, and this can be compensated by the hybridization with the Q-learning. The backward Q-learning was applied in cliff walk, mountain car and cart–pole balancing control system, where improvement in both learning time and finishing performance were observed. Duguleana and Mogan proposed the solution for a mobile robot in avoiding both static and dynamic obstacles with the combination of Q-learning and neural network planner [20]. This solution was beneficial for application with time as a constraint, due to the ability to set the desired speed prior to the trajectory computation. Validation in both simulation and real-world demonstrated a satisfying rate of conversion. Das et al. proposed an alternative in path trajectory optimization, involving multi-robots in chaos environment [21]. The proposed solution was a combination of four fundamental principles in classical Q-learning, particle swarm optimization (PSO), and differentially perturbed velocity algorithm for faster convergence. The improvement in convergence rate was due to all mobile robots took shorter path length and arrival time to their own destinations. The reduced in turning angle which in turn decreased the energy usage of the robots, was another significant improvement obtained by the proposed solution. The proposed path trajectory optimization has been validated in both simulation and real-world using Khepera-II robot. Carlucho et al. proposed the incremental Q-learning for adaptive PID control [22]. A temporal memory was applied in the learning process for better efficiency. This study has brought a solution to the unknown operation conditions due to the characteristic of Q-learning which improved PID tuning significantly. The states and actions in the learning spaces were defined based on the behaviour of the system, and the proposed algorithm was validated using a mobile robot with different conditions for its real-world operation. Experimental results showed the successful performance of this proposed solution. Rakshit et al. combined the Q-learning as part of local refinement and differential evolution for global search in adaptive memetic algorithm [23]. Experimental simulation in the multi-robot navigation in the Khepera II environment showed that the proposed method outperformed the genetic algorithm (GA), PSO and differential evolution, in terms of runtime, cost function evaluation and accuracy. The similar work has been conducted in [24], with the artificial bee colony used for the global search.

On the other hand, Q-learning can be improved through appropriate initialization of Q-values. Several methods had been used in this regard. Oh et al. showed that the hybridization of the fuzzy rules to initialize the Q values was able to reduce the learning complexity of the classical Q-learning [25]. Wiewiora used potential-based shaping function to initialize Q-values while having classical Q-formula to update the Q-table [26]. Koenig et al. showed that enforcing the conditions of consistency and admissibility on initialization of Q-values reduced the complexity of Q-learning [27]. Song et al. successfully improved the speed of convergence and stability of Q-learning through applying dynamic wave expansion neural network into initialization of Q-values [28]. Charypar et al. proved that high Q-values during initialization will improve the exploration ability of Q-learning while low Q-values will lead to finding feasible solution faster but slower convergence for the optimal solution [29]. Simsek et al. introduced a new cost function for initialization of Q-table in order to overcome the slow convergence rate of Q-learning [30]. Yan and Xiang initialized the Q-table using the inverse Euclidean distance between the current position and the target position, in order to accelerate the learning efficiency of Q-learning [31].

Metaheuristic algorithm, too, has been widely studied for its impact on Q-learning. Das et al. integrated improved particle swarm optimization with differentially perturbed velocity (IPSO-DV) into an improved Q-learning for convergence acceleration [21]. Such integration expedites the learning process of Q-learning in solving multi-robots path planning problem, in which total computational time, turning angle and path length, were reduced. Comparison with PSO and classical Q-learning corroborated the superior performance of IPSO-DV. Rakshit et al. proposed an adaptive memetic algorithm through combining differential evolution (DE) algorithm and Q-learning, called DE-TDQL, and thereafter applied it into multi-robots path planning. In the proposed DE-TDQL, DE and Q-learning were used for exploration search and exploitation search, respectively [23]. The simulation showed that the DE-TDQL outperformed PSO and GA in terms of the total number of steps needed, the average path traversed (APT) and average total path deviation (ATPD). Rakshit et al. extended their work by replacing the DE with artificial bee colony (ABC) in their proposed algorithm, called ABC-TDQL [32]. The substitution of DE with ABC gave better performance in mobile robot path planning, in terms of convergence time, accuracy, ATPD and average uncovered target distance. Wang et al. developed the sequential Q-learning through the combination of Q-learning and GA [33] and applied it to solve multi-robot coordination during the object transportation. The proposed sequential Q-learning demonstrated shorter computational time than single-agent Q-learning, as well as increasing adaptivity and flexibility

when the algorithm was applied in an unknown environment. Sadhu et al. integrated the Q-learning into the framework of the firefly algorithm (FA), where the former was used to optimize the light absorption coefficient of FA [34]. Solving the path planning of a robotic manipulator with the proposed method showed the improvement in terms of solution quality and computational complexity, in comparison with other reference algorithms.

As observed in previous studies, Q-learning suffers from slow convergence due to the calculation of all possible action-state, but it can be alleviated through appropriate initialization of Q-values using different approaches [19,21,28,30]. Despite this, there is limited literature that explores this question using metaheuristic algorithm. Motivated by the promising results reported in [19,21,28,30], flower pollination algorithm (FPA) — another interesting facet of metaheuristic algorithms which draws inspiration from the flower pollination process, is integrated into the Q-learning framework from the aspect of initialization of Q-values. The FPA is favoured in this work due to its simplicity and efficiency. The preliminary study in [35] has shown its promising performance in function optimization as opposed to GA and PSO, however, with the competitive advantage of employing fewer control parameters. Hence, an upsurge in the utilization of FPA in diverse fields soon after its inception, including scheduling problem [36], visual shape matching [37], machining process [38], path planning [39] and data mining [40], to name a few, have been seen. An improved Q-learning with flower pollination algorithm (IQ-FPA) is proposed in this study wherein, the FPA is utilized to initialize the Q-table prior to the implementation of Q-learning. Through the assignation of initial Q-values, the Q-learning can start the exploration from a better foundation instead of performing action randomly from the initial state without any guidance, which may end up stuck in a pit before reaching the goal. Hence, the main contribution of this study is the formulation of FPA into the initialization of Q-values in Q-learning, and the introduction of the formulated IQ-FPA into the mobile robot path planning to address the slow convergence of Q-learning. The computational time, total travelled distance, the smoothness of path and collision-free navigation path are the main concern during the development of the IQ-FPA into the path planning of a mobile robot. In short, the aim of this study is to generate optimal collision-free path with the shortest distance, focusing on minimizing the total learning time and maximizing the smoothness of the path.

The paper is organized as follows. Section 2 presents an overview of the Q-learning, while in Section 3, a brief introduction of the FPA, is given. The working principles and steps involved in the IQ-FPA are presented in Section 4. The simulation results and performance comparison in terms of computational time, travelled distance and the path smoothness in four case studies are discussed in Section 5. In Section 6, validation of the IQ-FPA in real-world experimental is presented. Lastly, conclusions are drawn in Section 7.

2. Q-learning algorithm and its limitation

Q-learning is a type of reinforcement learning (RL) algorithms developed by Watkins in 1989 [41]. Q-learning applies the concept of reward and penalty in exploring the unstructured environment. The terms used in the Q-learning algorithm involve state, action, agent, reward and penalty, as shown in Fig. 1. The agent in this figure represents the mobile robot. A state is the position where the agent defines its location in the environment, whereas an action is the movement that the agent takes to move from one state to another state. For a reward, it is the positive value given to increase the Q-value for the correct action taken by the agent at the particular state, whereas a penalty is the negative value given to decrease the Q-value for the wrong action taken by the agent.

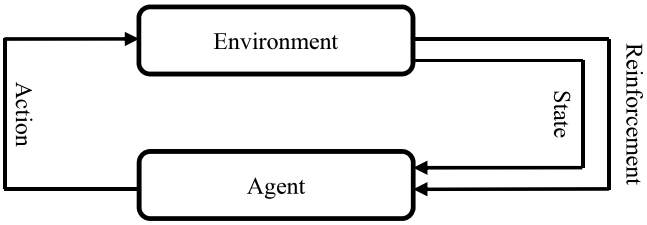


Fig. 1. Interaction between the Agent and the Environment.

The experience is obtained through exploration and exploitation by the RL agent. In general, exploration is used at the beginning and exploitation is used at the end of the learning process. This is because exploration involves the selection of random actions, while the agent learns without considering the current state in order to visit all the state–action pair in the environment. On the other hand, exploitation involves the knowledge from the agent to be applied in choosing actions to maximize the reward of the current state [42]. In Q-learning, it is not necessary to have the complete environment modelling such that the transition probabilities matrix of all the states and actions to be known initially. This algorithm has contributed to being a vital development in RL. The policy being applied is not affected by the values of the optimal Q to be converged. The Q-values of the Q-learning algorithm is updated using the expression of:

$$Q(s_t, a_t) \leftarrow (1-\alpha)Q(s,a) + \alpha\left[r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})\right] \quad (1)$$

where
s_t : Current state
a_t : Action performed in s_t state
r_{t+1} : Received reinforcement signal after s_t executed
s_{t+1}: Next state
γ : Discount factor $(0 \leq \gamma < 1)$
α : Learning coefficient $(0 < \alpha < 1)$

The next action can be determined based on the Q-values of the next possible state with the following expression:

$$a^*(s) = \max_a Q(s_{t+1}, a) \quad (2)$$

The pseudo-code of the classical Q-learning algorithm is summarized as in Algorithm 1.

Even though the Q-learning is able to learn and improve the solution from time to time, it encounters several Achilles' heels. Consider a searching environment with m states and n number of possible actions, the dimension of the constructed Q-table will be $m \times n$. When moving from the current state to the next state, the agent has to select the action with the highest Q value among the n possible actions. This implies that $(n-1)$ times of comparison is required. To update the Q-table with n states, the number of comparisons needed is $m(n-1)$. Hence, when the size and complexity of environment increase, particularly in the real-world situation, the time took for the Q-learning to complete the path planning increases exponentially as the search space increases [43].

Furthermore, during the initial stage of exploration, the motion of agent is completely random, resulting in wastage of computational effort, slower convergence rate and time-consuming. As shown in Eq. (2), the action selected for the next state will be determined by the highest Q-value. The agent has no choice but to perform random selection during the early stage of learning since all Q-values are initialized to zeros. The random movement with exploring probability continues to a certain extent, up until the Q-values are refined. In order to counter this limitation,

Algorithm 1: Classical Q-learning algorithm

Begin

Initiate all Q-values, $Q(s,a)$ in Q-table to zero

Select a starting state, $Q(s_1,a_1)$

while (iteration < Max iteration)

  while goal is not achieved

    Select an action, a within the available actions in the current state according to the highest Q-value in the next state

    Perform the selected action, a and reward or penalty, r will be given

    Update the Q-value using Equation (1)

    Move the state to new state, s'

  end while

end while

end

the improvement on Q-learning by specifying initial Q values is suggested in this work.

In the proposed IQ-FPA, prior to the learning of Q-learning, the FPA is applied to initialize some of the Q-values. As a result, the agent is no longer starting from zero information of its environment, but utilizing the prior knowledge to its exploration process, thus improving the computational complexity and convergence rate.

3. Flower pollination algorithm

Pollination in flower occurs when the pollen grain of one flower is transferred to others, which can be categorized into biotic and abiotic pollination. Around 90% of the flower species undergo the pollination by biotic pollination (also known as cross-pollination), involving pollinator such as the bee, whereas other 10% of flower species undergo abiotic pollination (also known as self-pollination), which does not require a pollinator [44]. Inspired by the biological pollination process, the FPA — a nature-inspired optimization algorithm, has been put forward by Yang in [35]. The derivation of the FPA has been idealized with four simplified rules as follows:

i. Global pollination process involves both biotic and cross-pollination. The pollinator follows Lèvy flights distribution when carrying pollen.
ii. Local pollination process involves both abiotic and self-pollination.
iii. Reproduction probability is viewed as flower constancy, which is related to the similarity between the two flowers involved.
iv. Switching probability, $p \in [0, 1]$ is used to control the occurrence of local and global pollination.

The first and third simplified rules in the FPA, involving the global pollination and flower constancy, is expressed mathematically as:

$$x_i^{t+1} = x_i^t + \gamma L(\lambda)(x_i^t - g^*) \tag{3}$$

where
x_i^t : Pollen i/ vector x_i at t iteration
g^* : Current found best solution among all solutions at current iteration
γ : Scaling factor (for step size control)
$L(\lambda)$: Pollination strength

A more effective way to imitate the characteristic for pollinators which move with different distance steps in a long distance is by applying the Lèvy flights. The Lèvy distribution, L > 0, is expressed by:

$$L \sim \frac{\lambda \Gamma(\lambda) \sin(\pi\lambda/2)}{\pi} \frac{1}{s^{1+\lambda}} \tag{4}$$

where Γ: Gamma function

The local search process in simplified Rule 2 and Rule 3 can be expressed as:

$$x_i^{t+1} = x_i^t + \varepsilon(x_j^t - x_k^t) \tag{5}$$

where
x_j^t and x_k^t : Pollen from same flower species but separate flower.
ε : A random number in uniform distribution of [0, 1]
x_i^t : Pollen i/ vector at t iteration

The pseudo-code of FPA is summarized as in Algorithm 2.

4. Improved Q-learning with flower pollination algorithm

In order to improve the limitation of slow convergence in the classical Q-learning, the IQ-FPA is proposed in this work, which can be divided into two phases: initialization of Q-values using FPA and exploration and exploitation process by the Q-learning.

4.1. Initialization of Q-values using flower pollination algorithm

4.1.1. Configuration of C-space

C-space simplifies the real world situation such that the location of the robot is represented by a point and obstacles are represented by darken regions, which means forbidden areas. Consider an environment of $u \times w$ grids (the navigation map),

Algorithm 2: Flower pollination algorithm

Begin

Define objective function f(x),x=$(x_1,x_2,x_3,\ldots,x_d)^T$

Do initialization of a population of n flowers in random positions.

Obtain the best solution g^* in the initial population

Define the range of switch probability p ∈ [0,1]

while (t < Max Generation)

for i = 1:n (all n flowers in the population)

if (rand > p)

Draw a (d-dimensional) state vector L from Levy distribution

Obtain global population using the relation

$$x_i^{t+1} = x_i^t + \gamma L(\lambda)(x_i^t - g^*)$$

else

Draw ∈ from a uniform distribution in [0,1]

Obtain local pollination using $x_i^{t+1} = x_i^t + \in (x_j^t - x_k^t)$

end if

Evaluate new solutions

If new solutions are better than update population

end for

Find the current best solution g^*

end while

end

where each grid represents a state in the IQ-FPA. Location of each grid is labelled through the Cartesian plane where the *x*-axis represents the horizontal position, the *y*-axis represents the vertical position, and coordinate is presented in (*x*, *y*) format. A matrix MAP is created to indicate whether the state (the position navigation map) is free space, goal or obstacle. In this regard, the dimension of the matrix MAP is corresponding to the size of the navigation map. Since in this study, all the simulation results used in the case studies are 20 × 20 grids and thus, the dimension of the matrix MAP is 20 × 20. The first and second dimension of MAP represent *x*-coordinate and *y*-coordinate of a grid in the map, respectively. The values of matrix MAP correspond to whether the particular position in the grid is free space, or occupied by a target or an obstacle. For example, when the position (x_1, y_1) is occupied by an obstacle, its value in MAP is assigned as -1, *i.e.*, $MAP\,(x_1, y_1) = -1$. On the other hand, it takes the value of 2, 1 and 0 respectively, if the position is a free space, or occupied by a robot and a target position. As shown in Eq. (2), the mobile robot selects the next action based on the highest Q-values among all next possible states. By penalizing the position of an obstacle with a negative value, it is guaranteed that the mobile robot will always stay away from the obstacles.

4.1.2. Generation of the initial population

An initial population consists of *q* flowers is generated randomly, where each flower in the population represents a state in the navigation map and is converted into the position in (x, y) coordinates, as shown in Eq. (6).

$$Flower = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_q & y_q \end{bmatrix} \tag{6}$$

The *x*- and *y*-coordinates of each flower are generated within the range ([1, width(map)],[1, height(map)]). The bound acts as the constraint to prevent the newly generated flower falls outside the navigation map. Additionally, only the feasible solution, *i.e.*, the generated flower does not fall in an obstacle area, can be accepted as the initial population. If such conditions are violated, a new flower will be generated to replace the infeasible solution. The pseudo-code of how to initialize the population in FPA is summarized as in Algorithm 3.

Algorithm 3: Initialization of population in flower pollination algorithm

```
Begin
Define the population size, q=50
Define the lower bound, L=1
Define the upper bound, U=20
Create a matrix, Flower with dimension of q x 2, Flower(q,2)
while (i <= q)
        Flower(i,1) = random within lower and upper bound
        Flower(i,2) = random within lower and upper bound
        while location of (Flower(i,1),Flower(i,2)) has an obstacle
                Flower(i,1) = random within lower and upper bound
                Flower(i,2) = random within lower and upper bound
        end while
end while
end
```

4.1.3. Local search and global search using flower pollination algorithm to update Q-values

The proposed IQ-FPA intends to initialize the Q-values using FPA prior to the implementation of Q-learning. Therefore, following from the population initialization, the fitness of each flower, representing the Q-value, is evaluated using Eq. (1), and the best free space position, g_{best} with the highest Q-value, is determined. Subsequently, the population will undergo the global and local pollination process by utilizing Eqs. (3) and (5) in order to search for the feasible path.

From the current position of each flower, the selection of next position relies on Eq. (1). The fitness value of each flower in a new position, which corresponds to the Q-value, is assigned to a negative value as a penalty when an intersection with an obstacle occurs. Hence, this position will be ignored and a new position will be generated. Besides, generation of the new position of each flower will be confined within the range ([1, width(map)],[1, height(map)]), such that it does not fall outside the navigation map. The fitness values of the entire population are evaluated again. When a better free space is found, the free space will be the current g_{best}. The process of pollination is repeated for maximum iteration (Max_iter) of 1000 in this study. Through these repetitive steps, eventually, the highest Q-value along the path can be found within a shorter computational time. Since the main concept in IQ-FPA is to generate Q-values by FPA such that the FPA can explore the states using flower pollination process, therefore, the fitness function (evaluation of Q-value) for FPA is the expression for calculating the Q-values as stated in Eq. (1), where the Q- values are to be maximized. Hence, the Q-values in the Q-table are updated continuously with the maximum Q-values. Thus, an updated Q-table with different Q-values is constructed, as compared to classical Q-table with Q-value of zero for all states.

4.2. Exploration and exploitation process by the Q-learning

After some Q-values in the Q-table have been initialized by FPA, the Q-learning will take place subsequently. The exploration and exploitation processes at this stage are similar to the classical Q-learning, except that the IQ-FPA starts with an updated Q-table with different Q-values. The agent will start from the starting point and continue its search by selecting the next state using Eq. (2) by referring to the updated Q-table. Since all the next possible states will have different Q-values, the selection of next state will not be completely random as compared to classical Q-learning. After proceeded to the next state, Q-value of the previous state will be updated based on Eq. (1). The searching process continues until the target position is found. It is worth mentioning that since the FPA is used to initialize the Q-values prior to the implementation of Q-learning, additional computational complexity of O($qMax\_iter$) has been introduced by the proposed IQ-FPA.

The flow chart of the IQ-FPA is shown in Fig. 2.

5. Results and performance analysis of IQ-FPA

In this section, the proposed IQ-FPA is tested and compared with the classical Q-learning and Improved Decentralized Q-learning algorithm (IDQ) through four case studies, by considering a navigation map with 20 × 20 grid environment and with the predefined obstacles in different shapes, sizes and layout. The possible actions for the mobile robot to consider are forward, backward, left, and right. The Q-value of each action associated with a particular position is stored in the Q-table with dimension 400 × 4. For all case studies, the mobile robot has the same starting and destination points at (2, 16) and (19, 11), respectively. The simulation is repeated for 30 runs and subsequently, Q-learning and IDQ algorithms are compared with IQ-FPA in terms of average and standard deviation for computational time, total travelled distance and smoothness of path. For all algorithms, the total computational time upon completion of all predefined maximum iteration number was taken for each run. For example, as shown in Fig. 2, the maximum number of iteration been assigned to IQ-FPA for Q-values initialization phase by FPA and learning phase using Q-learning are 1000 and 100, respectively. Hence, the time

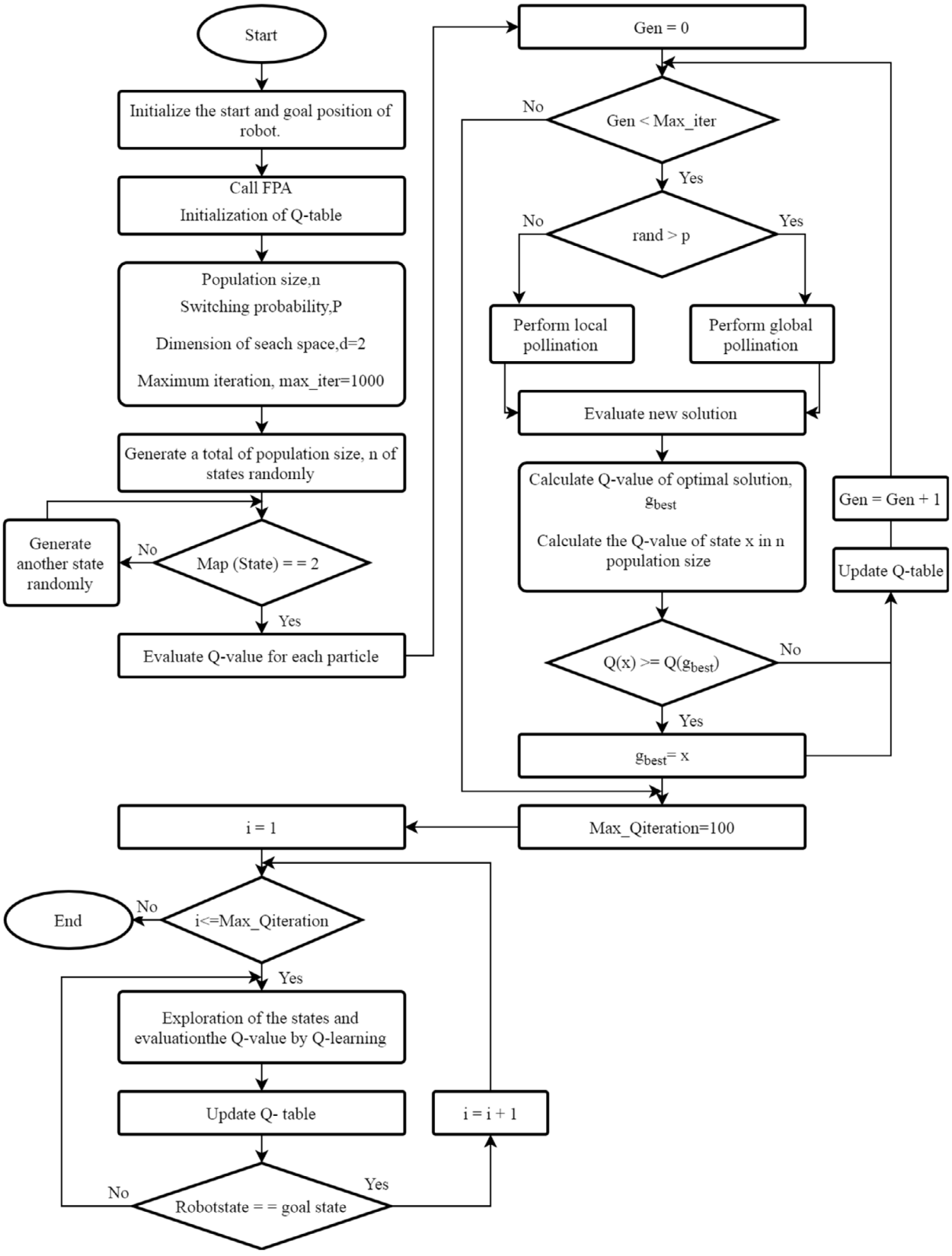


Fig. 2. Flow chart of IQ-FPA.

taken in completing both phases for each run is measured, and the average for all 30 runs is calculated subsequently. For the Q-learning and IDQ, the maximum number of iteration for each run is assigned as 100. The total travelled distance is expressed as:

$$\textit{total travelled distance} = \sum \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}, \quad (7)$$

which is the summation of the travelled distance from the initial state to the goal state, where (x_i, y_i) and (x_{i+1}, y_{i+1}) represent the current state and the next state of the mobile robot, respectively. The path smoothness, on the other hand, is evaluated by summing the robot's turning angle in the obtained path, which is given as:

$$\theta = \sum \left[\text{atan2}\left(\frac{y_{i+1} - y_i}{x_{i+1} - x_i}\right) - \text{atan2}\left(\frac{y_i - y_{i-1}}{x_i - x_{i-1}}\right) \right] \tag{8}$$

Intuitively, the lesser the turning angle is, the smoother the obtained path is, since the mobile robot does not change its direction frequently. Hence, the path smoothness and the total of the robot's turning angle are reciprocal to each other.

For Q-learning, the learning coefficient, α in Eq. (1) determines the range of the latest received data that will override the previous data. When $\alpha = 0$, the agent will learn nothing, whereas the agent will only read the latest received information when α=1. According to Watkins [41], when the value of α is relatively small, Q-value in the function Q(s, a) will eventually converge to optimal Q-value, Q^* after the agent has travelled all the states and considered all the possible actions in the given environment. As such, the learning rate, α is assigned to 0.2 in this study, in accordance with the work of Khriji et al. [45]. Discount factor, γ is another parameter to be considered in the Q-learning to determine the type of reward that the agent receives. When $\gamma = 0$, the agent will only consider immediate reward, whereas the agent will consider future reward when γ approaches 1. François-Lavet et al. summarized that a high discount factor, γ in the range of $0 \leq \gamma < 1$ and low value of learning rate, α in the range of $0 < \alpha < 1$ are an effective combination of Q-learning [46]. The use of a high value of discount factor enables more exploration and prevents the Q-learning from getting trapped in the local optimum, but may lead to the instability of the agent in the learning process. This problem can be improved by using a small learning rate, α. The discount factor, γ of 0.8 is assigned in this study in accordance with the work in [45].

On the other hand, for the IQ-FPA, the effect of varying switch probability, p, and population size, n, on the navigation capability of IQ-FPA is explored. The switch probability, p is varied from $p = 0.5$ to $p = 0.9$ with an increment of 0.1, and the population size, n is varied from $n = 10$ to $n = 50$ with an increment of 5. Fig. 3 shows the required computational time in order for the mobile robot to reach the destination point from the start point in test case 1 with 8 obstacles, by using the IQ-FPA with different combinations of switch probability and population size. It can be observed that in general, an increase in the population size will lead to an increment in computational time. This is intuitive as a larger population size needs longer searching time. As such, the population size of 10 is sufficient to solve the path planning problem in this work, since the robot is able to reach the destination point within a shorter time with low population size. In addition, it can be observed that the IQ-FPA with the switch probabilities of $p = 0.5$ and $p = 0.8$ gives the best performance in general.

The similar trend of variation in terms of computational time with different combinations of switch probability and population size is observed for other case studies and hence, the figures are omitted here for brevity. The optimal values of the switch probability, p, and the population size, n, for all case studies after empirical attempts of combinations are summarized in Table 1. The simulation is performed using MATLAB R2014a software in AMD 10, Quad core, and 2.5 GHz personal computer with Windows 10 platform. To validate the statistical significance of the obtained results, T-test is applied in this work. The null hypothesis is rejected at the confidence interval of 5%.

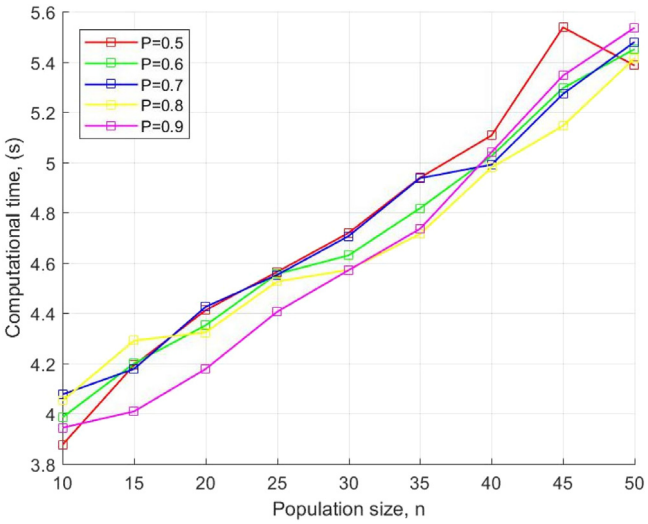


Fig. 3. Variation in terms of computational time with different switch probabilities and population sizes used in the IQ-FPA in test case 1 with 8 obstacles.

Table 1
Optimal values of switch probability and population size used in the IQ-FPA for all case studies.

Test Case		Switch Probability, p	Population Size, n
1	8 obstacles	0.5	10
	9 obstacles	0.5	10
	10 obstacles	0.5	10
	11 obstacles	0.6	10
	12 obstacles	0.9	10
2		0.8	10
3		0.8	10
4		0.5	10

Additionally, to evaluate the effectiveness of initializing the Q-values using FPA, a similar framework, namely, the improved decentralized Q-learning algorithm (IDQ) proposed by Simsek et al. [30] has been simulated for performance comparison. In IDQ, instead of initializing the Q-values using the metaheuristic algorithm, a newly developed reward function was applied to initialize the values of Q-table. This is to overcome the slow convergence behaviour of Q-learning, which serves the same purpose and integrates a similar approach as in the IQ-FPA. For these reasons, IDQ is chosen for comparison in this work.

However, it is pertinent to note that in [30], the IDQ is used for interference reduction and thus, a few modifications is required to accommodate IDQ for path planning. On this basis, the states in IDQ are representing the position of the mobile robot, while the actions in IDQ have been changed to forward, backward, left, and right, as in the case with IQ-FPA. The new cost function (reward function) of IDQ is modified and determined by the difference between the position of the next action state and position of the target, given as:

$$difference_i = \left| x_{a_i} - x_{t\,\arg et} \right| + \left| y_{a_i} - y_{t\,\arg et} \right| \tag{9}$$

where $difference_i$ represents the difference for each action a_i, x_{a_i} represents the x-position of action a_i, $x_{t\,\arg et}$ represents the x-position of the target, y_{a_i} represents the y-position of action a_i and $y_{t\,\arg et}$ represents the y-position of action a_i.

The cost function of each action is assigned as:

$$\text{cost} = \begin{cases} difference_{lowest} = -0.2 \\ difference_{second\ lowest} = -0.4 \\ difference_{third\ lowest} = -0.6 \\ difference_{highest} = -0.8 \end{cases} \tag{10}$$

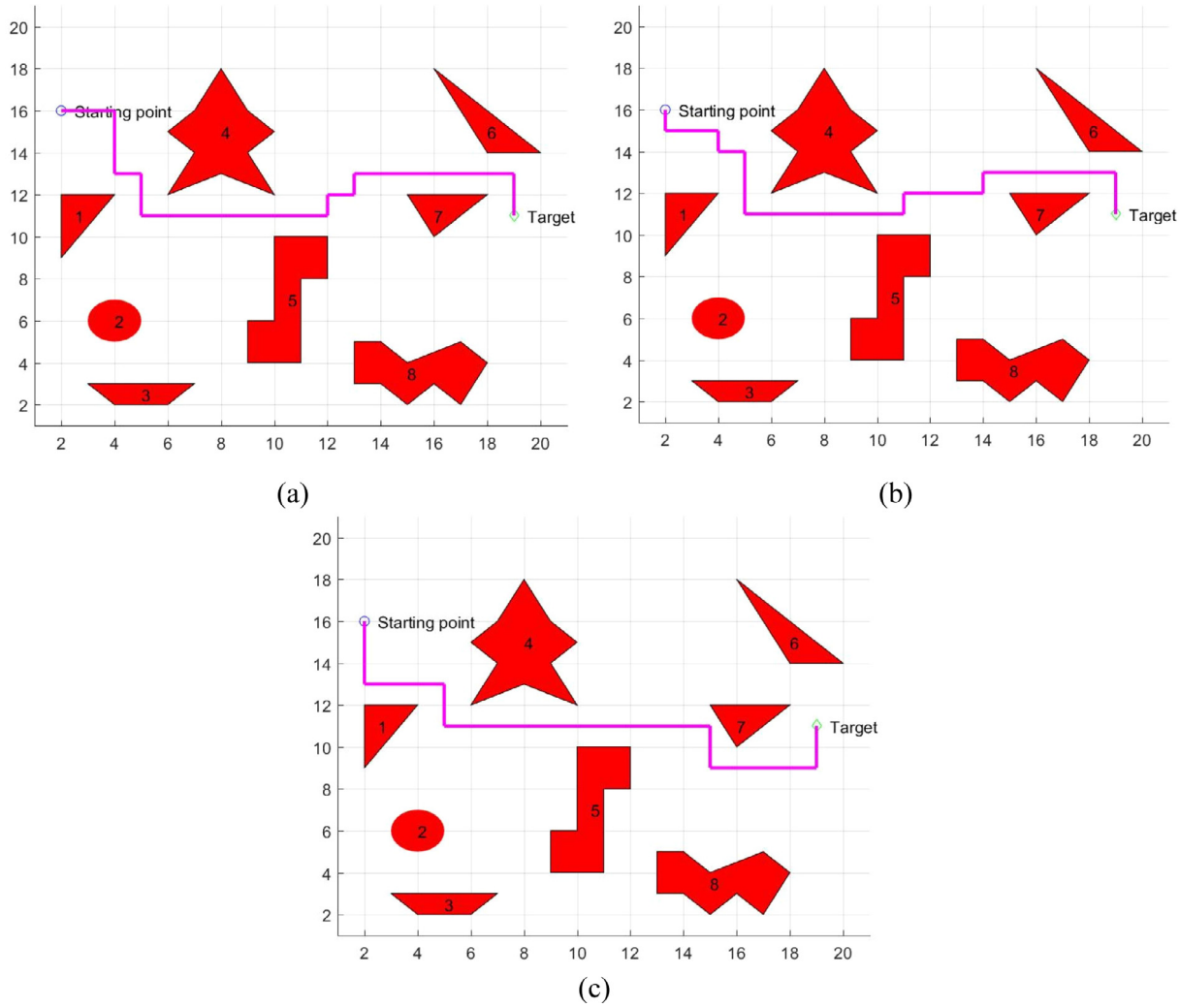


Fig. 4. Test Case 1 with 8 obstacles — the obtained optimal path by (a) Q-learning; (b) IQ-FPA; and (c) IDQ.

The action with the lowest difference between the position of the next action state and position of the target will be assigned a cost function equals to −0.2 and so on. The learning rate, discount factor and greedy value are assigned to 0.2, 0.9 and 0.01, respectively, according to Simsek et al. [30]. However, for test case 3 and test case 4, the greedy value has been changed to 0.3, which is due to the simulation using the greedy value of 0.01 in IDQ is getting stuck in local optima. Readers can refer to [30] for the details of IDQ.

5.1. Test case 1

The map used in test case 1 is a self-designed map, which intends to study the relationship between the path planning performance of the proposed IQ-FPA with a different number of obstacles in a cluttered environment. The number of obstacles is varied from 8 to 12.

Figs. 4–8 present the navigation map in test case 1 with 8, 9, 10, 11 and 12 obstacles, respectively, and the obtained optimal path using the Q-learning, the proposed IQ-FPA and IDQ, while Tables 2 and 3 compare the performance of IQ-FPA with Q-learning and IDQ, respectively, in terms of average computation time taken to reach the destination point from the starting point, average travelled distance and smoothness of path. Intuitively, longer computation time is required with the increasing number of obstacles. However, as shown in Table 2, Q-learning takes the shortest computation time to complete the path planning for the navigation map with the highest number of obstacles. This may due to the decrease in free spaces with an increasing number of obstacles. Hence, the mobile robot has less free spaces to explore and in turn, less possible actions to be considered. Consequently, the stopping condition can be achieved in a shorter period of time, giving a shorter computational time with the higher number of obstacles.

Interestingly, the opposite trend is observed for IQ-FPA and IDQ wherein, an increase in the number of obstacles will lead to longer computational time. As described in Section 4, the position of the newly generated flower shall not fall in an obstacle area during the initialization of Q-values, or else generation of new solution will be repeated to replace the infeasible solution. Hence, for IQ-FPA, the chance to regenerate the possible path solution is getting higher with the existence of more obstacles. This can be observed through a significant drop in performance in the test cases with 11 and 12 obstacles, where the IQ-FPA requires 7.82% and 9.34% extra computational time than the Q-learning in solving the path planning. As for the IDQ, more local optima are created when the number of obstacles increases and hence, a longer time is taken in completing the path.

From Table 2, in comparison with the classical Q-learning, using the IQ-FPA improves the total computation time significantly in test case 1 with 8 and 9 obstacles. And, most encouragingly, the improvement of about 13.30% is achieved, for the navigation

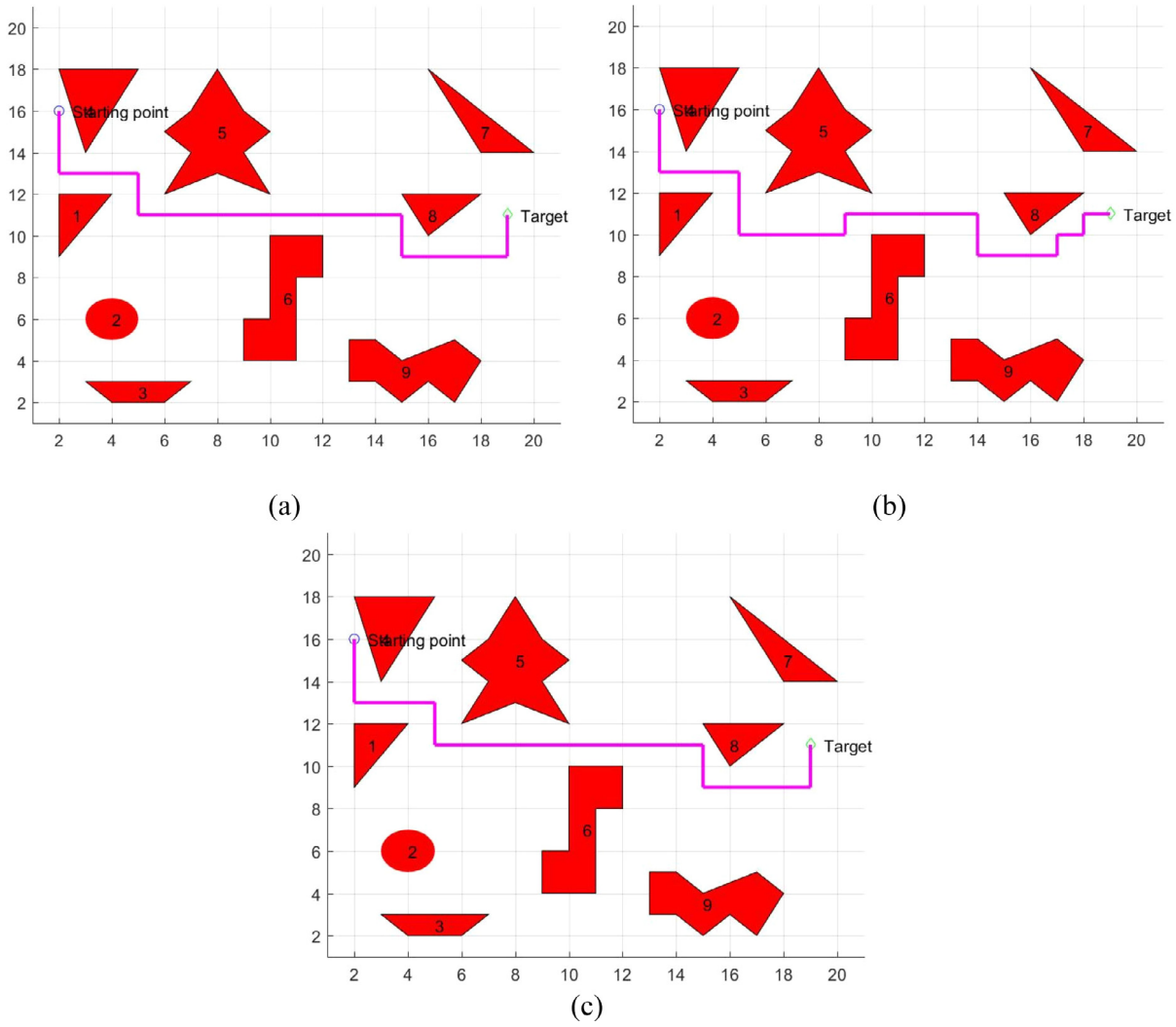


Fig. 5. Test Case 1 with 9 obstacles — the obtained optimal path by (a) Q-learning; (b) IQ-FPA; and (c) IDQ.

Table 2
Performance Comparison between Q-Learning and IQ-FPA.

Test Case	Average Computational Time (s)				Average Travelled Distance (Unit)				Average Path Smoothness (Rad)			
	Q-Learning	IQ-FPA	Percentage of Improvement (%)	Statistical Significant?	Q-Learning	IQ-FPA	Percentage of Improvement (%)	Statistical Significant?	Q-Learning	IQ-FPA	Percentage of Improvement (%)	Statistical Significant?
1 (8 obstacles)	4.06	3.52	13.30	Yes	28.93	29.33	−1.38	No	22.62	22.99	−1.64	No
1 (9 obstacles)	4.04	3.62	10.40	Yes	30.67	31.27	−1.96	No	20.94	22.72	−8.50	No
1 (10 obstacles)	3.89	3.88	0.26	No	30.00	29.60	1.33	No	21.10	20.73	1.75	No
1 (11 obstacles)	3.64	3.98	−9.34	Yes	27.67	28.73	−3.83	No	16.28	18.48	−13.51	No
1 (12 obstacles)	3.71	4.00	−7.82	Yes	27.67	29.00	−4.81	No	17.49	19.32	−10.46	No
2	3.23	3.66	−13.31	No	30.07	26.80	10.87	No	26.13	19.84	24.07	No
3	3.65	3.63	0.55	No	26.33	26.60	−1.03	No	19.32	18.48	4.35	No
4	3.00	2.70	10.00	Yes	24.00	23.47	2.21	No	15.24	13.72	9.97	No

map with 8 obstacles. In terms of the average distance travelled and smoothness of path, although there seem slight differences in Figs. 4–8, the results in Table 2 show that the Q-learning produces shorter and smoother path than the IQ-FPA in general, with 1.38–4.81% improvement for average travelled distance, and 1.64–13.51% improvement for average path smoothness, but such improvements are statistically insignificant. By comparing the standard deviations of the obtained results by Q-learning and IQ-FPA as presented in Table 4, it can be seen that here too, the performance of IQ-FPA is far more consistent than Q-learning

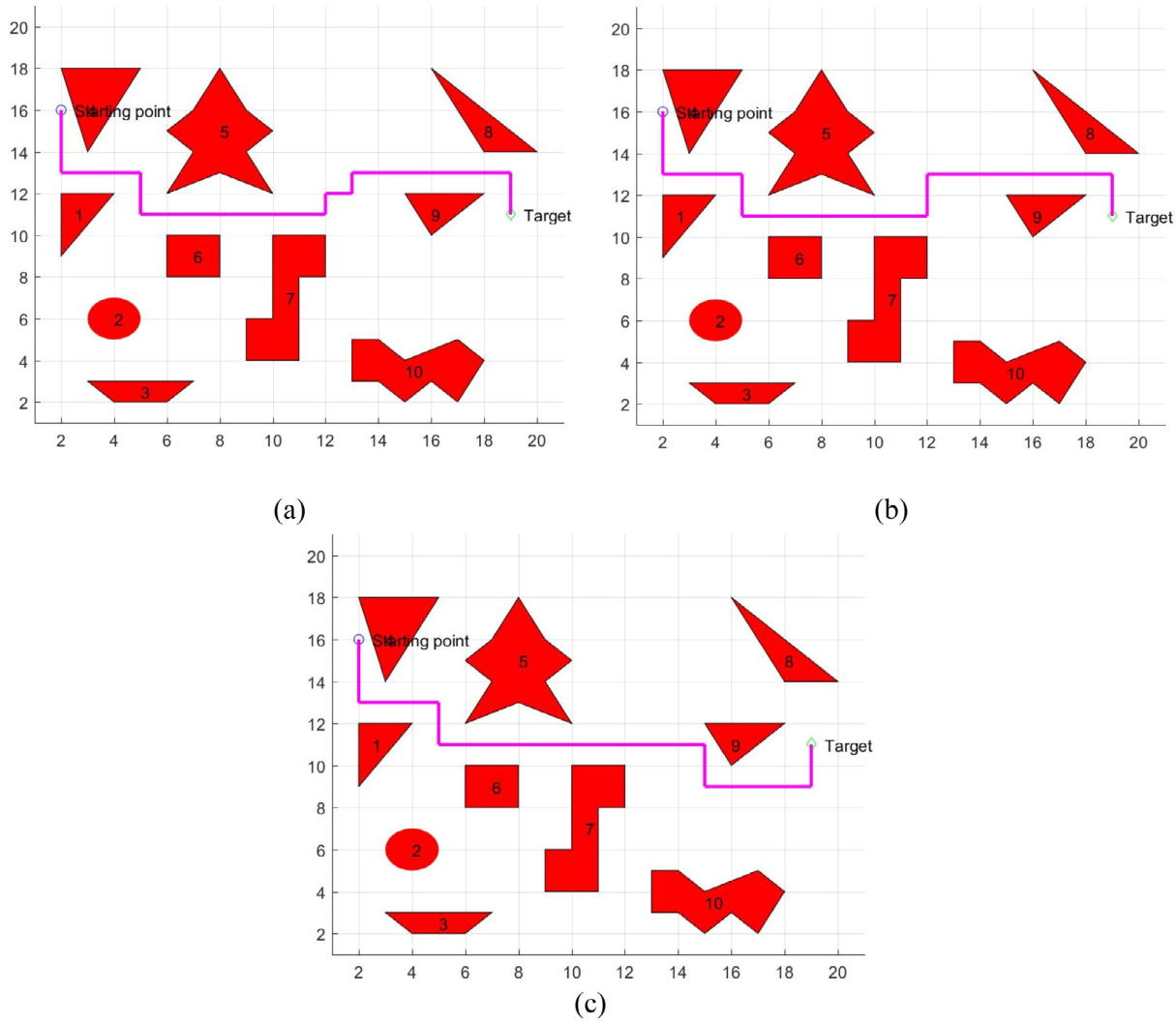


Fig. 6. Test Case 1 with 10 obstacles — the obtained optimal path by (a) Q-learning; (b) IQ-FPA; and (c) IDQ.

Table 3
Performance Comparison between IDQ and IQ-FPA.

Test Case	Average Computational Time (s)				Average Travelled Distance (Unit)				Average Path Smoothness (Rad)			
	IDQ	IQ-FPA	Percentage of Improvement (%)	Statistical Significant?	IDQ	IQ-FPA	Percentage of Improvement (%)	Statistical Significant?	IDQ	IQ-FPA	Percentage of Improvement (%)	Statistical Significant?
1 (8 obstacles)	9.26	3.52	61.99	Yes	26.27	29.33	−11.65	Yes	9.95	22.99	−131.06	Yes
1 (9 obstacles)	10.05	3.62	63.98	Yes	26.37	31.27	−18.58	Yes	9.84	22.72	−130.89	Yes
1 (10 obstacles)	10.28	3.88	62.26	Yes	26.00	29.60	−13.85	Yes	9.42	20.73	−120.06	Yes
1 (11 obstacles)	27.10	3.98	85.31	Yes	26.27	28.73	−9.36	Yes	9.74	18.48	−89.63	Yes
1 (12 obstacles)	23.39	4.00	82.90	Yes	26.53	29.00	−9.31	Yes	9.84	19.32	−96.34	Yes
2	2.87	3.66	−27.53	Yes	26.00	26.80	−3.08	No	9.48	19.84	−109.28	Yes
3	6.02	3.63	39.70	Yes	26.33	26.60	−1.03	No	12.25	18.48	−50.86	Yes
4	6.09	2.70	55.67	Yes	24.07	23.47	2.49	No	10.73	13.72	−27.87	Yes

in most cases, regardless of the number of obstacles. The is due to for Q-learning, the initial values of Q-table are assigned as zero. Hence, the Q-learning begins the exploration process without prior information about the environment, and random action will be taken since there is no guidance in the first place. Such randomness leads to inconsistency in the chosen path for each run, which is closely related to the performance in terms of computational time, travelled distance and path smoothness.

More intriguing results are observed in Table 3, where the IQ-FPA has improved the computational time to a great extent, in

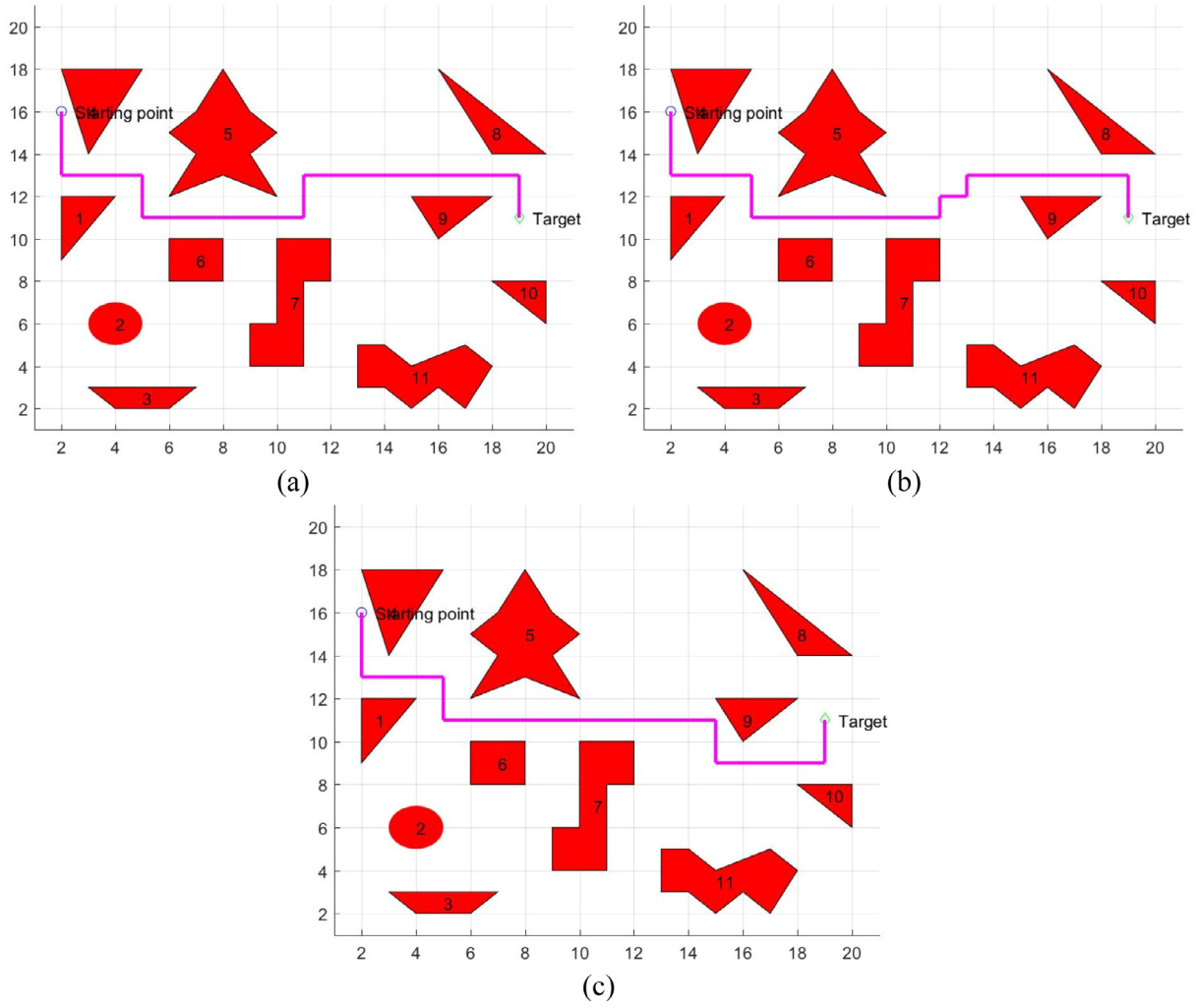


Fig. 7. Test Case 1 with 11 obstacles — the obtained optimal path by (a) Q-learning; (b) IQ-FPA; and (c) IDQ.

Table 4
Performance Comparison between Q-Learning and IQ-FPA in terms of standard deviation.

Test Case	Computational Time (s)			Travelled Distance (Unit)			Path Smoothness (Rad)		
	Q-Learning	IQ-FPA	Percentage of Improvement (%)	Q-Learning	IQ-FPA	Percentage of Improvement (%)	Q-Learning	IQ-FPA	Percentage of Improvement (%)
1 (8 obstacles)	0.43	0.28	34.88	2.86	1.69	40.91	8.38	5.48	34.61
1 (9 obstacles)	0.64	0.27	57.81	7.05	2.43	65.53	17.78	5.84	67.15
1 (10 obstacles)	0.72	0.45	37.5	6.32	2.37	62.5	14.37	7.19	49.97
1 (11 obstacles)	0.35	0.24	31.43	2.17	1.78	17.97	4.98	5.39	−8.23
1 (12 obstacles)	0.33	0.22	33.33	1.75	1.55	11.43	6.00	5.64	6.00
2	1.31	0.27	79.39	15.64	2.20	85.93	35.58	5.35	84.96
3	0.36	0.34	5.55	2.73	2.53	7.33	6.38	5.54	13.17
4	0.27	0.24	11.11	1.82	1.66	8.79	4.52	4.21	6.86

comparison with the IDQ. Particularly, at least 61.99% reduction of computational time is achieved, with significant improvement as much as 85.31% is attained for the test case with 11 obstacles. On the other hand, the paths generated by the IDQ for all cases are shorter and smoother, as compared to IQ-FPA. This is attributed to the cost function used in the IDQ, where the path with the lowest difference between the position of next action state and the target position, which is normally a straight path, is always selected. Hence, fewer turns are involved, leading to shorter travelled distance and better path smoothness. In terms of standard deviation of the results, it can be observed in Table 5 that the performance of IQ-FPA is more consistent than IDQ in terms of computational time, where at least 92.36% improvement is achieved. The large variation in IDQ is tightly correlated with

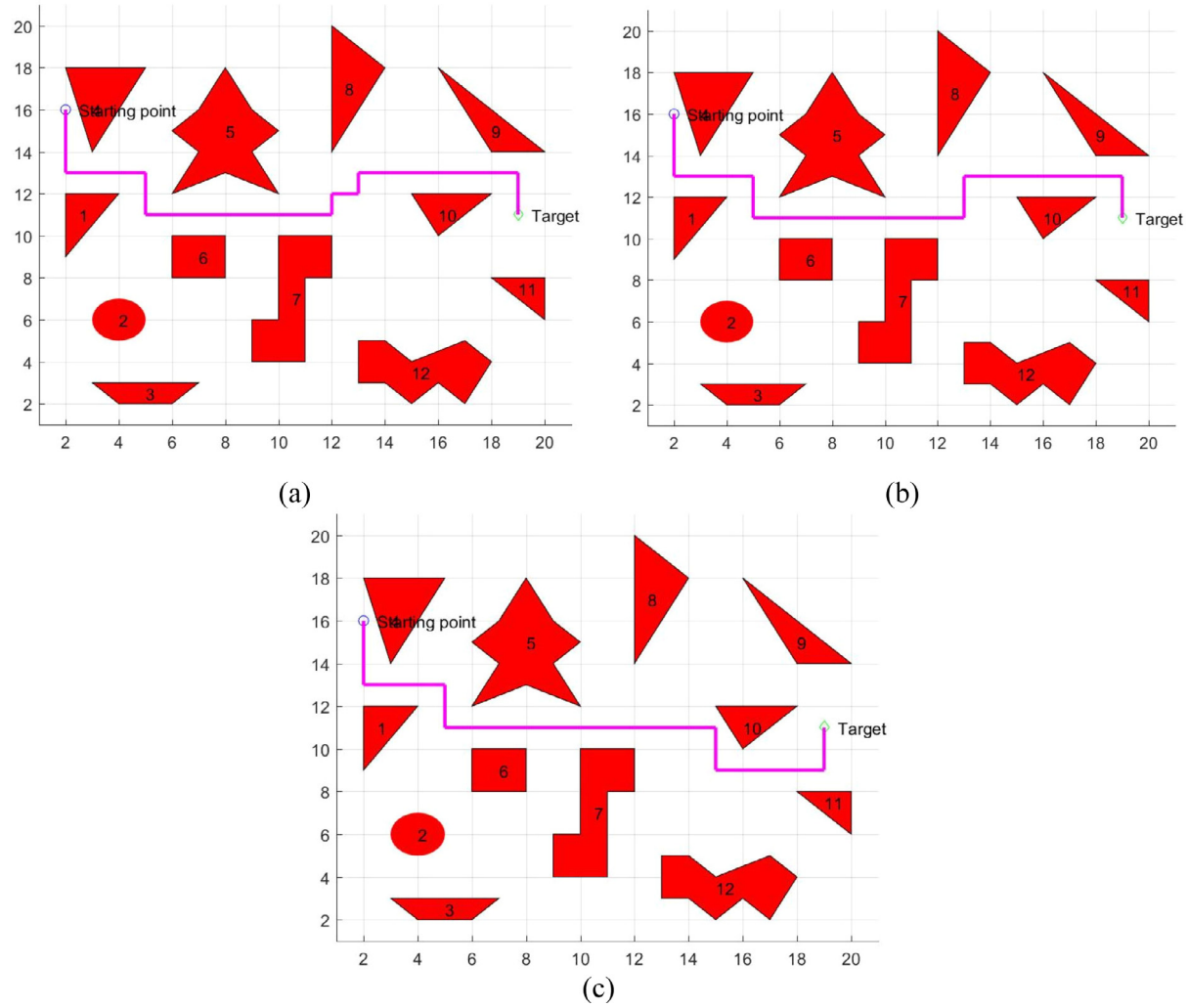


Fig. 8. Test Case 1 with 12 obstacles — the obtained optimal path by (a) Q-learning; (b) IQ-FPA; and (c) IDQ.

Table 5
Performance Comparison between IDQ and IQ-FPA in terms of standard deviation.

Test Case	Computational Time (s)			Travelled Distance (Unit)			Path Smoothness (Rad)		
	IDQ	IQ-FPA	Percentage of Improvement (%)	IDQ	IQ-FPA	Percentage of Improvement (%)	IDQ	IQ-FPA	Percentage of Improvement (%)
1 (8 obstacles)	7.67	0.28	96.35	1.46	1.69	−15.75	1.86	5.48	−194.62
1 (9 obstacles)	6.38	0.27	95.77	1.46	2.43	−66.44	1.79	5.84	−226.26
1 (10 obstacles)	5.89	0.45	92.36	0.00	2.37	–	0.00	7.19	–
1 (11 obstacles)	50.84	0.24	99.53	1.46	1.78	−21.92	1.72	5.39	−213.37
1 (12 obstacles)	30.95	0.22	99.29	2.03	1.55	23.65	1.79	5.64	−215.08
2	0.27	0.27	0.00	0.00	2.20	–	0.29	5.35	−1744.83
3	2.46	0.34	86.18	0.76	2.53	−232.89	2.42	5.54	−128.93
4	4.61	0.24	94.79	4.71	1.66	64.76	3.53	4.21	−19.26

the incompetence of IDQ to escape from the local optima. Nevertheless, IQ-FPA shows a higher standard deviation than IDQ in most cases in terms of travelled distance and path smoothness. The cost function introduced by IDQ provides lower variation in path selection, due to the difference between the position of next action state and targeted position is always the same for the same state and the same action. The selected path is similar or even the same (test case with 10 obstacles) for each run, and thus the average travelled distance and path smoothness will not vary considerably.

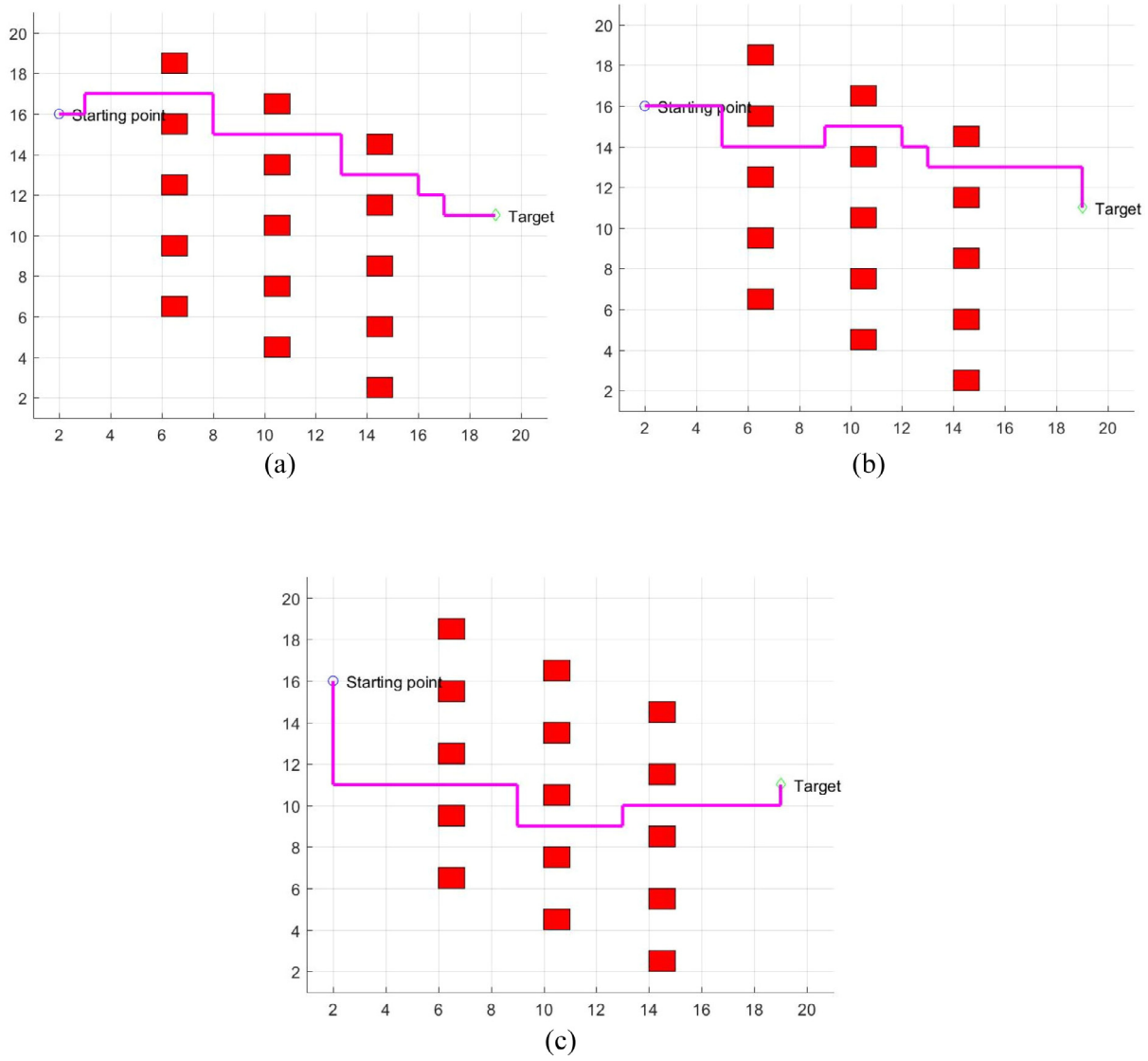


Fig. 9. The obtained optimal path by (a) Q-learning; (b) IQ-FPA and (c) IDQ for Test Case 2.

5.2. Test case 2

This test case includes a multi-obstacles with same size and shape, as presented in Fig. 9 [47]. Such arrangement creates many narrow passages in between the obstacles. The mobile robot has to try to navigate through the narrow passages without colliding with an obstacle. Akin to the previous test case, the IQ-FPA may need to regenerate the feasible solution more frequently, when a large number of obstacles exists. In this test case, 15 obstacles are formed, leading to longer computational time in comparison with Q-learning and IDQ.

On the other hand, as compared to Q-learning, the obtained path by the IQ-FPA is with fewer zigzags, which can be observed from the improvement of around 24.07% in terms of path smoothness in Table 2. This indicates that IQ-FPA has fewer turning in the navigation process, leading to the improvement of 10.87% in the average total travelled distance as well. Also, it can be observed in Table 4 that the performance of IQ-FPA is more consistent than Q-learning in all aspects, where the notable improvement of 79.39%, 85.93% and 84.96% are achieved for the average computational time, average travelled distance, and average path smoothness, respectively. This suggests that incorporate prior knowledge from the FPA gives the mobile robot a better exploration foundation. Without initialization of Q-values in the classical Q-learning, the mobile robot does not have early information of its environment. Therefore, it has to perform action randomly from the start state since there is no guidance information, very often ends up explore blindly and stuck in a pit before reaching the goal, hence, the consistency is in doubt. From Fig. 9, it can be noticed that the path generated by the IDQ is having the least turning due to the straight path is preferred when selecting the next action. This has led to the lowest average path smoothness of 9.48 rad, as presented in Table 3.

Fig. 10 presents the graph of average total steps used over iterations in order to observe the convergence behaviour of all algorithms. As shown in this figure, the IDQ is superior to the Q-learning and IQ-FPA in terms of convergence rate. This is due to the Q-table of IDQ is initialized based on the difference between the position of the next action state and the targeted position. This has somehow guided the mobile robot to move from initial position to target position more efficiently since the information about the distance to the targeted position is available.

5.3. Test case 3

Fig. 11 shows the optimal path generated by the Q-learning, IQ-FPA and IDQ in test case 3, where the mobile robot has to avoid

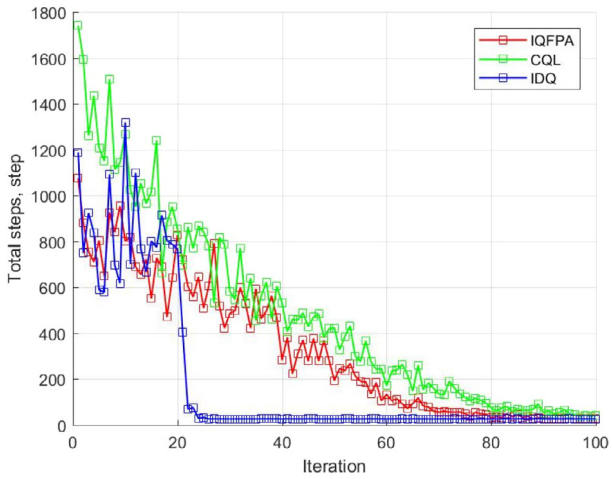


Fig. 10. The convergence behaviour of Q-learning, IQ-FPA and IDQ for test case 2.

the collision with three collateral walls [47]. As shown in Table 2, the mobile robot will only be able to reach the destination point after a large amount of attempts, indicated by the average computational time of 3.65 s for Q-learning, as compared to test case 2 and 4. This is probably due to the environment structure with large free spaces. With more free spaces, the more possible decisions there are, and hence, the mobile robot takes more time to reach its destination point. It can be noticed in Tables 2 and 4 that both Q-learning and IQ-FPA have similar performance.

For this test case, IDQ takes longer computational time of 6.02 s in completing the path, which due to the local optima exist around the starting point and the middle collateral wall. In the case when some initial Q-values are specified with the FPA, the mobile robot is able to follow the optimal policy starts from the initial state. Hence, the average computational time has been accelerated significantly with about 39.7%, as shown in Table 3.

5.4. Test case 4

Fig. 12 presents the map with multi-wall traps [47] and the obtained optimal path by Q-learning, IQ-FPA and IDQ in test case 4. As shown in Table 2, although the IQ-FPA improves both total travelled distance and path smoothness by 2.21% and 9.97%, respectively, as compared to Q-learning, the improvement is not statistically significant as indicated by the T-test. Nonetheless, the proposed IQ-FPA generates an optimal path in relatively short time for this environment, where the time consumption has been reduced by approximately 10.00% and 55.67%, as compared to the

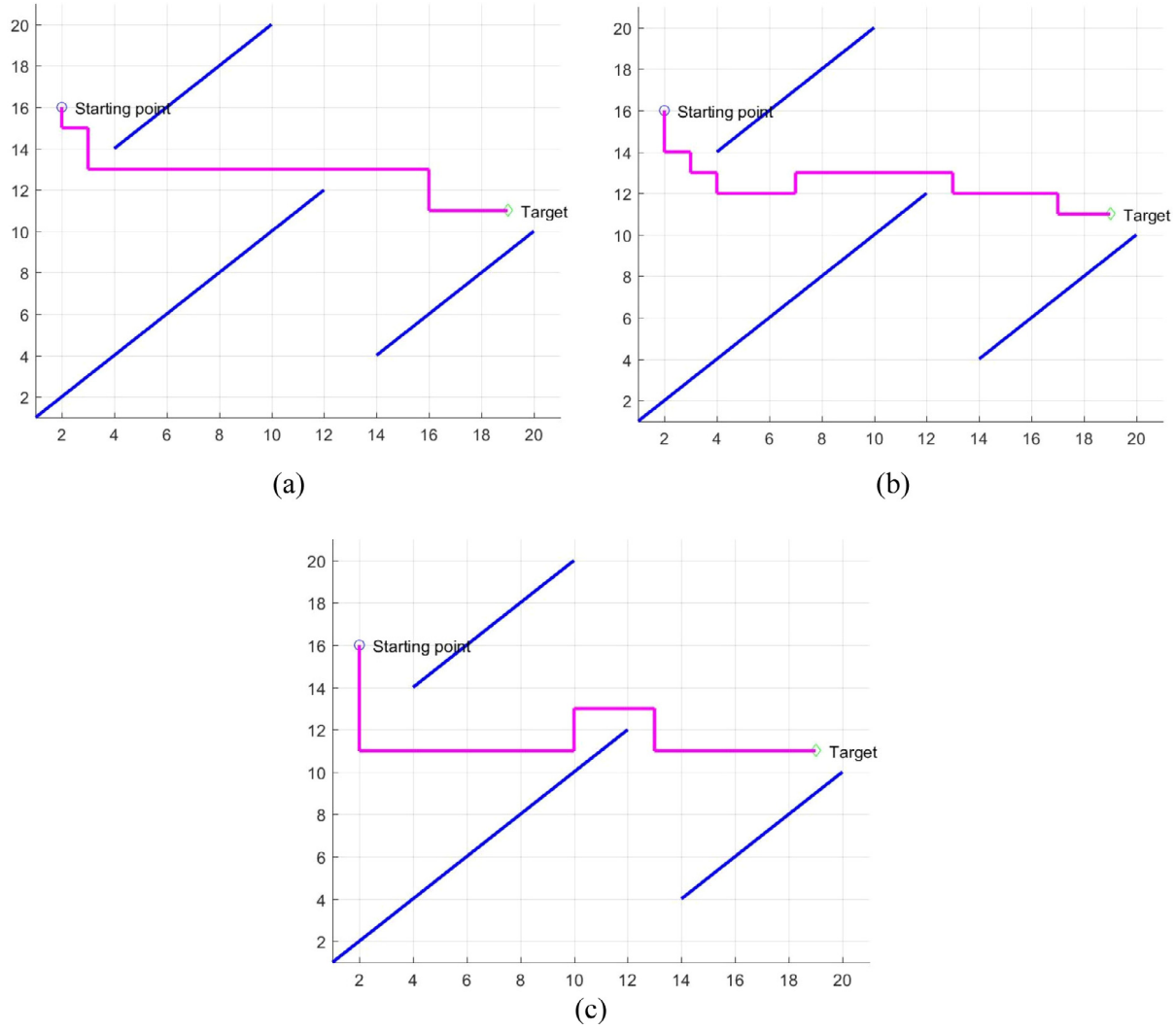


Fig. 11. The obtained optimal path by (a) Q-learning; (b) IQ-FPA and (c) IDQ for Test Case 3.

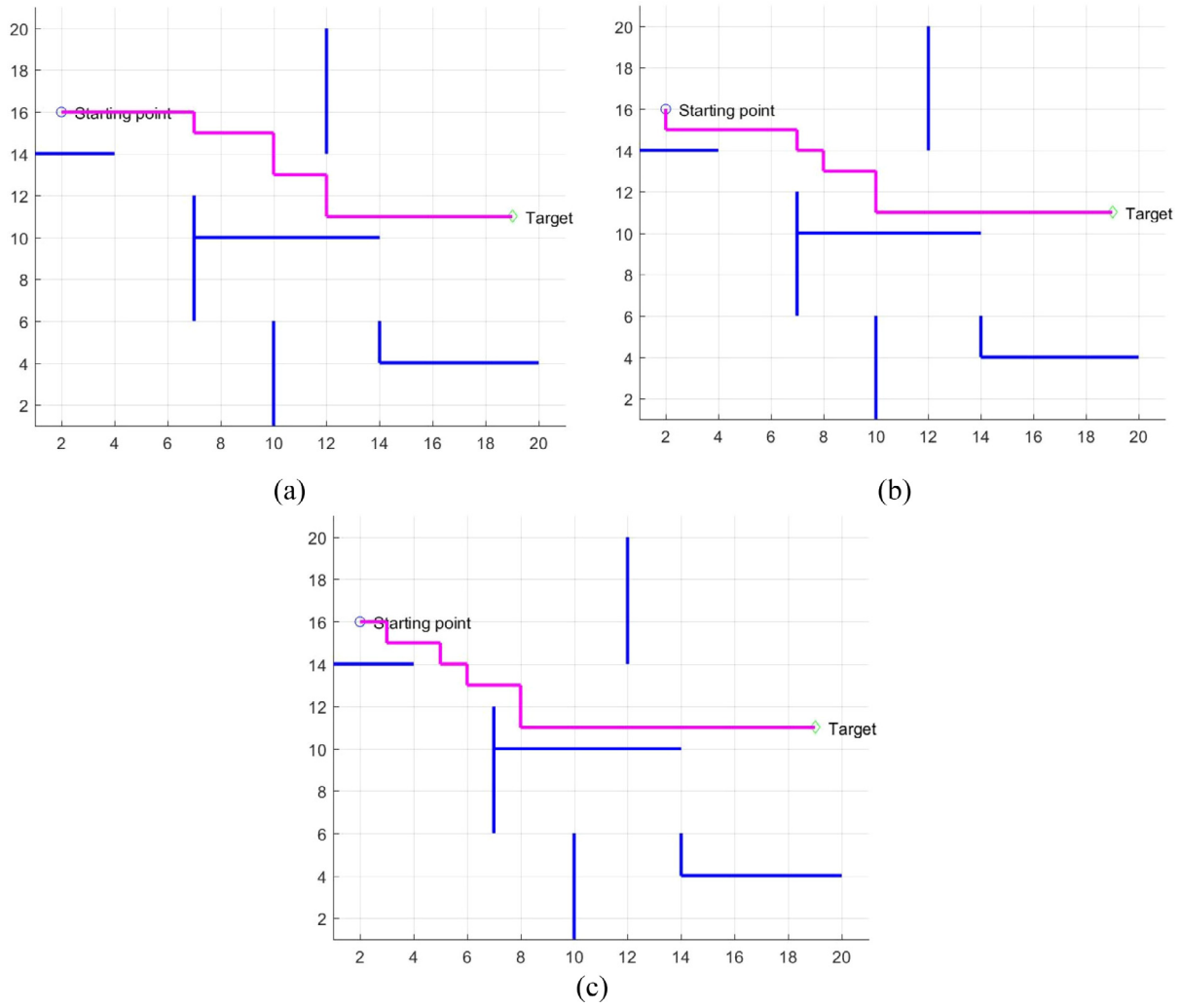


Fig. 12. The obtained optimal path by (a) Q-learning; (b) IQ-FPA and (c) IDQ for Test Case 4.

classical Q-learning and IDQ, respectively. In this test case, IQ-FPA can reduce the computational time to a greater extent. Again, it suggests that with proper initialization of Q-values, the mobile robot can start to explore the environment from its initial state with the better foundation by utilizing the prior knowledge.

The convergence behaviour of all algorithms is illustrated in Fig. 13. In contrast to test case 2, the IDQ takes considerably more iteration steps for convergence but, the IQ-FPA with the utilization of FPA in initializing the Q-values is more efficient than others. The slow convergence of IDQ is due to the existence of a few local optima in test case 4. It should be noted that the similar convergence behaviours are observed for test case 1 and test case 3, and hence those convergence curves are omitted here.

6. Experimental validation of the IQ-FPA in real-world environment

In order to validate the proposed IQ-FPA in the real-world environment, a three-wheeled mobile robot controlled by Arduino Uno is developed, as shown in Fig. 14. The components used are Arduino Uno, chassis with two motors, motor controller L298P, Bluetooth module HC-05, and Li-ion battery with the output voltage of 12v. The Arduino Uno microcontroller is connected to a personal computer using Bluetooth module HC-05. The microcontroller controls motors through L289P motor driver to change

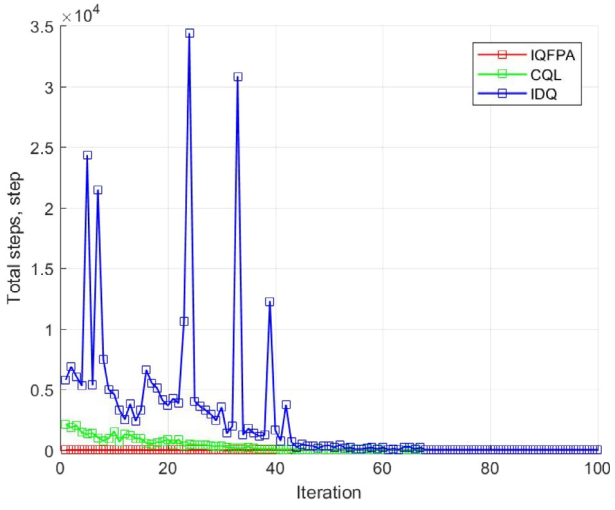


Fig. 13. The convergence behaviour of Q-learning, IQ-FPA and IDQ for test case 4.

the direction of the mobile robot, with speed control is applied. Power is supplied to the board via a power cord from 9v voltage regulator.

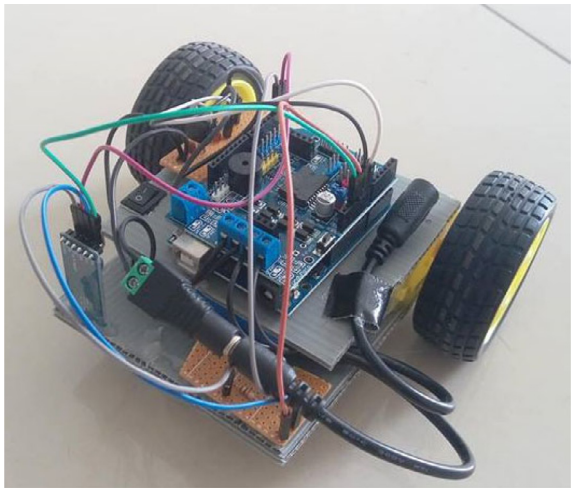


Fig. 14. The assembled mobile robot.

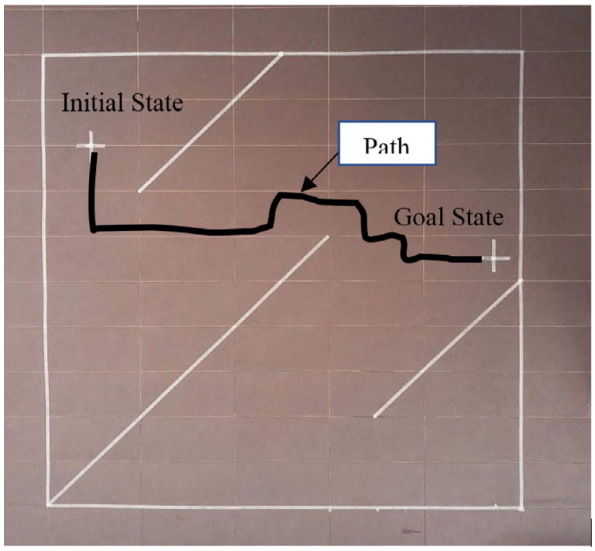


Fig. 15. Experimental navigation map of test case 3 in real-world validation.

The navigation map in test case 3 is used to validate the navigation capability of Q-learning, IQ-FPA and IDQ in a real-world environment. The parameter setting of the switching probability, p, and population size, n, is as in Table 1. Fig. 15 presents the navigation map of test case 3 in real-world validation, with a dimension of 3 m $\times$ 3 m. The simulated navigation map with the grid size of 20 $\times$ 20 has been resized, with one unit in the simulation corresponds to 0.15 m in a real-world environment. The robot does not include a sensor to detect the obstacles. Thus, no feedback system is applied. The position of obstacles is coded in the programming beforehand (with the ratio of 1 unit: 0.15 m). Moreover, the mobile robot is unable to turn precisely to the desired angle occasionally and the actual travelled distance in the real-world environment is less than one unit as in the simulation. In order to overcome this limitation, a manual adjustment is done by the user, where the user can send a command through MATLAB to the Arduino UNO microcontroller to adjust the desired turning angle and move the robot to travel one unit (0.15 m) in the real-world environment, when necessary. Also, only the actual travelled distance by the mobile robot in real-world environment is measured and compared. Comparison in terms of computational time is not practical in this case due to the mobile robot has different speeds in simulation and experimental validation. The constant speed can be achieved in simulation, however, in a real-world environment, it has some external factors to be considered. Exemplary, friction of the robot wheel due to the floor surface condition can affect the navigation speed. The actual path smoothness is not considered as the mobile robot fails to travel in a straight line along with all its trajectory to the goal state.

Fig. 15 shows the path travelled by the mobile robot in the real-world environment, while Table 6 presents the comparison of both actual and simulated total travelled distance taken by the mobile robot to reach the goal state from the initial state in real-world experimental validation. From Table 6, it can be observed that a small percentage of deviation of 0.88% to 1.68% is observed between the actual travelled distance and the simulated travelled distance. This deviation may due to the additional distance travelled by the mobile robot in the real-world environment as it cannot move in a straight path as in the simulation. Random error and error when using measuring tools might also affect the results. In addition, the presence of transmission of signal error from computer to Arduino UNO controller, the frictional error between the robot wheel and floor, and rotating element's frictional error may lead to impreciseness in the real-world environment, as discussed in [48].

Table 6
Comparison of actual and simulated computational time and travelled distance of the Q-learning, IQ-FPA and IDQ in real-world validation.

Algorithm	Distance Travelled (unit)		
	Simulation	Actual	Percentage of Deviation (%)
Q-learning	22	22.37	1.68
IQ-FPA	24	23.57	1.79
IDQ	26	26.23	0.88

7. Conclusion

In this study, an improved Q-learning based on FPA, specifically, the IQ-FPA, has been proposed for the path planning of mobile robot in different environments with static obstacles in various shapes, sizes and layout. Through the integration of the prior knowledge obtained from the FPA into the classical Q-learning, simulation results demonstrate that initialization of the Q-values serves as a good exploration foundation to accelerate the learning process of the mobile robot. As compared with the classical Q-learning, the computational time has been improved significantly for test case 1 (8 obstacles and 9 obstacles) and test case 4, with the percentage of improvement of 13.3%, 10.4% and 10%, respectively. In terms of average travelled distance and smoothness of path, simulation results confirm that both approaches are able to converge to the optimal path with a similar trajectory, however, the IQ-FPA outperforms the Q-learning with higher consistency. On the other hand, performance comparison with IDQ indicates that utilizing the FPA into Q-learning has reduced the computational time significantly in most cases. However, the paths produced by the IDQ are shorter and smoother than those generated by IQ-FPA in most cases. This is presumably due to the straight path is always preferable by IDQ when performing the next action. In addition, the real-world validation experiment using a three-wheeled mobile robot indicates a good agreement of the effectiveness of IQ-FPA.

Despite initializing the Q-table using FPA provides some guides prior to the exploration and exploitation using Q-learning, the IQ-FPA still selects the action randomly when there are more than one actions with same values at a state. The ε-greedy method in action selection can be used in this regard, in which the mobile robot will choose a random action with probability ε and a probability of $(1-\varepsilon)$ for the optimal Q-value with the largest value [18]. Since only static path planning is considered in this study, it is recommended that applying the IQ-FPA in dynamic path planning which involves dynamic moving obstacles is another direction to pursue in the future study. More robots can be considered in future work, where the collision avoidance with other moving robots and obstacles can be studied. This is more reflected in the real-world environment which requires a proper dynamic path planning. In addition, as the real environment may involve a large area in the path planning, the implementation of IQ-FPA may not be effective when too many states and actions to be considered. An artificial neural network can be integrated in IQ-FPA in this regard, as the artificial neural network has shown promising results in processing spacious states and actions in path planning [20].

Acknowledgements

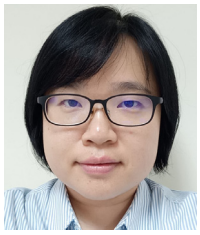
The authors would like to express the deepest appreciation to the Ministry of Education Malaysia, for funding this project through the Fundamental Research Grant Scheme, Malaysia (FRGS – Vot K070). Additional support from Universiti Tun Hussein Onn Malaysia (UTHM) in the form of GPPS Vot H034 is also gratefully acknowledged.

References

[1] S.G. Tzafestas, Introduction to Mobile Robot Control, Elsevier, 2013.
[2] Z. Tahir, et al., Potentially guided bidirectionalized RRT* for fast optimal path planning in cluttered environments, Robot. Auton. Syst. 108 (2018) 13–27.
[3] B. Fu, et al., An improved A* algorithm for the industrial robot path planning with high success rate and short length, Robot. Auton. Syst. 106 (2018) 26–37.
[4] O. Zaki, M. Dunnigan, A navigation strategy for an autonomous patrol vehicle based on multi-fusion planning algorithms and multi-paradigm representation schemes, Robot. Auton. Syst. 96 (2017) 133–142.
[5] M. Davoodi, Bi-objective path planning using deterministic algorithms, Robot. Auton. Syst. 93 (2017) 105–115.
[6] T.T. Mac, et al., Heuristic approaches in robot path planning: A survey, Robot. Auton. Syst. 86 (2016) 13–28.
[7] B. Kovács, et al., A novel potential field method for path planning of mobile robots by adapting animal motion attributes, Robot. Auton. Syst. 82 (2016) 24–34.
[8] X. Zhang, et al., A novel phase angle-encoded fruit fly optimization algorithm with mutation adaptation mechanism applied to UAV path planning, Appl. Soft Comput. J. 70 (2018) 371–388.
[9] X. Liang, et al., A geometrical path planning method for unmanned aerial vehicle in 2D/3D complex environment, Intell. Serv. Robot. 11 (3) (2018) 301–312.
[10] A. Abbadi, R. Matousek, Hybrid rule-based motion planner for mobile robot in cluttered workspace: A combination of RRT and cell decomposition approaches, Soft Comput. 22 (6) (2018) 1815–1831.
[11] N.B.A. Latip, R. Omar, S.K. Debnath, Optimal path planning using equilateral spaces oriented visibility graph method, Intl. J. Electr. Comput. Eng. 7 (6) (2017) 3046–3051.
[12] S. Haghzad Klidbary, S. Bagheri Shouraki, S. Sheikhpour Kourabbaslou, Path planning of modular robots on various terrains using Q-learning versus optimization algorithms, Intell. Serv. Robot. 10 (2) (2017) 121–136.
[13] W.-Y. Shin, et al., Line segment selection method for fast path planning, Int. J. Control Autom. Syst. 15 (3) (2017) 1322–1331.
[14] M. Wulfmeier, et al., Large-scale cost function learning for path planning using deep inverse reinforcement learning, Int. J. Robot. Res. 36 (10) (2017) 1073–1087.
[15] D.L. Cruz, W. Yu, Path planning of multi-agent systems in unknown environment with neural kernel smoothing and reinforcement learning, Neurocomputing 233 (2017) 34–42.
[16] D. Luviano, W. Yu, Continuous-time path planning for multi-agents with fuzzy reinforcement learning, J. Intell. Fuzzy Systems 33 (1) (2017) 491–501.
[17] M. Duguleana, G. Mogan, Neural networks based reinforcement learning for mobile robots obstacle avoidance, Expert Syst. Appl. 62 (Supplement C) (2016) 104–115.
[18] A. Arin, G. Rabadi, Integrating estimation of distribution algorithms versus Q-learning into Meta-RaPS for solving the 0-1 multidimensional knapsack problem, Comput. Ind. Eng. 112 (Supplement C) (2017) 706–720.
[19] Y.-H. Wang, T.-H.S. Li, C.-J. Lin, Backward Q-learning: the combination of Sarsa algorithm and Q-learning, Eng. Appl. Artif. Intell. 26 (9) (2013) 2184–2193.
[20] M. Duguleana, G. Mogan, Neural networks based reinforcement learning for mobile robots obstacle avoidance, Expert Syst. Appl. 62 (2016) 104–115.
[21] P. Das, H. Behera, B. Panigrahi, Intelligent-based multi-robot path planning inspired by improved classical Q-learning and improved particle swarm optimization with perturbed velocity, Eng. Sci. Technol. Intl. J. 19 (1) (2016) 651–669.
[22] I. Carlucho, et al., Incremental Q-learning strategy for adaptive PID control of mobile robots, Expert Syst. Appl. 80 (2017) 183–199.
[23] P. Rakshit, et al., Realization of an adaptive memetic algorithm using differential evolution and Q-learning: A case study in multirobot path planning, IEEE Trans. Syst. Man Cybern. Syst. 43 (4) (2013) 814–831.
[24] P. Rakshit, et al., ABC-TDQL: An adaptive memetic algorithm, in: 2013 IEEE Workshop on Hybrid Intelligent Models and Applications, HIMA, 2013.
[25] C.-H. Oh, T. Nakashima, H. Ishibuchi, Initialization of Q-values by fuzzy rules for accelerating Q-learning, in: Neural Networks Proceedings, 1998. IEEE World Congress on Computational Intelligence. The 1998 IEEE International Joint Conference on, IEEE, 1998.
[26] E. Wiewiora, Potential-based shaping and Q-value initialization are equivalent, J. Artificial Intelligence Res. 19 (2003) 205–208.
[27] S. Koenig, R.G. Simmons, The effect of representation and knowledge on goal-directed exploration with reinforcement-learning algorithms, Mach. Learn. 22 (1–3) (1996) 227–250.
[28] Y. Song, et al., An efficient initialization approach of Q-learning for mobile robots, Intl. J. Control Autom. Syst. 10 (1) (2012) 166–172.
[29] D. Charypar, K. Nagel, Q-learning for flexible learning of daily activity plans, Transp. Res. Record J. Transp. Res. Board 2005 (1935) 163–169.
[30] M. Simsek, et al., Improved decentralized Q-learning algorithm for interference reduction in LTE-femtocells, in: Wireless Advanced (WiAd), IEEE, 2011.
[31] C. Yan, X. Xiang, A path planning algorithm for UAV based on improved Q-learning, in: 2018 2nd International Conference on Robotics and Automation Sciences, ICRAS, 2018.
[32] P. Rakshit, et al., ABC-TDQL: An adaptive memetic algorithm, in: Hybrid Intelligent Models and Applications (HIMA), 2013 IEEE Workshop on, IEEE, 2013.
[33] Y. Wang, C.W. de Silva, A machine-learning approach to multi-robot coordination, Eng. Appl. Artif. Intell. 21 (3) (2008) 470–484.
[34] A.K. Sadhu, et al., Synergism of firefly algorithm and Q-learning for robot arm path planning, Swarm Evol. Comput. 43 (2018) 50–68.
[35] X.-S. Yang, Flower pollination algorithm for global optimization, in: UCNC, Springer, 2012.
[36] G. Zhou, R. Wang, Y. Zhou, Flower pollination algorithm with runway balance strategy for the aircraft landing scheduling problem, Cluster Comput. 21 (3) (2018) 1543–1560.
[37] Y. Zhou, et al., Using flower pollination algorithm and atomic potential function for shape matching, Neural Comput. Appl. 29 (6) (2018) 21–40.
[38] P. Ong, et al., Modeling and optimization of cold extrusion process by using response surface methodology and metaheuristic approaches, Neural Comput. Appl. 29 (11) (2018) 1077–1087.
[39] Y. Zhou, R. Wang, An improved flower pollination algorithm for optimal unmanned undersea vehicle path planning problem, Int. J. Pattern Recognit. Artif. Intell. 30 (04) (2016) 1659010.
[40] R. Wang, et al., Flower pollination algorithm with bee pollinator for cluster analysis, Inform. Process. Lett. 116 (1) (2016) 1–14.
[41] C.J. Watkins, P. Dayan, Q-learning, Mach. Learn. 8 (3–4) (1992) 279–292.
[42] F.L. Lewis, D. Liu, Reinforcement Learning and Approximate Dynamic Programming for Feedback Control, Wiley, 2013.
[43] A. Konar, et al., A deterministic improved Q-learning for path planning of a mobile robot, IEEE Trans. Syst. Man Cybern. Syst. 43 (5) (2013) 1141–1153.
[44] R. Salgotra, U. Singh, Application of mutation operators to flower pollination algorithm, Expert Syst. Appl. 79 (Supplement C) (2017) 112–129.
[45] L. Khriji, et al., Mobile robot navigation based on Q-learning technique, Intl. J. Adv. Robot. Syst. 8 (1) (2011) 4.

[46] V. François-Lavet, R. Fonteneau, D. Ernst, How to discount deep reinforcement learning: Towards new dynamic strategies, 2015, arXiv preprint arXiv:1512.02011.
[47] Z. Yijing, et al., Q learning algorithm based UAV path learning and obstacle avoidence approach, in: Control Conference (CCC), 2017 36th Chinese, IEEE, 2017.
[48] S. Ghosh, B.K. Panigrahi, D.R. Parhi, Analysis of FPA and BA meta-heuristic controllers for optimal path planning of mobile robot in cluttered environment, IET Sci. Meas. Technol. 11 (7) (2017) 817–828.

Low Ee Soong is currently pursuing Ph.D. in Universiti Tun Hussein Onn Malaysia (UTHM) while graduated from UTHM in 2017 for Bachelor's degree in Mechanical Engineering. Major research interests are Q-learning, optimization algorithms, robotic and automation



Ong Pauline has a first degree in Pure Mathematics from Universiti Sains Malaysia, Malaysia. After receiving her Ph.D. from the Universiti Sains Malaysia in 2011, she worked for some time as post-doctoral fellow at School of Mathematical Sciences, Universiti Sains Malaysia before joining Universiti Tun Hussein Onn Malaysia where she has had a permanent post since 2013. Her research interest concerns neural networks and evolutionary computation over large scale of applications.

Cheah Kah Chun received his first degree in mechanical engineering from Universiti Tun Hussein Onn Malaysia in 2018. Currently he is working as mechanical design engineer in Canon Machinery(M) Sdn Bhd.