Engineering Science and Technology, an International Journal 19 (2016) 651–669

HOSTED BY

Contents lists available at ScienceDirect

Engineering Science and Technology, an International Journal

journal homepage: http://www.elsevier.com/locate/jestch

Full Length Article

Intelligent-based multi-robot path planning inspired by improved classical Q-learning and improved particle swarm optimization with perturbed velocity

P.K. Das [a,*], H.S. Behera [a], B.K. Panigrahi [b]

[a] Department of Computer Science & Engineering and Information Technology, VSSUT, Burla, Odisha, India
[b] Department of Electrical Engineering, IIT, Delhi, India

ARTICLE INFO

Article history:
Received 30 July 2015
Received in revised form
9 September 2015
Accepted 22 September 2015
Available online 15 December 2015

Keywords:
Q-learning
Path planning
Mobile robots
Energy
IPSO-DV
Khepera II

ABSTRACT

Classical Q-learning takes huge computation to calculate the Q-value for all possible actions in a particular state and takes large space to store its Q-value for all actions, as a result of which its convergence rate is slow. This paper proposed a new methodology to determine the optimize trajectory of the path for multi-robots in clutter environment using hybridization of improving classical Q-learning based on four fundamental principles with improved particle swarm optimization (IPSO) by modifying parameters and differentially perturbed velocity (DV) algorithm for improving the convergence. The algorithms are used to minimize path length and arrival time of all the robots to their respective destination in the environment and reducing the turning angle of each robot to reduce the energy consumption of each robot. In this proposed scheme, the improve classical Q-learning stores the Q-value of the best action of the state and thus save the storage space, which is used to decide the Pbest and gbest of the improved PSO in each iteration, and the velocity of the IPSO is adjusted by the vector differential operator inherited from differential evolution (DE). The validation of the algorithm is studied in simulated and Khepera-II robot.

1. Introduction

The path planning problem in mobile robotics is considered a complex task. It [1] determines a path for the robot to reach in predefine goal location from a specified starting location without hitting various obstacles in the given environment. The path planning problem has been classified into different categories. One of the classifications is static and dynamic path planning based on the environmental information. In the static path planning, the obstacles and goals are motionless. But in the dynamic path planning, the obstacles and goals are moving in the environment each time, and also the environment is changing every time. Another classification is local and global path planning. Robot navigates through the obstacles by steps and determines its next position to reach the goal by satisfying constraints like path, time and energy optimality [2–8], with the help of the local path planning scheme. In global planning, the robot decides the entire collision free path before its movement toward the goal from a specified initial position. The above mentioned global planning is termed as *offline planning* [9]. Local path-planning, which includes navigation and online planning, is sometimes referred to as navigation only in the literature. The phrase motion planning, which includes the notion of time with the position of a robot on a planned trajectory, is often used in the context of path-planning. In path planning, we need to generate a collision free trajectory path in the world map by avoiding the obstacles, and path is optimized with respect to certain criteria. However, the environment may be vast, dynamic, imprecise, uncertain and partially non-structured. In such environment, the mobile robots often used the machine learning to become aware about its environment. In early, research was used the supervised learning to train the robots to determine its next position in the given world map based on the sensory data gained from the environment. But it has provided the best result for mobility management of robots in fixed maps. However, it is difficult to guide the robot to decide its next position, although the acquired knowledge to small changes in the robot's world map. So a complete training is required for the robot with both old and new sensory data–action pair to overcome the above problem.

Reinforcement learning is considered as an alternative learning policy, which is based on the principle reward and penalty. In this learning an agent performs an action on the environment and

* Corresponding author. Tel.: +919439005466; fax: 06632430573.
E-mail address: daspradipta78@gmail.com (P.K. Das).
Peer review under responsibility of Karabuk University.

http://dx.doi.org/10.1016/j.jestch.2015.09.009

receives an immediate reward or penalty based on the action. The learner adapts its parameter based on the status of (reward/penalty) the feedback signal from its environment. Since the exact value of the futuristic reward is not known, it is guessed from the knowledge about the robot's world map. The primary advantage of reinforcement learning lies in its inherent power of automatic learning even in the presence of small changes in the world map. There exist extensive research on multi-robot navigation on reinforcement learning that has been tested in many simulated environments [3,10–18] but on a limited basis in real-world scenarios. A real-world environment poses more challenges than a simulated environment, such as enlarged state spaces [11], increased computational complexity, significant safety issues (a real robot can cause real damage), and longer turnaround times for results. This research measures how well reinforcement-learning technique, such as Q-learning, can apply to the real robot for navigational problem [19,20]. The author [21] has implemented the multi-robot navigation in the Khepera-II environment by designing an adaptive memetic algorithm (AMA) by utilizing the composite benefits of Q-learning for local refinement and differential evolution (DE) for global search. In the paper [22] multi-robot navigation is solved in the real world map by hybridization of the Artificial Bee Colony (ABC) for global search and Q-learning for local refinement; the performance is evaluated in terms of runtime, cost function and accuracy. The paper [23] used the Lyapunov design principle in the reinforcement learning to switch control policy instead of training the agent for control policy and combine PSO and Q-value-based reinforcement learning for neuro-fuzzy system design. The multi-goal Q-learning algorithm has been modeled to solve the multiple goal learning problems in the virtual team [24]. In this presented paper, we modified the classical Q-learning algorithm (CQL), hereafter called improved Q-learning (IQL), and is integrated with an improved particle swarm optimization (IPSO) hybridized with DV, called IQ value-based IPSO-DV, to improve its performance for path-planning problem of multi-robots.

The online trajectory path planning of multi-robot from specified initial position to a goal position without hitting obstacles and a teammate is presented in this work. In a multi-robot path planning problem, each robot has a specified initial and goal position in a given environment, and each robot has to plan its collision free path without hitting any of the colleagues or obstacles present in the map through offline or online approach. The obstacles present in the environment may be static or dynamic. However, in this paper, we have considered static obstacles in the given environment for the robots, and the robot is treated as a dynamic obstacle for other robots. The path planning problem for multi-robot can be solved by two different approaches, such as centralized or distributed approach. The cost or objective function and the constraints for computing the path for all the robots are considered together in the centralized approach [25,26], whereas in the distributed planning [27] each robot determined its collision free trajectory path independently without making collision with static obstacles or colleagues at the time of moving toward the destination. The multi-robot navigational problem has divided into two smaller problems: velocity planning and path planning. In the first phase, each robot constructs the individual path by satisfying the optimum path for each robot. In the velocity planning, each robot avoids the collision with obstacles and the teammates. Many researchers are using the multi-robot navigational problem as a meta-heuristic optimization problem, and different meta-heuristic optimization algorithms have been used to generate the optimum trajectory collision free path for each robot, such as genetic algorithm (GA), particle swarm optimization (PSO) [28,29], and differential evolution (DE) [26].

In our research, we have integrated reinforcement learning techniques with an improved particle swarm optimization (IPSO) hybridized with DV to compute the optimal trajectory path of all the robots from specified initial positions to fixed goal positions in the cluttered environment and with an objective to minimize the path distance for each robot. In this paper, we enhance our implementation of IQ value-based IPSO-DV algorithm to determine the trajectory of path for multiple robots from predefined initial positions to predefined target positions in the environment with an objective to minimize the path length of all the robots. The result shows that the algorithm can improve the solution quality in a reasonable amount of time. This paper contributed to improve the classical Q-learning algorithm for improving the global path planning problem of the multi-robots by integrating it with IPSO-DV and improve the convergence rate, and performance metrics are evaluated in terms of path deviation, path traveled, number of turns and total time required to reach the destination. Finally, the efficiency of the IQ value-based IPSO-DV will be proven through the simulation as well as the Khepera robot, and the results are compared with other evolutionary computing, such an IPSO-DV, IPSO and DE.

The remaining part of the paper is outlined as follows. Problem formulation for the multi-robot navigation has been elaborated in section 2. Classical Q-learning and its limitation is introduced in section 3. Classical Q-learning has improved based on the proposed properties called improved Q-learning, and overcomes the limitation of the classical Q-learning, as introduced in section 4. The algorithm for the improved Q-learning is presented in section 5. The classical particle swarm optimization and improved particle swarm optimization are described briefly in section 6. A differential evolution algorithm is presented in section 7. Theoretical description and its algorithm of the hybrid IPSO-DV for path planning of multi-robot is presented in section 8. The QIPSO-DV algorithm-based multi-robot path planning is given in section 9. Implementation of hybrid QPSO-DV and performance analysis is briefly described in section 10. Section 11 provides the experimental result with the Khepera II robot. Conclusions are listed in section 12.

2. Problem formulation for multi-robot navigation

The multi-robot navigation problem is formulated as to compute the next location for each robot from its current location in the environment by avoiding collision with teammates (which is dynamic in nature) and obstacles (which are static in nature) in its path to reach the goal. The set of principles is considered in formulating multi-robot path planning problem with the help of the following assumption:

Assumptions

1. Current position/initial position and goal positions/target position of all the robot is known in prior coordinate system.
2. At any instant of time, the robot can decide any action from a set of predefined actions for its motion.
3. Each robot is performing its action until reaching their respective target position in steps.

The following principles have been taken care of for satisfying the given assumptions.

1. For determining the next position from its current position, the robot tries to align its heading direction toward the goal position.
2. The alignment may cause a collision with the robots/obstacles (which are static in nature) in the environment. Hence, the robot turns its heading direction with a certain angle either to the left or right for determining its next position from its current position.
3. If a robot can align itself with a goal without collision, then it will move to that determined position.
4. If the heading direction is rotated to the left or right, then it is required for the robot to rotate the same angle about its z-axis; if it is the same for more than one, then decide randomly.

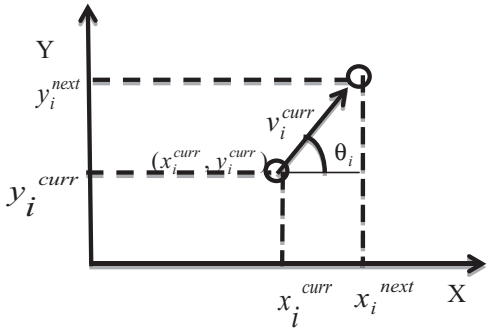


Fig. 1. Representation of next location from current location for i^{th} robot.

Consider the initial position of the i^{th} *robot* at time t is (x_i^{curr}, y_i^{curr}), the next position of the same robot at time $(t+\Delta t)$ is (x_i^{next}, y_i^{next}), v_i^{curr} is the velocity of the robot R_i and (x_i^{goal}, y_i^{goal}) is the target or goal position of the robot R_i.

So the expression for the next position (x_i^{next}, y_i^{next}) can be derived from the Fig. 1 as follows:

$$x_i^{next} = x_i^{curr} + v_i^{curr}\cos\theta_i \Delta t \tag{1}$$

$$y_i^{next} = y_i^{curr} + v_i^{curr}\sin\theta_i \Delta t \tag{2}$$

When $\Delta t = 1$, Eqs. (1) and (2) are reduced to

$$x_i^{next} = x_i^{curr} + v_i^{curr}\cos\theta_i \tag{3}$$

$$y_i^{next} = y_i^{curr} + v_i^{curr}\sin\theta_i \tag{4}$$

Consider initially, the robot R_i is placed in the location at (x_i^{curr}, y_i^{curr}). We want to find the next location of the robot (x_i^{next}, y_i^{next}) by joining of the two points between $\{(x_i^{curr}, y_i^{curr}), (x_i^{next}, y_i^{next})\}$ and $\{(x_i^{next}, y_i^{next}); (x_i^{goal}, y_i^{goal})\}$ should not touch the obstacle in the world map, as represented in Fig. 2, and minimizes the total path length from current position to a goal position without touching the obstacle by forming constraint. Then the objective function F_1, which determines the trajectory path length for n, number of robots, is

$$F_1 = \sum_{i=1}^{n} \left\{ \sqrt{\left((x_i^{curr} - x_i^{next})^2 + (y_i^{curr} - y_i'^{next})^2\right)} + \sqrt{\left((x_i^{next} - x_i^{goal})^2 + (y_i^{next} - y_i^{goal})^2\right)} \right\} \tag{5}$$

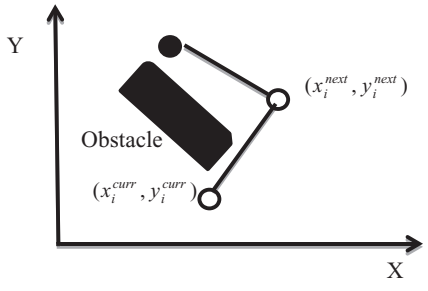


Fig. 2. Selection of next position (x_i^{next}, y_i^{next}) from current position (x_i^{curr}, y_i^{curr}) for avoiding collision with obstacle.

By putting the value x_i^{next} and y_i^{next} from the expressions (3) and (4) into the expression (5), we obtain

$$F_1 = \sum_{i=1}^{n} \left\{ v_i^{curr} + \sqrt{\begin{matrix}(x_i^{curr} + v_i^{curr}\cos\theta_i - x_i^{goal})^2 \\ + (y_i^{curr} + v_i^{curr}\sin\theta_i - y_i^{goal})^2\end{matrix}} \right\} \tag{6}$$

The second objective function is considered a repulsive function. The repulsive function is defined as a function of the relative distance between the robot and obstacles. Let $d_{\min}(X_p)$ the minimum distance of X_p from the obstacles. So the repulsive field for each static obstacle is defined in expression (7).

$$F_2(X_p) = \begin{cases} \frac{k}{\gamma}\left(\frac{1}{d_{\min}(X_p)} - \frac{1}{\eta_0}\right), & if\ d_{\min}(X_p) \le \eta_0 \\ 0, & otherwise \end{cases} \tag{7}$$

where η_0 is the influence range of the obstacle, k is positive constant and $\gamma \ge 2$ shapes the radial profile of potential. The third function is considered on the basis of prediction of the dynamic object in the world map, which will appear dynamically in the trajectory path of robots. So the robot has to predict the dynamic obstacle position before deciding its next position for a moment. The objective function, including the prediction principle, is expressed as

$$F_3 = \sum_{i=1}^{n} \sqrt{(x_p - x_i^{goal})^2 + (y_p - y_i^{goal})^2} \tag{8}$$

Again smoothness of the path is considered using fourth objective function. The smoothness is expressed as the angle between two hypothetical lines connecting the goal point and two successive positions of the robot's in each iteration, i.e. $gbest^i$ and $gbest^{i-1}$ in i^{th} iteration. The objective function for the smoothness of the path is expressed mathematically as

$$F_4 = \frac{\cos^{-1}\left[\begin{matrix}(x_i^{curr} - x_i^{goal})\cdot(x_{gbest^{i-1}} - x_i^{goal}) \\ + (y_i^{curr} - y_i^{goal})\cdot(y_{gbest^{i-1}} - y_i^{goal})\end{matrix}\right]}{\sqrt{(x_i^{curr} - x_i^{goal})^2 + (y_i^{curr} - y_i^{goal})^2} \times \sqrt{(x_{gbest^{i-1}} - x_i^{goal})^2 + (y_{gbest^{i-1}} - y_i^{goal})^2}} \tag{9}$$

Now, the multi robot navigation problem can be represented as an optimization problem. The optimization problem contains an objective function that minimizes the Euclidean distance between the current location of each robot with its respective goal location and constraint based on avoiding collision with obstacles/teammates on its path. The constraints have been modeled by three types of penalty function. The first penalty function is used to avoid a collision of mobile robots with obstacles or teammates, whereas the second penalty is used to avoid collision between a mobile robot and dynamic obstacles, and the third penalty is considered for the smoothness of the path. Thus, the overall objective (or fitness) functions are obtained by the weighted sum of four objective functions such as:

$$F = \lambda_1 F_1 + \lambda_2 F_2 + \lambda_3 F_3 + \lambda_4 F_4 \tag{10}$$

where λ_1, λ_2, λ_3 and λ_4 are the weights of the shortest path, static obstacles, dynamic obstacles and smoothness of the path respectively. These weights are adjusted in the simulation and Khepera II robot, with preeminent values found $\lambda_1 = 1$, $\lambda_2 = 1$, $\lambda_3 = 0.25$ and $\lambda_4 = 0.25$. So the optimized path is obtained by minimizing the fitness function in Eq. (10) with assigned weights of each criterion.

3. Classical Q-learning(CQL) and its limitation

Classical Q-learning (CQL) is one of the reinforcement learning in which the agent performs an action through state transition in the environment and receives a reward or penalty by executing the action to reach the goal state. The main aim of the agent is to learn the control strategy to select an action from the possible set action at the particular state to maximize the sum of the rewards from the specified start state to goal state through the state transition process. Let $S_0, S_1, S_2, \ldots, S_n$ be n possible states of an agent, and $a_0, a_1, a_2, \ldots, a_m$ be the set of m possible actions for each state. An agent selects an action from a set of possible n action in each state and receives the specific reward that the agent acquires, which is known as immediate reward. Consider the immediate reward and total reward that the agent A acquires by executing an action a_j at state S_i is $r(S_i, a_j)$ and $Q(S_i, a_j)$ respectively. A policy is used to select the next state from its current state of an agent, and this policy is used to maximize the cumulative reward that the agent could be acquired during the transition process of states from the next state to the goal state. For example, let the agent be in state S_i and is expected to select the next best state. Then the Q-value at state S_i due to action of a_j is given in Eq. (11).

$$Q(S_i, a_j) = r(S_i, a_j) + \gamma \underset{a^l}{Max} Q(\delta(S_i, a_j), a^l) \tag{11}$$

where $\delta(S_i, a_j)$ denotes the next state due to selection of action a_j at state S_i. Let the next state selected be S_k. Then $Q(\delta(S_i, a_j), a^l) = Q(S_k, a^l)$. Consequently selection of a^l that maximizes $Q(S_i, a_j)$ is an interesting problem. In the above Q-learning, if there are n number of states and m number of action in each state, then Q-table will be constructed with an $n \times m$ dimension. The agent moves from the present state to the next state based on the Q-value stored in the Q-table. When the agent tries to move from the present state at that time, it fetches the Q-table from the memory to find the action storing maximum Q-value, and this action will be selected to get the appropriate next state. As a result, every time, it accesses the Q-table to get to the next state, which is a time-consuming process to select the next state and takes more space to create the Q-table with high dimension. The number of comparisons required to get the action storing the maximum Q-value is (m–1). To update Q-value for m number of state with n number of action requires n(m–1) comparison. It takes $n \times m$ memory space to store the Q-table and to keep track of dynamic Q-value; it takes an addition $n \times m$ memory space. So the total memory required is 2 nm. If the state–action pair increases, then it is difficult to represent the Q-table as well as identify the termination of classical Q-learning. The main limitation of the classical Q-learning is to know the Q-value at a state S_k for all possible action a^l. So the above limitation is overcome by remodeling the classical Q-learning.

In the proposed improved Q-learning, Q-table is constructed by storing the Q-value for best action as one field and another field is used for storing lock variable of the particular state. In this process, we are able to reduce the space requirement for storing the Q-table.

4. Improved Q-learning

To overcome the pitfalls of the classical Q-learning mentioned in the previous section, we have constructed the Q-table containing the best Q-value for best action as one field, and the other field as lock flag indicates that the Q-value of that state will be unchanged. This Q-table is constructed based on the four properties that the improved Q-learning has proposed. We make the agent learn the environment with the proposed new algorithm by generating the Q-value for each state and allowed it to learn until all the lock variables are set.

Consider S_k to be any state. Let the distance between the goal state and the next possible states of S_k be known. Let $S \in \{S_a, S_b, S_c, S_d\}$ be the next possible state of S_k and G be the goal. Let the city block distance between S_a, S_b, S_c, S_d and G be d_{aG}, d_{bG}, d_{cG} and d_{dG}, respectively, and the distance in order be $d_{bG} < d_{aG} < d_{cG} < d_{dG}$. Then the agent should select the next state S_b from its current state S_k. If the Q-value of the state S_b is known, we can evaluate the Q-value of state S_k by the following approach.

$$\begin{aligned} Q(S_k, a^l) &= r(S_k, a^l) + \gamma \underset{a^{ll}}{Max} Q(\delta(S_k, a^l), a^{ll}) \\ &= 0 + \gamma \underset{a^{ll}}{Max} Q(\delta(S_k, a^l), a^{ll}) \end{aligned} \tag{12}$$

Now $\delta(S_k, a^l) = S_a | S_b | S_c | S_d$, where | denotes OR operator.
Therefore,

$$\begin{aligned} &\underset{a^{ll}}{Max} Q(\delta(S_k, a^l), a^{ll}) \\ &= \underset{a^{ll}}{Max} Q\{S_a | S_b | S_c | S_d, a^{ll}\} \\ &= Q(S_b, a^{ll}). \quad (\because d_{bG} < d_{aG} < d_{cG} < d_{dG}) \end{aligned} \tag{13}$$

Combining (2) and (3) we have:

$$Q(S_k, a^l) = 0 + \gamma Q(S_b, a^{ll}).$$

Thus, if the next state having the shortage distance with the goal is known, and the Q-value of this state is also known, then the Q-value of the current state is simply $\gamma \times$ Q-value of the next state.

Let S_p, S_n and S_G be the present, next and the goal states respectively. Let Q_p and Q_n be the Q-value at the present and the next states S_p and S_n respectively. Let d_{xy} be the city block distance between the states S_x and S_y. We use a Boolean variable lock: L_x to indicate that the Q_x value of a state is fixed permanently. We set lock $L_n = 1$ if the Q-value of the state n is fixed, and won't change further after L_n is set to 1. The lock variable for all states except the goal will be initialized zero in our proposed Q-learning algorithm. We observe four interesting properties as indicated below.

Property 1: If $L_n = 1$ and $d_{pG} < d_{nG}$, then $Q_p = \gamma \times Q_n$ and set $L_p = 1$.

Proof. Let the neighborhood state of S_p be $S \in \{S_a, S_b, S_c, S_n\}$, and the agent selects S_n as the next state as $d_{nG} < d_{xG}$ for $x \in \{a, b, c, n\}$.

Now,

$$\begin{aligned} Q_p &= Q(S_p, a) \\ &= r(S_p, a) + \gamma \underset{a^l}{Max} Q(\delta(S_p, a), a^l) = 0 + \gamma \underset{a^l}{Max} Q(S_a | S_b | S_c | S_n, a^l) \\ &= \gamma \times Q(S_n, a^l) (\because d_{nG} \le d_{xG} \text{ for } x \in \{a, b, c, n\}) \\ &= \gamma \times Q_n. \end{aligned}$$

Since $L_n = 1$ and $d_{pG} > d_{nG}$, $\therefore Q_p < Q_n$, and thus $Q_p = \gamma \times Q_n$ for $0 < \gamma < 1$ is the largest possible value of Q_p, and Q_p should not be updated further. So $L_p = 1$ is set.

Property 2: If $L_n = 0$ and $d_{pG} > d_{nG}$, then $Q_p = \text{Max}(Q_p, \gamma \times Q_n)$.

Proof. Let the neighborhood state of S_p be $S \in \{S_a, S_b, S_c, S_n\}$, and the agent selects S_n as the next state as $d_{nG} \le d_{xG}$ for $x \in \{a, b, c, n\}$.

Now,

$$\begin{aligned} Q_p &= Q(S_p, a) \\ &= r(S_p, a) + \gamma \underset{a^l}{Max} Q(\delta(S_p, a), a^l) \\ &= 0 + \gamma \underset{a^l}{Max} Q(S_a | S_b | S_c | S_n, a^l) \\ &= \gamma \times Q(S_n, a^l) (\because d_{nG} \le d_{xG} \text{ for } x \in \{a, b, c, n\}) \\ &= \gamma \times Q_n. \end{aligned} \tag{14}$$

Since $d_{pG} > d_{nG}$ and $Q_p < Q_n$, $Q_p = \gamma \times Q_n$ is logically satisfied. However, as $L_n = 0$, Q_n is not yet stable at the current iteration. So Q_p is also not stable yet. Let t be the current iteration. Then

$$\left.\begin{aligned} Q_p(t) &= \gamma \times Q_n(t) \quad \text{if } Q_p(t-1) \le \gamma \times Q_n(t). \\ &= Q_p(t-1), \qquad\qquad \text{otherwise.} \end{aligned}\right\} \tag{15}$$

The two alternatives in (15) can be put together as in (16).

$$Q_p(t) = Max(Q_p(t-1), \gamma Q_n(t)) \tag{16}$$

Ignoring the notion of time t from (6), we obtain $Q_p(t) = Max(Q_p, \gamma \times Q_n)$.

Property 3: If $L_p = 1$ and $d_{nG} > d_{pG}$ then $Q_n = \gamma \times Q_p$ and set $L_n = 1$.

Proof. Let the neighborhood of the next state S_n be $S \in \{S_a, S_b, S_c, S_p\}$. Since $d_{pG} \le d_{xG}$ for $x \in \{a, b, c, p\}$, the agent will select the state S_p during its transition form S_n.

Now, $Q_n = Q(S_n, a)$

$$\begin{aligned} &= r(S_n, a) + \gamma \underset{a'}{Max} Q(S_a|S_b|S_c|S_p, a') \\ &= 0 + \gamma \underset{a'}{Max} Q(S_a|S_b|S_c|S_p, a') \\ &= \gamma \times Q(S_p, a') \quad (\because d_{pG} \le d_{xG}, \forall x). \\ &= \gamma \times Q_p. \end{aligned}$$

Since $d_{nG} > d_{pG}$, $Q_n < Q_p$, $Q_n = \gamma \times Q_p$ is logically acceptable for $0 < \gamma < 1$. Further, as $L_p = 1$, and S_n is the nearest state to S_p with respect to the given distance metric, L_n is set to 1.

Property 4: *If $L_p = 0$* and $d_{nG} > d_{pG}$, then $Q_n = \text{Max}(Q_n, \gamma \times Q_p)$.

Proof. Let the neighborhood of state S_n be $S \in \{S_a, S_b, S_c, S_p\}$. Let $d_{pG} > d_{xG}$ for $x \in \{a, b, c, p\}$. Then the agent will select S_p during its transition from S_n,

Now, $Q_n = Q(S_n, a)$

$$\begin{aligned} &= r(S_n, a) + \gamma \underset{a'}{Max} Q(\delta(S_n, a), a') \\ &= r(S_n, a) + \gamma \underset{a'}{Max} Q(S_a|S_b|S_c|S_p, a') \\ &= 0 + \gamma \times Q(S_p, a') \quad (\because d_{pG} \le d_{xG}, \forall x) \\ &= \gamma \times Q_p. \end{aligned}$$

Since $d_{nG} > d_{pG}, \therefore Q_n < Q_p$, $Q_n = \gamma \times Q_p$ for $0 < \gamma < 1$ is logically satisfactory. Now, as $L_p = 0$, Q_p is not fixed until this iteration; so Q_n is also not fixed in the current iteration. Now, adding the notion of iteration t in Q_n and Q_p, we have:

$$\left.\begin{aligned} Q_n(t) &= \gamma \times Q_p(t), \quad \text{if } Q_n(t-1) < \gamma \times Q_p(t), \\ &= Q_n(t-1), \qquad\qquad \text{otherwise.} \end{aligned}\right\} \tag{17}$$

Combining the expressions under (17) by a single expression, we write:

$$Q_n(t) = Max(Q_n(t-1), \gamma \times Q_p(t)) \tag{18}$$

Now, eliminating t from both sides of (18), we obtain,

$$Q_n = Max(Q_n, \gamma \times Q_p) \tag{19}$$

5. Improved Q-learning algorithm

In the proposed new Q-learning algorithm, only two fields for each grid are required, one is used to store Q-value and the other is used to store the lock variable. In the new algorithm every state stored only the Q-value and not the best action to reach the goal state. The best path between the current state to the goal is given by the city–block distance; therefore, there will be more than one best path to reach the goal. As a result selection of the best action is deferred until the path planning, and by doing so the energy required by the robot can be minimized. The newly proposed algorithm has four steps. Initialization of Q-table is done in step 1, and in step 2 the value of γ and the starting state of the robot are initialized. Q-table is updating only after the robot reaches the goal state for the first time. Thus the robot is allowed to come to the goal state without updating Q-table. This is done in step 3. The updating of the Q-table is done in step 4 using the properties introduced in section 3.

Pseudo Code for Improved Q-learning

$Algorithm$ Initializa tion Qtable $(x_i^{curr}, y_i^{curr}, x_i^{goal}, y_i^{goal})$

//convert position to state by $S_p = (x_i^{curr} - 1) \times rowsize + y_i^{curr}$

//$S_G = (x_i^{goal} - 1) \times rowsize + y_i^{goal}$

{

For all S_i, i = 1 to n except $S_i = S_G$

{set $L_i = 0; Q_i = 0;$

}

L_G (for goal S_G) = 1;

Q_G (for goal S_G) = 100;

2. Assign γ in (0, 1) and initial state= S_p;

3. Repeat

{

Select a_i from $A = \{a_1, a_2, ..., a_m\}$;

} Until $S_P = S_G$;

}

Algorithm Q_update(X)

Repeat

{

a) Select a_i from $A = \{a_1, a_2, ..., a_m\}$;

b) Determine d_{nG} and d_{pG};

If $(d_{nG} < d_{pG})$

Then if $L_n = 1$

Then if $L_p = 0$

Then $\{Q_p = \gamma \times Q_n; L_p = 1;\}$

Else if $(Q_p < \gamma \times Q_n)$

Then $\{Q_p = \gamma \times Q_n\}$;

Else if $L_p = 1$

Then if $L_n = 0$

Then $\{Q_n = \gamma \times Q_p; L_n = 1$

Else if $(Q_n < \gamma \times Q_p)$

Then $\{Q_n = \gamma \times Q_p\}$

} Un til $L_i = 1$ for all i; ;

5.1. Space-complexity

In classical Q-learning, if there are n states and m action per state, then the Q-table will be of $(m \times n)$ dimension. In the improved Q-learning, for each state 2 storages are required, one for storing Q-value and other for storing value of the lock variable of a particular state. Thus for n number of states, we require a Q-table of $(2 \times n)$ dimension. The saving in memory in the present context with respect to classical Q thus is given by mn–2n = n(m–2).

5.2. Time-complexity

In classical Q-learning, the updating of Q-values in a given state requires determining the largest Q-value, in that cell for all possible actions. Thus if there are m possible actions at a given state, maximization of m possible Q-values requires m–1 comparison. Consequently, if we have n states, the updating of Q values of the entire Q table by classical method requires n(m–1) comparisons. Unlike the classical case, here we do not require any such comparison to evaluate the Q values at a state S_p from the next state S_n. But we need to know whether state n is locked, i.e., Q-value of S_n is permanent and stable. Thus if we have n number of states, we require n number of comparison. Consequently, we save n(m–1) – n = nm–2n = n(m–2).

6. Particle swarm optimization (PSO)

6.1. Classical PSO (CPSO)

The CPSO is a stochastic population-based, bio-inspired evolutionary optimization algorithm, which was originally introduced by Kennedy and Eberhart (1995), which utilizes swarm intelligence to achieve the goal of optimization. It is based on intelligent collective behavior of schools of fish or bird flocks. In the classical PSO algorithm, each member of the population is known as a particle in a D-dimensional search and a set of particles is called swarm. The velocity parameter of the CPSO is dynamically updated by the particles' own experience and flying experience of its accompaniment. The members of the entire population share the information among individuals to change each particle position to find the best position in the search space. The advantage of the CPSO over other optimization algorithm is easy to implement and there are few parameters to be adjusted.

Let N be the population size. In each generation k, the velocity and position of the particles are updated using Eq. (1).

$$\begin{aligned} V_{id}(t+1) &= V_{id}(t) + C_1 \cdot \varphi_1 \cdot (pbest_{id}(t) - x_{id}(t)) \\ &\quad + C_2 \cdot \varphi_2 \cdot (gbest_d(t) - x_{id}(t)) \\ x_{id}(t+1) &= x_{id}(t) + V_{id}(t+1) \end{aligned} \tag{20}$$

where $x_i = (x_{i1}, x_{i2}, \ldots\ldots x_{iD})$ represents the current position vector of the particle $i\,(1 \le i \le N)$ in a D-dimensional search space, $V_i = (V_{i1}, V_{i2}, \ldots\ldots V_{iD})$ represents the velocity of i^{th} particle, $C_1 (C_1 \neq 0)$ and $C_2 (C_2 \neq 0)$ are the acceleration constants, and φ_1 and φ_2 are two random numbers in the range [0, 1]. x_{pbest} is the previous best position of the i^{th} particle in generation k, x_{gbest} is the previous global best position among all the particles in generation k. If $C_1 = 0$, then PSO algorithm is converted to social-only model. Similarly, if $C_2 = 0$, then it becomes a cognition-only model.

6.2. Improved particle swarm optimization

To bring a balance between the exploration and exploitation characteristics of PSO, Shi and Eberhated proposed a PSO on inertia weight in which the velocity of each particle is updated, and claimed that the larger the value of the inertia weight will be provides a global search, while the smaller the value will be provides local search. So it needs to change the inertia weight dynamically to adjust the search capability dynamically. Therefore, there are several proposals to modify the PSO algorithm by changing the inertia weight value in adaptive manner in each iteration. In this paper, we have improved PSO (IPSO) in terms of adaptive weight adjustment and acceleration coefficients to increase the convergence rate to optimum value in PSO; the classical PSO equation is modified according to the following form.

$$\begin{aligned} V_{id}(t+1) &= w_i V_{id}(t) + C_1 \cdot \varphi_1 \cdot (pbest_{id}(t) - x_{id}(t)) \\ &\quad + C_2 \cdot \varphi_2 \cdot (gbest_d(t) - x_{id}(t)) \\ x_{id}(t+1) &= x_{id}(t) + V_{id}(t+1) \end{aligned} \tag{21}$$

The local best value in IPSO can be computed as:

$$pbest_{id}(t+1) = \begin{cases} pbest_{id}(t), & if\ O_{bj}(x_{id}(t)) \succ O_{bj}(pbest_{id}(t)) \\ x_{id}(t+1), & if\ O_{bj}(x_{id}(t+1)) \prec O_{bj}(pbest_{id}(t)) \end{cases} \tag{22}$$

where f stands for the fitness function of the moving particles and the global best position is obtained as:

$$gbest_d(t) = \min\{O_{bj}(pbest_{1d}(t)), O_{bj}(pbest_{2d}(t)), \ldots\ldots O_{bj}(pbest_{ND}(t))\} \tag{23}$$

The convergence rate of the PSO has been improved by fine tuning of its parameter with the help of several techniques. These techniques usually change the PSO update equations without altering the inherent structure of the algorithm. The velocity during the previous time step is scale it by a scale factor inertia weight (w) to update a new velocity every time the particle moves the search space. Empirical experiments have been performed in the past with an inertia weight decreasing linearly from 0.9 (w_{max}) to 0.4 (w_{min}) as per the following:

$$w_i = w_{\min} + (w_{\max} - w_{\min}) \frac{rank_i}{K} \tag{24}$$

where K is the number of particles, and $rank_i$ is the position of the i^{th} particle, when particles are ordered based on their best fitness value. Similarly, the fixed value is set for the acceleration coefficients (conventionally fixed at 2.0). The large value of the social component C_2 in comparison with cognitive component C_1 leads particles to local optimum prematurely, and relatively high values of cognitive components result to wander the particles around the search space. The quality of the solution quality is improved by modifying cognitive and social coefficient term in such a way that the cognitive component is reduced and social component is increased as generation proceeds. The modification of the coefficients are made (for k^{th} generation) using Eq. (23) and Eq. (24).

$$C_1 = C_{1i} - \left(\frac{C_{1i} - C_{1f}}{Max\_Iter} \right) t \tag{25}$$

$$C_2 = C_{2i} + \left(\frac{C_{2f} - C_{2i}}{Max\_Iter} \right) t \tag{26}$$

where C_{1i}, C_{1f}, C_{2i} and C_{2f} are initial and final values of cognitive and social components acceleration factors, respectively, and *Max_Iter* is the maximum number of allowable iterations.

7. Differential evolutionary algorithm

Differential evolution (DE) algorithm is proposed by Storn and Price in 1995 as a new alternative form of evolutionary algorithm for global optimization [12,15]. It uses selection, mutation and recombination as a special kind of differential operator to create new offspring from parent genomes or chromosomes instead of using

a classical crossover or mutation for the next generation. DE is specially used for a global search in a D-dimension search space. It is a population-based algorithm-like genetic algorithm (GA) with similar operators, but the main difference is that DE relies on mutation operation, whereas genetic algorithm relies on crossover. Mutation operation is used as a search mechanism and the selection operation forwarded the search toward the prospective regions of the search space in DE algorithm.

DE begins with a randomly generated population for D-dimensional search variable vectors at a time $t = 0$. In the subsequent generations, discrete time steps can be represented as $t = 0, 1.... t, t+1$, etc. Since the vectors are likely to be changed over different generations, the following notations were used for representing the i^{th} vector of the population at the current generation t as:

$$\vec{X}_i(t) = [x_{i1}, x_{i2}, x_{i3}.... x_{iD}] \tag{27}$$

This vector is referred as chromosomes. For the better result, the value of the each parameter must change within a certain range in which each variable lies. All the parameters and independent variables of the problem are initialized within their feasible numerical range during the start of the DE run at t = 0. As result, if m^{th} parameter of a given problem has its lower and upper bound as $x_m^{\min}$ and $x_m^{\max}$, then we can initialize the m^{th} component of the i^{th} population as

$$x_{i,m}(0) = x_m^{\min} + \beta \cdot (x_m^{\max} - x_m^{\min}) \tag{28}$$

where β is a random number with [0,1]. A donor vector $V_i(t)$ is created by changing each population member $X_i(t)$ in each generation. Various types of DE schemes have been used to create the donor vector. Here, we have used DE/rand/1 as the mutation strategy. In this strategy, to create $V_i(t)$ for each i^{th} member, other three parameter vectors (say z_1, z_2 and z_3) are chosen randomly from the current population. Next, calculate the difference $x_{z_2} - x_{z_3}$, scale it by a scalar Factor F and add the third vector x_{z_1} as the third term to obtain the donor vector $V_i(t)$. We can mathematically expressed the m^{th} component for each donor vector as

$$v_{i,m}(t+1) = x_{z_1,m}(t) + F \cdot (x_{z_2,m}(t) - x_{z_3,m}(t)) \tag{29}$$

Next, crossover method comes to play in the DE scheme to increase the potential diversity of the population; DE uses two types of crossover: procedure one is exponential and the other is binomial. In this scheme the components of the donor vector exchange with the target vector $\vec{X}_i(t)$. In the exponential crossover procedure, choose an integer n from [0, D–1] in random manner, and this integer acts as starting point in the target vector to cross over or exchange with the components of the donor vector. Again generate another integer L from the interval [0, D–1] to denote the number of components the donor vector contributes to the target vector. After generating the n and L, the trial vector is as follows:

$\vec{U}_i(t) = [u_{i,1}(t), u_{i,2}(t)..... u_{i,D}(t)]$ is created with

$$\begin{aligned} u_{i,m}(t) &= v_{i,m}(t) \quad for \quad m = \langle n \rangle_D, \langle n+1 \rangle_D \ldots \langle n-L+1 \rangle_D \\ &= x_{i,m}(t) \end{aligned} \tag{30}$$

where the angular brackets $< >_D$ represent a modular function with module D. According to the following pseudo code the integer L is drawn from [1,D]

L = 0
Repeat
{
L = L+1
}until ((rand(0,1) < CR) AND (L < D))

Here, CR is the crossover constant and acts as control parameter for DE same as F. The effect probability $(L > P) = (CR)^{P-1}$. A new set of n and L is created as above for each donor vector V. In a "binomial" crossover, the crossover game is played on each variables of the D when the CR value is within 0 and 1. The outline of the scheme is presented as follows:

$$u_{i,m}(t) = \begin{matrix} v_{i,m}(t) & if\ rand(0,1) \prec CR \\ x_{i,m} & otherwise \end{matrix} \Bigg\}$$

In this way, an offspring vector $\vec{U}_i(t)$ is created for each trial vector $\vec{X}_i(t)$. To keep the population size constant over subsequent generations, selection procedure is called in the next step of the algorithm to determine the survival of the target vector or trial vector in the next generation at time t = t+1. This is the Darwinian principle of "survival of the fittest" used for selection process in DE, which can be outlined as

$$X_i(t+1) = \begin{matrix} U_i(t) & if\ f(U_i(t)) \le f(X_i(t)) \\ X_i(t) & if\ f(X_i(t) \prec f(U_i(t)) \end{matrix} \Bigg\} \tag{31}$$

where f() is the function to be minimized. If the trial vector generates a better fitness value, then the trial vector replaces its target in the next generation; otherwise the target remains unchanged in the population. Hence the population either gets better (with respect to the fitness values) or remains the same but never deteriorates.

8. The IPSO-DV algorithm

The concept of particle swarms emerged from a simulation of the collective behavior of social creatures and gradually evolved into a powerful global optimization technique, now well-known as the particle swarm optimization (PSO). PSO is arguably one of the most popular nature-inspired algorithms for real parameter optimization at present. The classical PSO model does not ensure convergence to an optimal solution as well as suffers from its dependency on external parameters, like acceleration parameters and inertia weight. Due to the above flaw its efficiency is comparatively poor; a multitude of measures has been taken to improve the performance of PSO, as presented in the previous section, and is called improved particle swarm optimization (IPSO). In most of the cases the convergence is premature; when most of the particles do not change their positions in the swarm in the successive stages, the global optimum cannot be discovered. This situation occurs due to a small value of the inertia weight or constriction coefficient. In Eq. (2), we found that if V_i^k is small and in addition to very small value of $|x_{pbest}^k - x_i^k|$ and $|x_{gbest}^k - x_i^k|$, then V_i^k cannot attain a large value in the upcoming iterations. That indicates a loss of exploration power. This can happen even at the early stage of the search process, when the particle is in the global best causing $|x_{pbest}^k - x_i^k|$ and $|x_{gbest}^k - x_i^k|$to be zero and gets damped at quickly with the ratio w. It also suffers from loss of diversity, when x_{pbest}^k and x_{gbest}^k are close enough [30–32]. Due to the encounter of the above problems, differential operator is used to adjust the velocity of the IPSO to remove premature convergence. A new scheme is proposed to adjust the velocity of the IPSO with a vector differential operator inherited from the differential evolutionary algorithm. The novel dimensional mean-based perturbation strategy, a simple aging guideline, and a set of nonlinearly time-varying acceleration coefficients to accomplish a better tradeoff between explorative and exploitative capabilities, and thus to avoid premature convergence on multimodal fitness landscapes. The aging guideline is used to introduce fresh solutions in the swarm when particles show no further improvement. In this work the particle velocities are perturbed by the weighted difference of the position vectors of any two distinct particles chosen from their previous and next neighbor of the swarm. This differential velocity term is inherited from the DE mutation scheme, and hence this algorithm is named IPSO-DV (improved particle swarm with

Pseudo-code

Input: Randomly initialized position $\vec{X}_i(0)$ and velocity $\vec{V}_i(0)$ of particles

Output: Approximation position of global optima $\vec{X}^*$

Begin

Initialize population size(PN) ,iteration(t=0),mutation parameter(F) and cross over probability(CR) ;
While the stopping condition not satisfied **do**

Begin

For i = 1 to PN

Begin
Mutation :
Generate a donor vector $V_i(t)$ by DE/rand/1
Crossover :
Generate a trial vector $\vec{U}_i(t)$ based on binomial crossover operation
Selection:
Evaluate $\vec{U}_i(t)$
If $f(U_i(t)) \le f(X_i(t))$ then
$X_i(t+1) = U_i(t),\ f(X_i(t+1)) = f(U_i(t))$
Else
$X_i(t+1) = X_i(t),\ f(X_i(t+1) = f(X_i(t))$
End If;
Increase the iteration t=t+1

End for

End while

End

differentially perturbed velocity). The greedy selection scheme of DE is adopted by the principle of survival of the fittest.

IPSO-DV uses a differential operator in the velocity update equation of the IPSO. The operator is introduced on the position vectors of two neighbor chosen particles from the population, and these are not in their best positions. Further, a particle is shifted to a new position only if the new position generates a better fitness value, in IPSO scheme, i.e. the selection process has incorporated into the swarm dynamics. In the proposed algorithm, each particle i in the swarm has considered the other two distinct particles, say j and $k\,(i \neq j \neq k)$,Where $j = i-1$ *and* $k = i+1$ are selected from previous and next neighbor of the particle i in the swarm. The difference in their positional coordinates is taken as a difference vector.

$$\vec{\delta} = \vec{X}_k - \vec{X}_j \tag{32}$$

Then, the dth velocity component $(1 \prec d \prec n)$ of the target particle i is updated as

$$V_{id}(t+1) = \begin{cases} \omega V_{id}(t) + \beta \cdot \delta_d + C_2 \cdot \phi_2 \cdot (P_{gd} - jX_{id}(t)), & \text{if } rand_d(0,1) \le CR \\ V_{id}(t), & \text{otherwise} \end{cases} \tag{33}$$

where CR is the crossover probability, δ_d is the dth component of the difference vector, and β is the scale factor in [0,1]. In essence the cognitive part of the velocity update formula in Eq. (1) is replaced with the vector differential operator to produce some additional exploration capability. Clearly, for $CR \le 1$ and it change with iteration is expressed in Eq. (14), some of the velocity components will retain their old values. Now, a new trial location Tr_i is created for the particle by adding the update velocity to the previous position X_i:

$$T\vec{r}_i = \vec{X}_i(t) + \vec{V}_i(t+1) \tag{34}$$

$$CR = \left(\frac{iteration\_value}{Maximum\_iteration\_value} \right)^n \tag{35}$$

where n is the non-linear index that tunes the crossover ration for the better convergence. The particle is placed in this new location only if the coordinates of the location yield a better fitness value. Thus if we are seeking the minimum of an n-dimensional function $f(\vec{X})$, then the target particle is relocated as follows:

$$\begin{aligned} &\vec{X}_i(t+1) = \vec{T}r_i \quad \text{if} \quad f(\vec{T}r_i) \le f(\vec{X}_i(t)) \\ &\vec{X}_i(t+1) = \vec{X}_i(t) \quad \text{otherwise} \end{aligned} \tag{36}$$

Therefore, every time its velocity changes, the particle either moves to a better position in the search space or sticks to its previous location. The current location of the particle is thus the best location it has ever found so far. If a particle does not change its position in a predefined number of iteration in the search space (i.e. stagnant at any point in the search space), then the particle is shifted to a new position by random mutation, and is mathematically explained below. This method helps to escape local minima and keeps the swarm moving.

$$\begin{aligned} &\text{if } \left(\vec{X}_i(t) = \vec{X}_i(t+1) = \ldots = \vec{X}_i(t+N)\right) \text{ and } f^*\left(\vec{X}_i(t+N)\right) \\ &\text{then} \\ &\text{for } j = 1 \text{ to } n \\ &\quad X_{ij}(t+N+1) = X_{\min} + rand_j(0,1) * (X_{\max} - X_{\min}) \end{aligned} \tag{37}$$

where f^* is the global minimum of the fitness function, N is the maximum number of iterations up to which stagnation can be

Procedure IPSO-DV (x_{curr-i} , y_{curr-i} , pvector)

Begin

Initialize population and algorithm parameters as well as problem parameters

Call Initialize Qtable(x_{curr-i} , y_{curr-i} , x_i^{goal} , y_i^{goal})

For iter=1 to maxiter

Begin

For i= 1 to number_ of_ particles

Evaluate fitness value of each particle;

Update the gbest ;

Call Q_update(pbest)

Call Q_update(gbest)

Call Q_update(X_i)

$if(Q(X_i) > Q(pbest_i))$

$pbest_i = X_i$

$if(Q(X_i) > Q(gbest))$

$gbest = X_i$

Select two neighbor particles j and $k(i \neq j \neq k)$

Construct the difference vector using Eq(32)

For d = 1 to number_ of_ dimensions

If $\text{rand}_d(0,1) \leq CR$ **then**

$V_{id}(t+1) = \omega V_{id}(t) + \beta.\delta_d + C_2.\phi_2.(gbest - jX_{id}(t))$

Else $V_{id}(t+1) = V_{id}(t)$

End if

End for

Create trial position $\vec{Tr}_i = \vec{X}_i(t) + \vec{V}_i(t+1)$

If ($f(\vec{Tr}_i) \leq f(\vec{X}_i(t))$ then $\vec{X}_i(t+1) = \vec{Tr}_i$

Else $\vec{X}_i(t+1) = \vec{X}_i(t)$

End if

End for

For i= 1 to no_of_particless

If X_i stagnates for N successive generations

For j= 1 to no_of_dimensions

$X_{ij}(t+1) = X_{\min} + rand_j(0,1) * (X_{\max} - X_{\min})$

End for

End if

End for

End for

Update

$x_{curr-i} = x_{curr-i} + v_i \cos\theta_i$

$y_{curr-i} = y_{curr-i} + v_i \sin\theta_i$

return

End

tolerated, and $(X_{\max}, X_{\min})$ is the permissible bounds of the search space.

The architecture of the learning algorithm QIPSO-DV is presented in Fig. 3. The whole learning process is completed in two steps, namely Q-value and IPSO-DV operation. The strategy for generating the Q-value of particles is described in section 4, and IPSO-DV algorithm is described in this section.

9. QIPSO-DV path planning algorithm

In newly proposed improved Q-learning, the best action at each state is selected dynamically. This helps in reducing the energy required by the agent while moving. In the planning algorithm the best action is selected by comparing the Q-value of the 8-neighbor states. If there are more than one state with equal Q-value, then the action that requires minimum energy for turning is selected.

In Fig. 4 the robot is at the center facing toward east. The next feasible states are the 8-neighbor states and the corresponding Q-values are given in each state. There are two states with maximum Q-value of 89. If the next state is located in the south, then the robot has to rotate toward right by 90°, and if the next state is located in the west, then the robot has to rotate by 180°. Therefore, the presumed next state will be the state in the south as the robot requires less energy to move to that state. During path planning the robot selects one out of several best actions at a given state. If an obstacle is present in the selected direction, then next optimal path is selected. If there is no action to be selected, then a signal is passed to the robot to backtrack one step.

10. Implementation of hybrid QPSO-DV and performance analysis

The path planning problem for multi-robot is carried out in a simulated environment. The simulation is conducted through programming in Matlab on a Pentium microprocessor. First, we designed an environment of 30×30 grids, and every grid is given a state number; the position (x, y) is converted into the state using Eq. (38). The robot is represented with 10 similar soft-bots of circular shape with different color codes, and each robot radius is 6 pixels. Prior to the start of the experiment, the initial location and goal location for the entire robot are assigned. The initial configuration of the world map for our experimental result is presented in Fig. 5 with five obstacles and 10 soft-bots; out of the 10 soft-bots, five have circular shape, with different colors representing the initial position of each robot, and the remaining five have rectangle shape, representing the goal position of each robot within the same color code. Initially, assigned same velocities for each robot and velocities changed for each robot through the algorithm in the subsequent iteration.

$$state\ no. = (x-1) \times rowsize + y \tag{38}$$

where x is the no of block in x direction, y is the no of block in y direction, and *rowsize* is the length of the row.

The experiment is conducted using central version of the algorithm using the fitness function (26) for deciding the next position of the robot. At the first instance, the following parameters are used in the simulation and Khepera II environment for QPSO-DV application in path planning of multi-robot: $w_{\min} = 0.4$, $w_{\max} = 0.9$, $T = 100$, $\beta = 0.8$, No_of_particles = 50, $\gamma = 0.8$.

The path planning of multi-robot is tuned by the hybrid QIPSO-DV; in this hybridization, Q-learning is used to update the Q-value of each state in the grid map using the proposed properties in section 3, which is embedded in the IPSO-DV algorithm to decide the pbest and gbest value in the swarm population for IPSO-DV based on comparing the Q-value of the particle position with the Q-value of generated pbest and gbest from PSO. The experiment is conducted in the MATLAB 8.0 on Pentium 3.0GHz processor with 4.0 GB of memory, and MATLAB code is executed with the swarm size of 50 using four different algorithms, namely QIPSO-DV, IPSO-DV, PSO

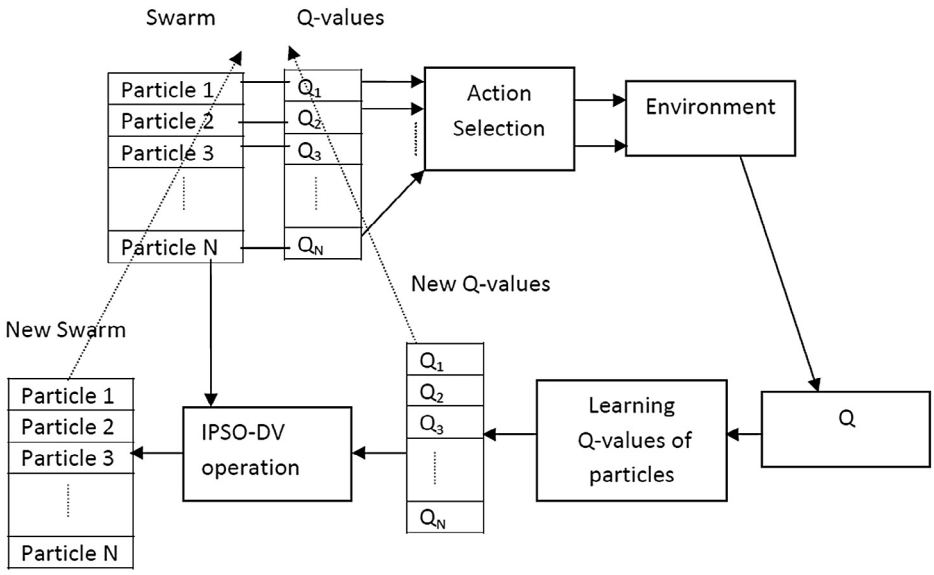


Fig. 3. Architecture of the QIPSO-DV.

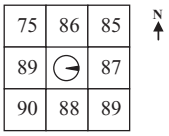


Fig. 4. Q-value of each state and robot at center.

and CQL, and each algorithm runs 30 times to obtain the performance of simulation in terms of average total trajectory path distance (ATTPD), average total trajectory path deviation (ATPD), average untraveled total trajectory target distance (AUTTD), end time of each robot, total travel time of all robots and mean squared error of robot/target distribution ratio. Simulation-based final configuration of the world map of the experiment is presented in Fig. 6, with five robots and five obstacles. The final stage of world map, where all the robots reached their predefined goal after execution of the (i) QIPSO-DV, (ii) IPSO-DV, (iii) CQL and (iv) PSO with 21, 23, 25 and 27 number

Pseudo Code for path planning

Input:(x_i^{curr}, y_i^{curr}), (x_i^{goal}, y_i^{goal}) and v_i^{curr} are the initial position, goal position and velocity for n robots respectively, $1 \le i \le n$ and ε is the threshold value.

Output: Optimal Trajectory of path OTP_i is generated for robot R_i from(x_i^{curr}, y_i^{curr}) to (x_i^{goal}, y_i^{goal})

Begin

For *i = 1 to n*

$x_{curr\_i} \leftarrow x_i^{curr}$; $y_{curr\_i} \leftarrow y_i^{curr}$;

End for

For each robot $i = 1$ to n

While ($Curr\_i \neq G_i$) // $_{curr\_i}$ = (x_{curr_i}, y_{curr_i}), G_i = (x_i^{goal}, y_i^{goal})//

Call IPSO-DV (x_{curr_i}, y_{curr_i} ,pvector);

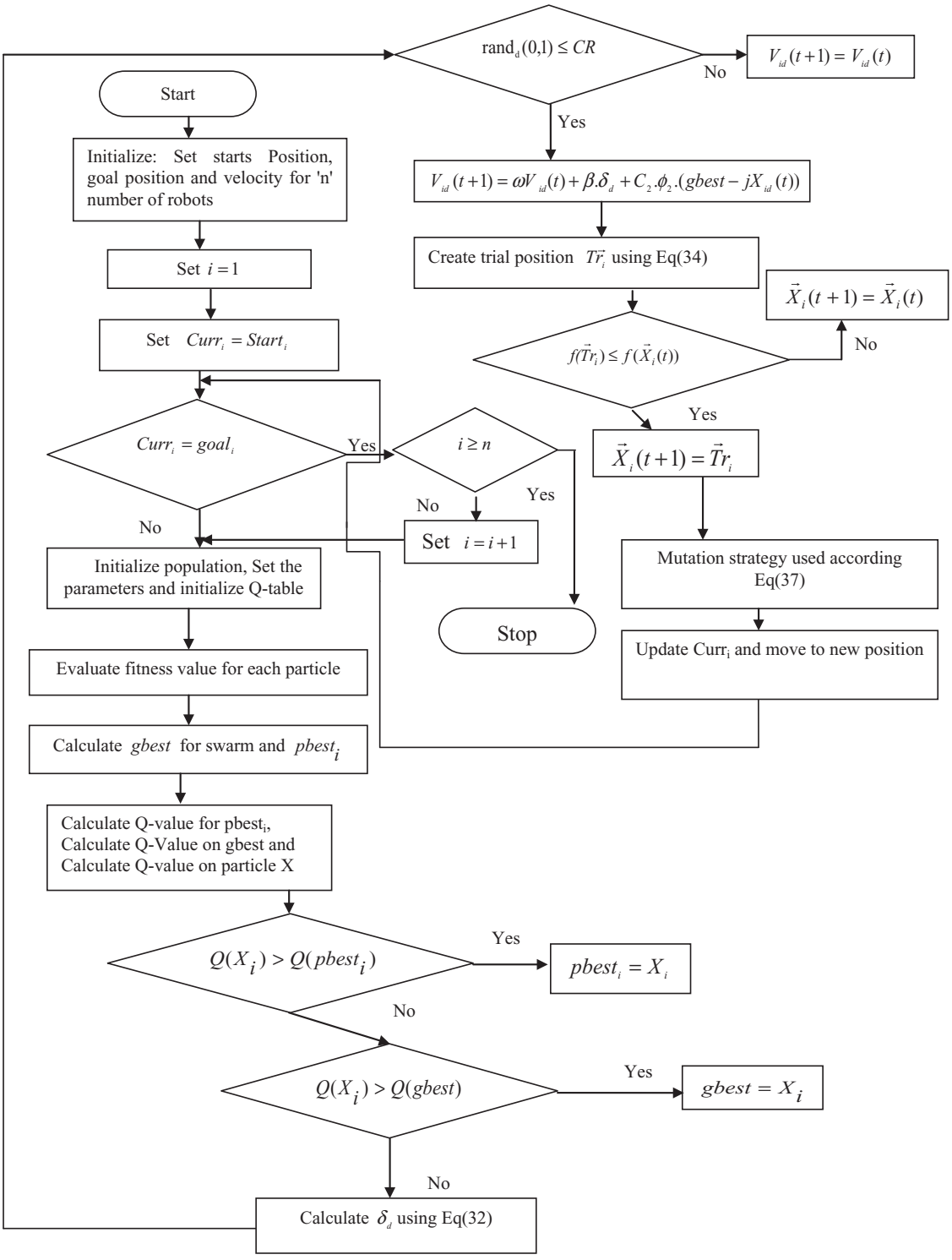
// pvector is the position vector denotes updated current position of the i-th robot //

Moveto (x_{curr_i}, y_{curr_i});

End for

End for

End



of steps, respectively, is presented in Fig. 6(i)–(iv), and the performance of simulation presented in Fig. 6 shows that QIPSO-DV takes less steps in comparison to other algorithms to reach the designation. Again, the performance of the simulation has been analyzed in terms of the number of turns required to reach the goal, and it is presented in Fig. 7. Fig. 7 shows that the QIPSO-DV takes 13 number of turns for all robots to reach the goal, whereas for other algorithms it takes more number of turns to reach the destination. From these two analyses, it shows that QIPSO-DV is more efficient and effective with respect to the other three algorithms in terms of the number of turn and the number of steps required to reach the destination.

10.1. Average total trajectory path deviation (ATTPD)

Consider the robot R_k is generated a collision free trajectory of path from specified initial position S_k to goal position G_k in the j^{th} number of iteration of the program is TP_{kj}. If $TP_{k1}, TP_{k2}, \ldots, TP_{kj}$ are the collision free trajectory of paths produced in the j^{th} number of iteration of the program. The average total trajectory path

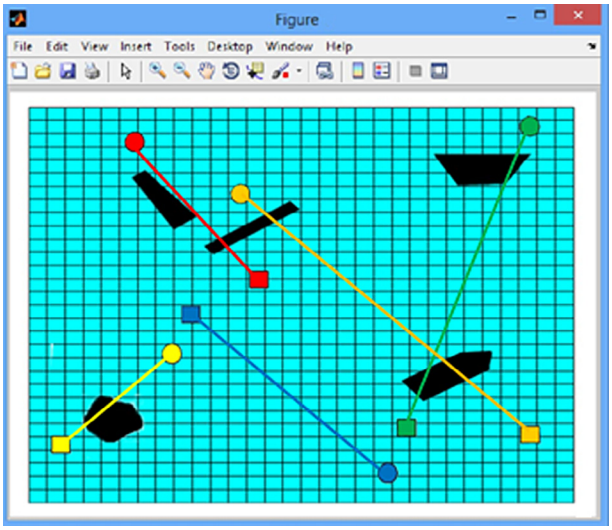


Fig. 5. Initial environment for execution of QIPSO-DV, IPSO-DV, CQL and PSO.

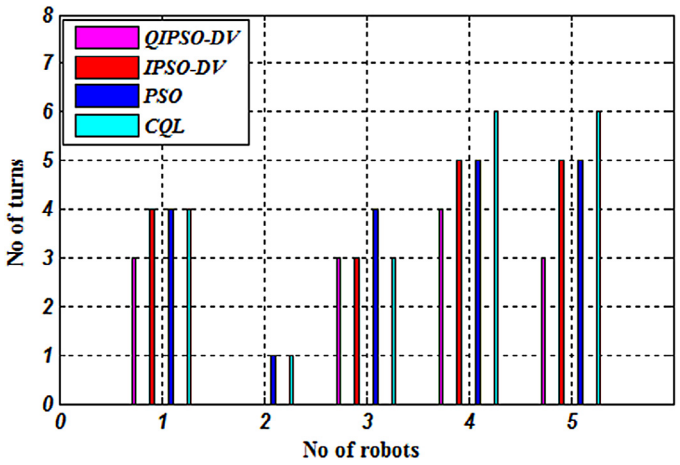


Fig. 7. Number of turns vs number of turns for each algorithm.

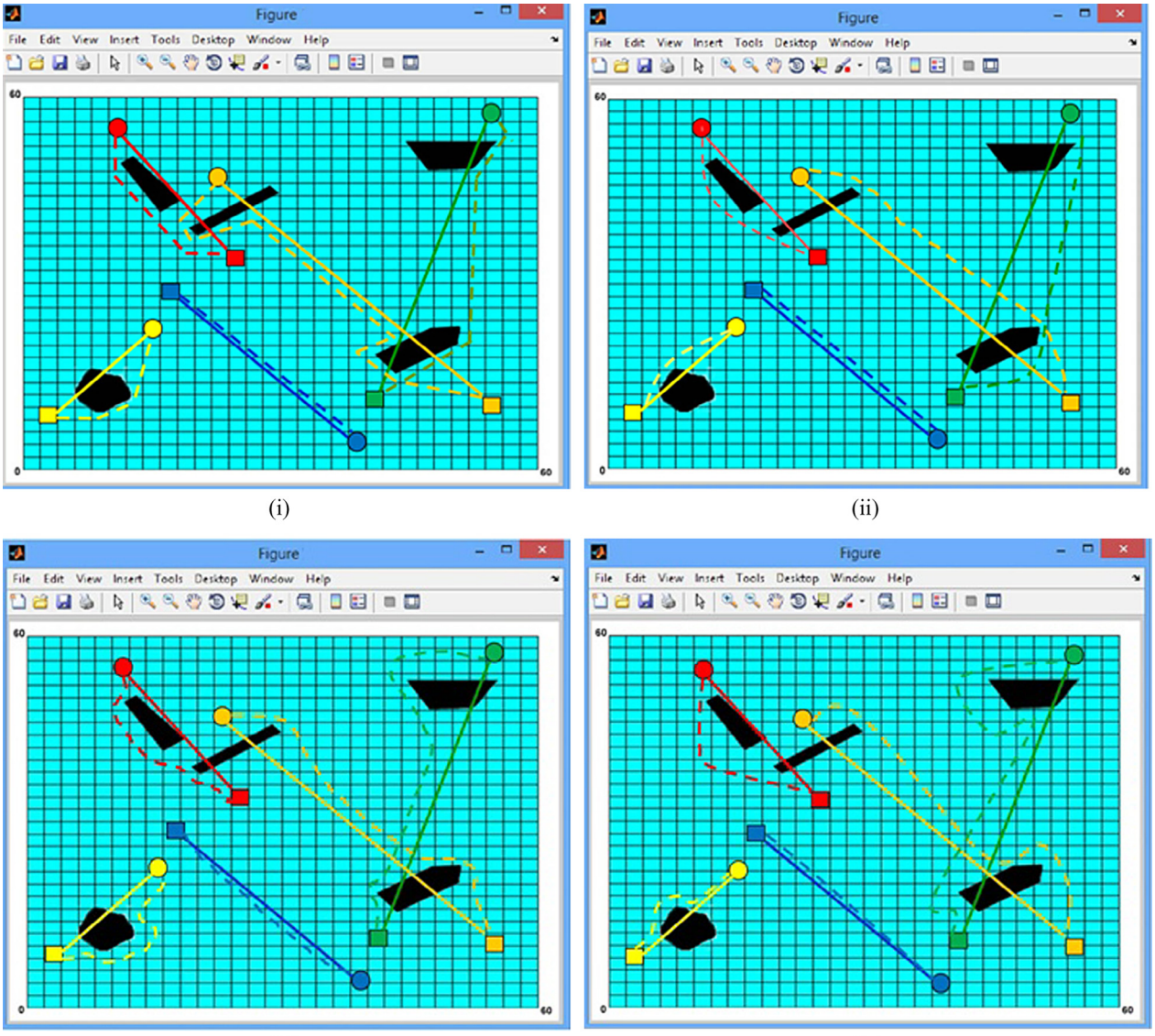


Fig. 6. All robots reached in their respective predefined goal after execution of (i) QIPSO-DV, (ii) IPSO-DV, (iii) CQL, and (iv) PSO by 21, 23, 25, 27 steps respectively.

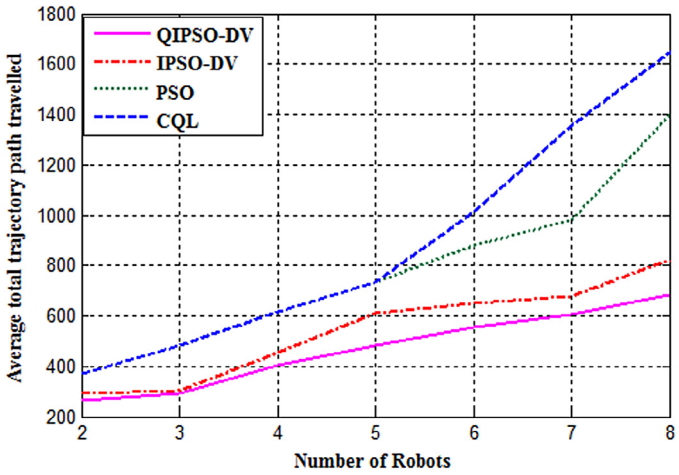


Fig. 8. Average total trajectory path traveled vs number of robots with fixed number of obstacles.

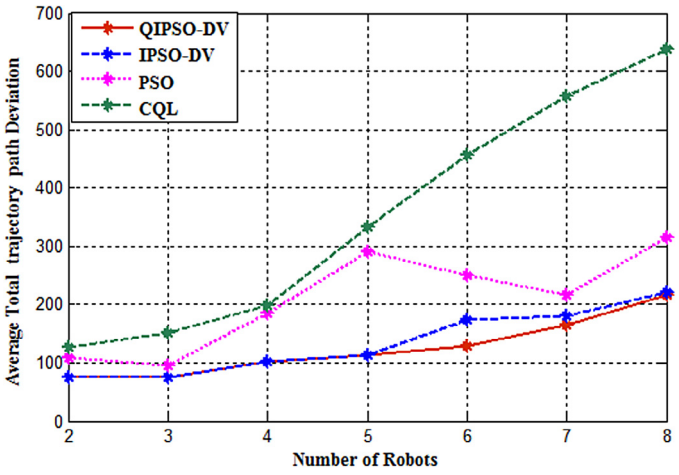


Fig. 9. Average total trajectory path deviation vs number of robots.

deviation (ATTPD) through j^{th} number of iteration for the robot R_k is expressed as $\sum_{r=1}^{j} TP_{kr} \Big/ j$. The total average trajectory path deviation for the robot R_k is calculated by taking the difference between ATTPD and the real shortest path between S_k to G_k. If TP_{k-real} is the real trajectory path for robot R_k, then the total average trajectory path deviation is expressed by

$$ATTPD = TP_{k-real} - \sum_{r=1}^{j} P_{kr} \Big/ j \tag{39}$$

Therefore for n robots in the environment, the average total trajectory path deviation (ATTPD) is

$$ATTPD = \sum_{i=1}^{n} \left(TP_{k-real} - \sum_{r=1}^{j} P_{kr} \Big/ j \right) \tag{40}$$

The performance for multi-robot navigation was analyzed in the simulation environment by plotting average total trajectory path traveled (ATTPT) with n number of robots by varying the number of robots from 1 to 8 through four algorithms, such as CQL, PSO, IPSO-DV and QIPSO-DV, and a plotted graph is presented in Fig. 8. It is noteworthy from Fig. 8 that QIPSO-DV possesses the least ATTPT in comparison to the algorithms irrespective of the number of robots. This indicates that QIPSO-DV outperforms the other three meta-heuristic algorithms in terms of average total trajectory path traveled by all robots.

Again, the performance is considered in terms of the average total trajectory path deviation in four different algorithms verse number of robots by varying the robots 1 to 8 in Fig. 9. We observed in Fig. 9 that QIPSO-DV performs better than the other three algorithms as ATTPD is smallest for QIPSO-DV in comparison to other three algorithms irrespective of the number of robots.

10.2. Average untraveled trajectory target distance (AUTTD)

On a two dimensional workspace, the mathematical expression for the untraveled trajectory target distance for a given robot k in terms of specified goal position G_k and current position C_k is $\|G_k - C_k\|$, where $\|.\|$ denotes Euclidean norm. Similarly, untraveled trajectory target distance (UTTD) for n number of robots is $\text{UTTD} = \sum_{i=1}^{n} \|G_k - C_k\|$. We can calculate the average of UTTDs in the jth number of iteration as $\text{AUTTD} = \sum_{i=1}^{n} \|G_k - C_k\| \Big/ j$. The average untraveled trajectory target distance (AUTTD) with the number of steps to goal is provided in Fig. 10 for four different algorithms. It indicates that AUTTD takes more time to converge with decreasing velocity and gradually terminated with iteration. Again, it is noted that the larger the velocity of the robot, the faster fall-off in the AUTTD. Fig. 10 shows that by increasing the number of robots, the convergence rate becomes slower. Slower convergence causes the delay in fall-off in AUTTD. The convergence of the algorithm QIPSO-DV is improved by increasing the maximum iteration value and is reflected in Fig. 10. The conclusion drawn in Fig. 10 is that AUTTD provides the smallest value of steps for QIPSO-DV.

Again the performance is calculated in terms of ending time, total traveled time of all robots and means squared error of robot/target distribution ratio by running each algorithm 30 times, and these are presented in Figs. 11–13 respectively. Fig. 11 shows that the ending time for QIPSO-DV is less; less ending time indicates that robots detect and process their target faster. Hence, we observed that in QIPSO-DV all robots detect and process their target faster than the other three meta-heuristic algorithms. The robot/target distribution ratio errors decide the robustness of the algorithm for allocating the robots dynamically into different targets. Fig. 12 shows the QIPSO-DV achieves less error in comparisons to other three algorithms. The performance is calculated in terms of power consumption, and power consumption is represented in terms of

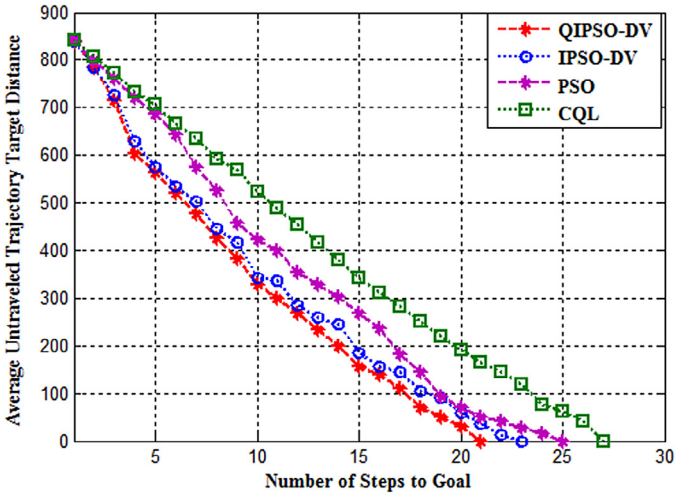


Fig. 10. Average untraveled trajectory target distance with number of steps to goal.

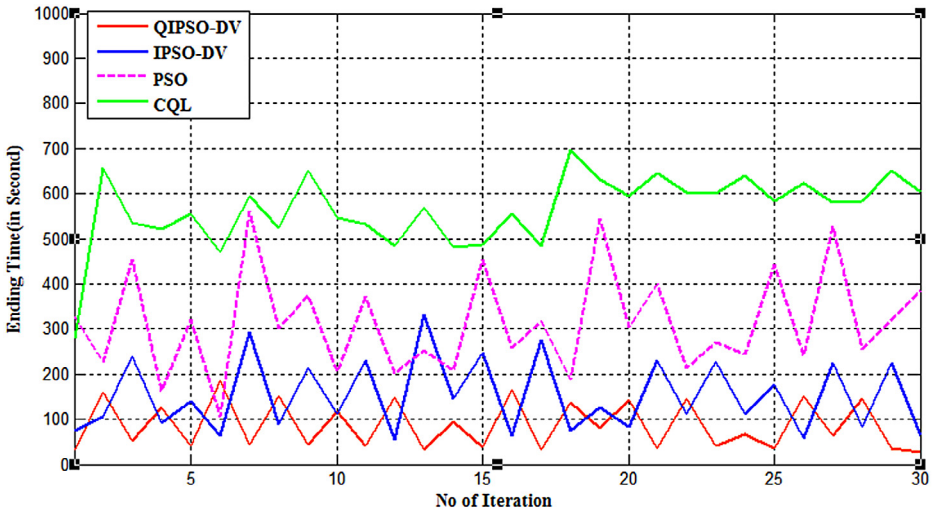


Fig. 11. Ending time vs no of iteration.

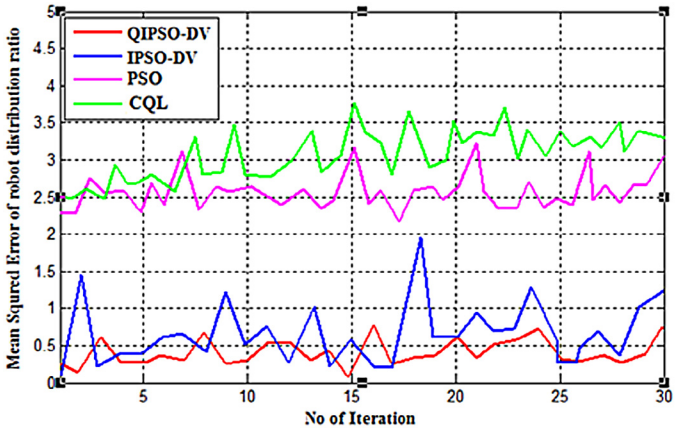


Fig. 12. Average untraveled trajectory target distance with number of steps to goal.

the total traveled time of all the robots since robot moments consume more power than communication and onboard computation of the robots. The total traveled time presented in Fig. 13 is less than the other three algorithms. The conclusion drawn from Fig. 13 is that QIPSO-DV consumes less power. Again, the performance is evaluated by the number of steps required for each robot in the different stages of iteration, and this is presented in Fig. 14. It indicates that the number of steps required for all robots in QIPSO-DV is less than the other three algorithms. Fig. 14 shows that the number of steps required is less in QIPSO-DV than the other three algorithms for five numbers of robots and five numbers of obstacles in 30 number of iterations, but it is less in QIPSO-DV irrespective of the number of iterations.

The experiment is conducted through the environment shown in Fig. 5 by four algorithms for the same fitness function presented in Eq. (10) with same parameter for 30 iterations. The fitness value generated for all robots through four different algorithms presented in Fig. 15 indicates that there is no conflict in the next position fitness value calculation by the robots; it shows that the best fitness value obtained for QIPSO-DV after 21 iteration is 3.031, but that achieved by IPSO-DV, CQL and PSO after 23, 25 and 27 is 3.631, 5.195 and 5.785 respectively. This presents that QIPSO-DV is better than

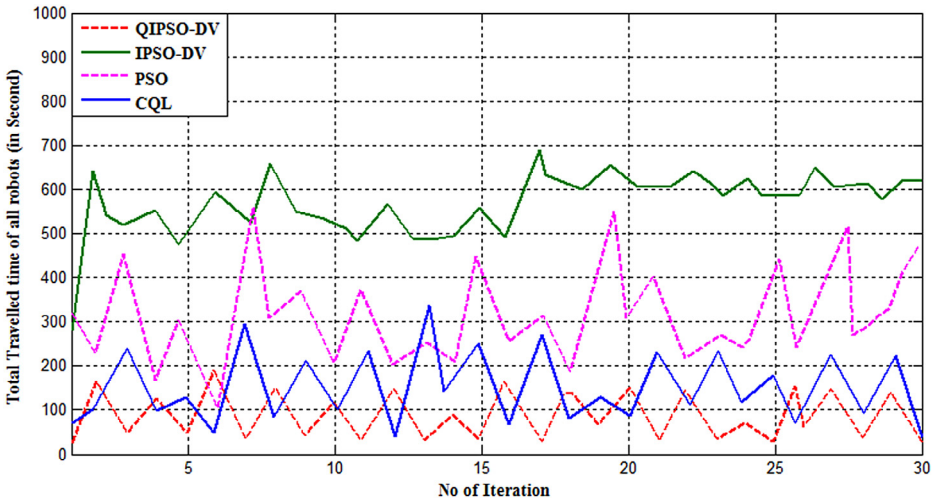


Fig. 13. Average untraveled trajectory target distance with number of steps to goal.

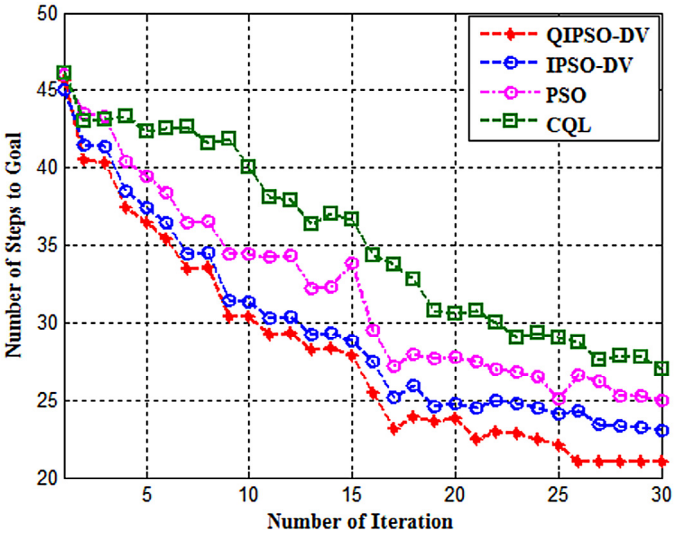


Fig. 14. Number of steps to goal with number of iteration.

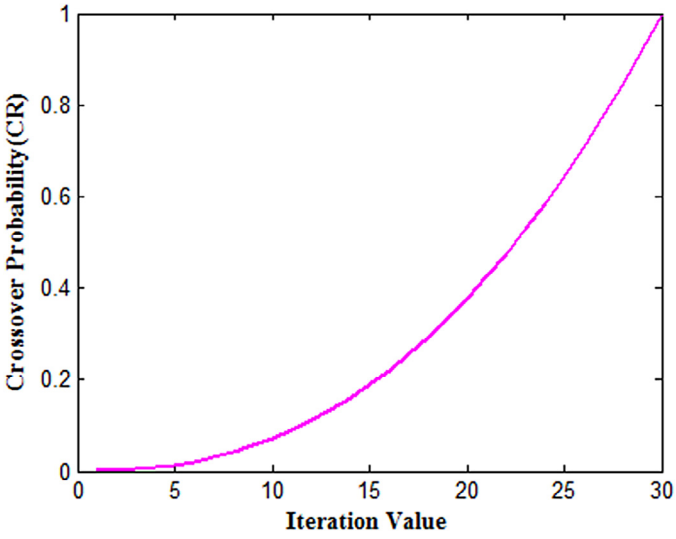


Fig. 16. rossover operator behavior.

IPSO-DV, CQL and PSO in terms of avoiding problem at local optima and faster convergence rate.

The crossover probability varying curve is presented in Fig. 16 for maximum_iteration = 30. In our implementation, at the first phase, the value of $\bar{X}_i(t)$ is affected with small range with small probabilities by crossover, and its value will be affected with full range, when iteration = maximum-iteration and probability will be 1.

The weight of the individual fitness function has been tested through simulation and the results obtained are presented in Table 1. Simulation results show that, for $\lambda_1 = 1, \lambda_2 = 1, \lambda_3 = 0.25$ and $\lambda_4 = 0.25$, it is able to generate collision free trajectory path with minimum average total trajectory path traveled and average total trajectory path deviation, and also improves the convergence rate by the proposed algorithm. Similarly, the fitness value has been evaluated for different values of $\lambda_1, \lambda_2, \lambda_3$ and λ_4, which have been presented in Fig. 17. Fig. 17 shows the convergence trained for different values of $\lambda_1, \lambda_2, \lambda_3$ and λ_4, and it is evident from the figure that fitness value is the minimum and convergence criterion satisfied after 20 iterations for $\lambda_1 = 1, \lambda_2 = 1, \lambda_3 = 0.25$ and $\lambda_4 = 0.25$, but for the other value of $\lambda_1, \lambda_2, \lambda_3$ and λ_4, the fitness value is more and starts convergence after 23 iterations.

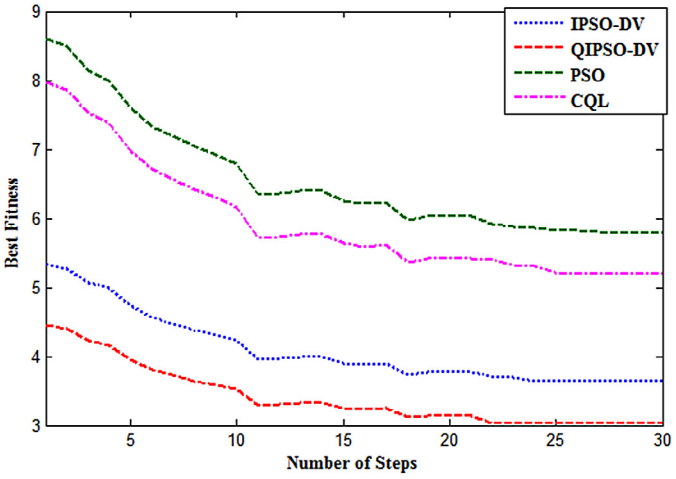


Fig. 15. Number of steps to goal with number of iteration.

Finally, the performance analysis was carried out by comparing the running time over the maximum number of iterations using four algorithms. Fig. 18 provides the time required for robots to reach their respective goal position by four different algorithms, and it shows that QIPSO-DV takes less time for robots to reach their destination.

The results of the experiments performed are summarized in Table 2 in terms of three performance metrics, namely 1) total number of steps required to reach the goal, 2) ATTPT, and 3) ATTPD; these have been used here to determine the relative merits of QIPSO-DV over the other algorithms for different robots. Table 1 confirms that outperforms the remaining three algorithms with respect to all three metrics for different robots.

11. Experiments with Khepera II robot

Khepera II is a miniature robot with a diameter of 7 cm equipped with 8 built-in infrared light sensors and 2 relatively accurate encoders for the two motors control, shown in Fig. 19. The sensors are positioned at fixed angles and have limited range of detection capabilities. The sensors are numbered clockwise starting from the leftmost sensor 0 to sensor 7 and its internal structure is presented in Fig. 20. Sensor values of the robot are measured ranging from 0 to 1023. The value of the sensor is 1023, if the obstacle has approximately 2 cm distance from the robot, and its value is zero when the obstacle is more than 5 cm from the robot. The robot consists of on board Motorola 68331 25 MHz microprocessor with a flash memory size of 512 KB. We used Khepera as a tabletop robot and connected to a workstation through a wired serial link. The Khepera II network and its accessories are presented in Fig. 21 for conduct of experiment. This configuration allows an optional experimental configuration with everything at hand: the robot, the environment and the host computer.

The improved Q-learning algorithm is used in the first phase to learn the movement steps from each grid in the map to its neighbor and compute the Q-value of each state and update its Q-value in each position based on the IPSO-DV to decide the gbest and pbest to move the best position in the world map toward goal. The initial world map for conducting the experiment in Khepera II is presented in Fig. 22 with 7 obstacles of different shapes, and predefined initial state and goal are marked on the map, where different metaheuristic algorithm is applied. Figs. 23 and 24 show the intermediate moment of the robot in the trajectory path toward the goal by

Table 1
Weight adjusted of the fitness function in Eq. (10) through simulation.

Weights				Trajectory path and convergence rate			
λ_1	λ_2	λ_3	λ_4	Average path traveled (in pixels)	Average path deviation (in pixels)	Collision free path (yes/no)	Success rate
1	1	0.6	0.7	1678	723	yes	30 (7)
			0.6	1620	684	Yes	30 (8)
			0.5	1680	725	Yes	30 (10)
			0.4	1240	539	Yes	30 (15)
			0.3	1140	460	Yes	30 (16)
			0.25	865	420	Yes	30 (20)
		0.5	0.7	1695	689	Yes	30 (12)
			0.6	1670	625	Yes	30 (12)
			0.5	1320	612	Yes	30 (18)
			0.4	1210	580	Yes	30 (20)
			0.3	1035	435	yes	30 (21)
			0.25	960	396	Yes	30 (23)
		0.4	0.7	1578	675	Yes	30 (10)
			0.6	1225	620	Yes	30 (16)
			0.5	1230	578	Yes	30 (20)
			0.4	1045	468	Yes	30 (22)
			0.3	945	375	Yes	30 (24)
			0.25	722	325	Yes	30 (25)
		0.3	0.7	1475	765	yes	30 (9)
			0.6	1135	455	Yes	30 (17)
			0.5	1048	425	Yes	30 (22)
			0.4	935	368	Yes	30 (24)
			0.3	845	320	Yes	30 (25)
			0.25	650	226	Yes	30 (26)
		0.25	0.7	1045	675	Yes	30 (12)
			0.6	860	415	Yes	30 (21)
			0.5	956	389	Yes	30 (22)
			0.4	720	320	Yes	30 (26)
			0.3	645	220	Yes	30 (27)
			0.25	**622**	**216**	**Yes**	**30 (30)**

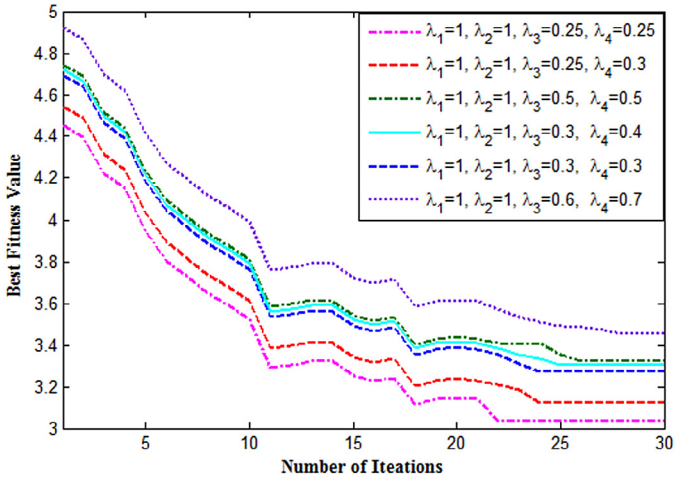


Fig. 17. Best fitness value with number of iterations for different value of weights used in fitness function in Eq. (10).

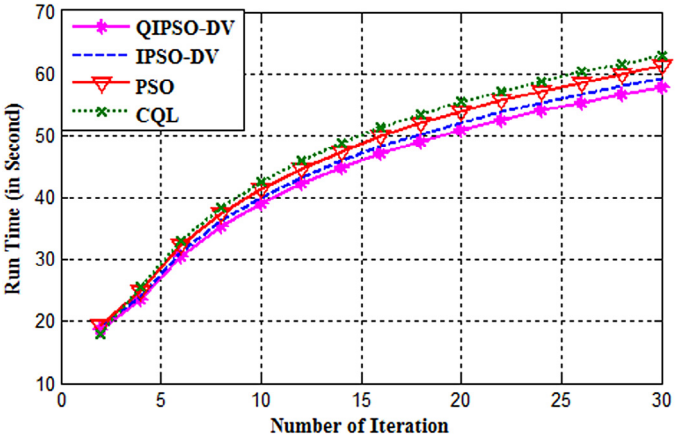


Fig. 18. Run time with number of iteration.

Table 2
Comparison of number of steps taken, ATTPT and ATTPD of different algorithms for different no. of robots.

No of robots	Algorithms (steps taken)				ATTPT (in inch)				ATTPD (in inch)			
	QIPSO-DV	IPSO-DV	PSO	CQL	QIPSO-DV	IPSO-DV	PSO	CQL	QIPSO-DV	IPSO-DV	PSO	CQL
2	12	15	16	19	34.2	35.7	36.5	35.4	3.7	4.7	5.7	4.7
3	15	17	18	22	35.7	37.8	39.6	38.4	3.9	4.9	6.8	5.8
4	17	20	21	24	37.3	39.7	42.5	39.6	4.2	6.8	7.3	6.9
5	**21**	**23**	**27**	**25**	**39.2**	**40.3**	**45.6**	**42.7**	**6.8**	**7.3**	**9.4**	**8.3**

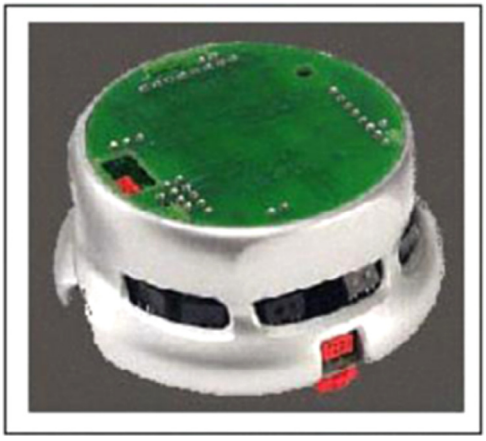


Fig. 19. The Khepera II robot.

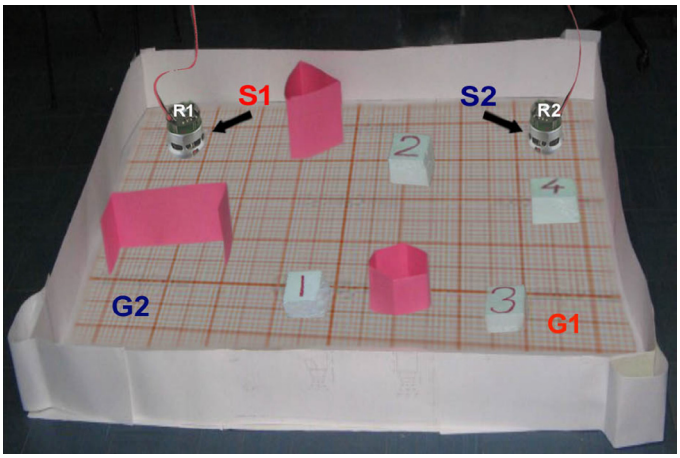


Fig. 22. Experimental world map without obstacle.

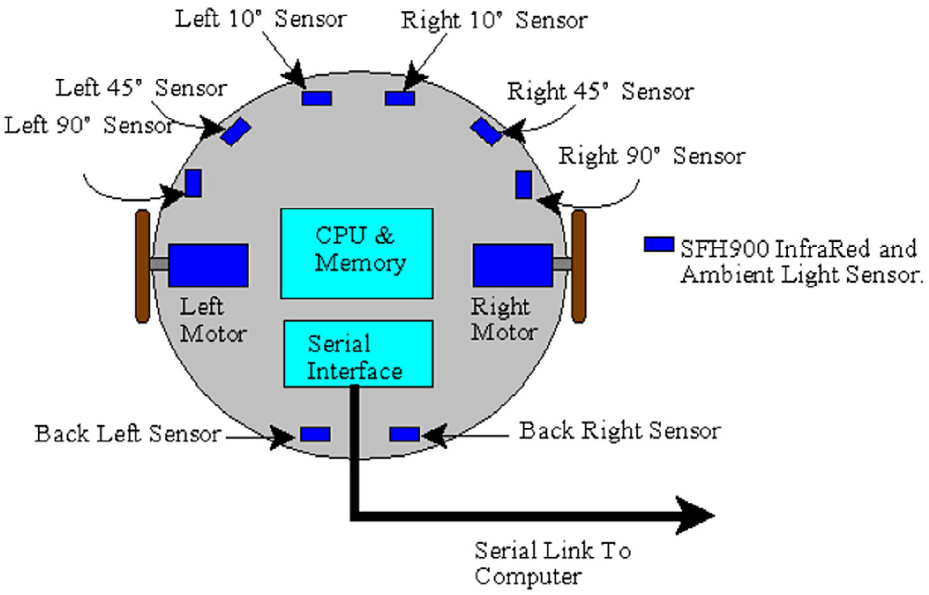


Fig. 20. Position of sensors and internal structure of Khepera II.

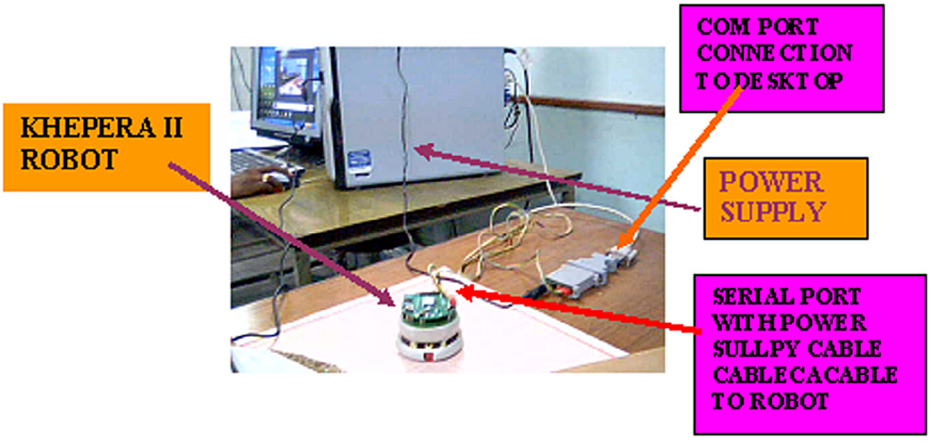


Fig. 21. Khepera network and its accessories.

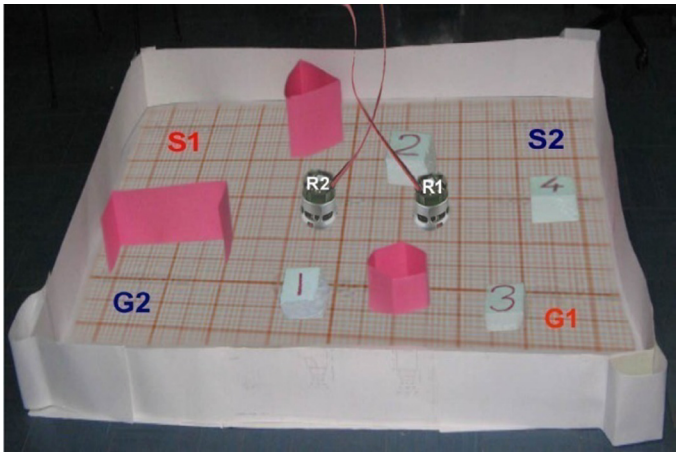


Fig. 23. Intermediate state of planning by Khepera II mobile robots using QIPSO-DV.

respective robot using QIPSO-DV. Finally, different meta-heuristic algorithm is applied in Khepera environment and the result of the trajectory path is presented in Fig. 25. QIPSO-DV is implemented in the Khepera-II environment considering two robots and compared with a different evolutionary computing algorithm, as demonstrated in Fig. 25. The QIPSO-DV implementation in Khepera-II shows better optimization in comparing with the other meta-heuristic algorithm, such as IPSO-DV, PSO and CQL, in terms of the number of turns and path traveled. In the case of QIPSO-DV, the average five numbers of turn are required to reach the destination, whereas IPSO-DV, CQL and PSO required more number of turns. This shows that QIPSO-DV is performing better than IPSO-DV, PSO and CQL.

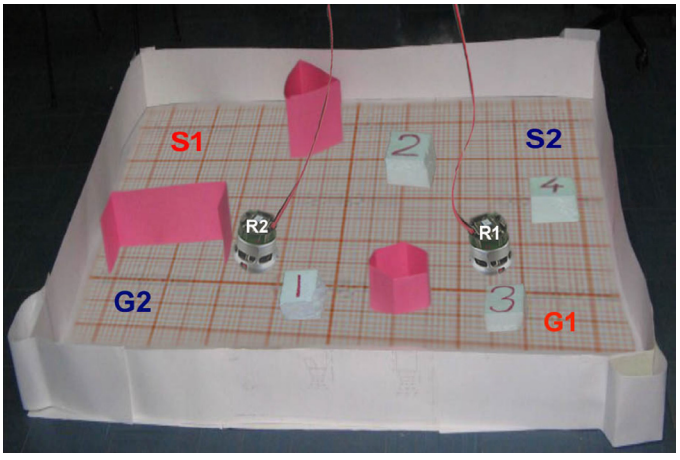


Fig. 24. Snapshot of intermediate stage of multi-robot path planning using QIPSO-DV in Khepera environment.

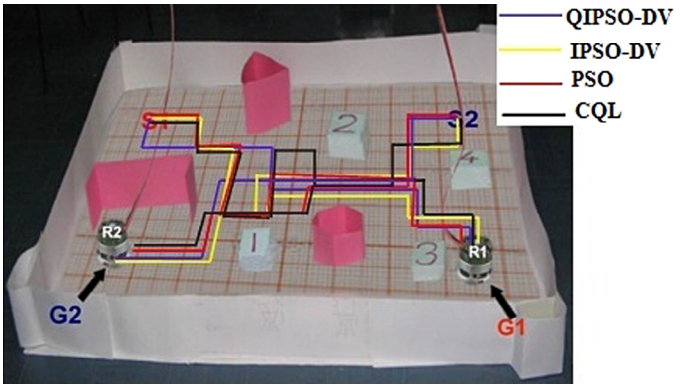


Fig. 25. Optimal path representation of different algorithm for mult-robot path planning in Khepera environment is represented by different color codes.

12. Conclusions

A hybridization of QIPSO-DV algorithm was proposed for trajectory path planning of multi-robots in order to find collision free smoothness optimal path from predefined start position to end position for each robot in the environment. The results obtained from the experimental work are in good agreement with proposed algorithm. The performance of the proposed algorithm is compared with the different meta-heuristic algorithms, such as IPSO-DV, CQL and PSO, through simulation and Khepera-II environment; it is concluded that the IPSO-DV technique is best over other algorithms used for navigation of multi-mobile robot. However, in this paper, both the environment and obstacles are static relative to the robot, whereas other robots are dynamic for priority robots. The proposed improved Q-learning reduces both time and space complexity of the classical Q-learning algorithm. A mathematical foundation of the proposed algorithm indicates the correctness, and the experiments on simulated and real platform validate without losing the optimal path in path-planning applications of a mobile robot. The algorithm is coded in MATLAB and tested on IBM Pentium-based simulation environment. The experimental study is performed on Khepera-II platform. Simulation results also confirm less number of 90° turning of the robot around its z-axis with respect to other meta-heuristic algorithm, and the proposed algorithm helps in energy saving of the robot in path-planning application.

References

[1] X. Liu, D. Gong, A comparative study of A-star algorithms for search and rescue in perfect maze, International Conference on Electric Information and Control Engineering (ICEICE), 15–17 April 2011, pp. 24–27, 2011.
[2] M. Gerke, H. Hoyer, Planning of optimal paths for autonomous agents moving in homogeneous environments, Proceedings of the 8th International Conference on Advanced Robotics, pp. 347–352, 1997.
[3] J. Xiao, Z. Michalewicz, L. Zhang, K. Trojanowski, Adaptive evolutionary planner/navigator for mobile robots, IEEE Trans. Evolut. Comput. 1 (1) (1997).
[4] Z. Bien, J. Lee, A minimum-time trajectory planning method for two robots, IEEE Trans. Rob. Autom. 8 (3) (1992) 443–450.
[5] K.G. Shin, Q. Zheng, Minimum time collision free trajectory planning for dual robot systems, IEEE Trans. Rob. Autom. 8 (5) (1992) 641–644.
[6] I. Duleba, J.Z. Sasiadek, Nonholonomic motion planning based on newton algorithm with energy optimization, IEEE Trans. Control Syst. Technol. 11 (3) (2003).
[7] M. Moll, L.E. Kavraki, Path-planning for minimal energy curves of constant length, Proceedings of the 2004 IEEE International Conference on Robotics & Automation, pp. 2826–2831, 2004.
[8] S.J. Buckley. Fast motion planning for multiple moving robots, Proc. IEEE Int. Conf. on Robotics and Autom., pp. 1419–1424, 1989.
[9] R. Smierzchalski, Z. Michalewicz, Path-planning in dynamic environments, in: S. Patnaik (Ed.), Innovations in Robot Mobility and Control, Springer-Verlag, Berlin Heidelberg, 2005.
[10] T. Dean, K. Basye, J. Shewchuk, Reinforcement learning for planning and control, in: S. Minton (Ed.), Machine Learning Methods for Planning and Scheduling, Morgan Kaufmann, 1993.
[11] R.E. Bellman, Dynamic Programming, Princeton University Press, Princeton, NJ, 2010, p. 957.
[12] C. Watkins, P. Dayan, Q-learning, Mach. Learn. 8 (1992) 279–292.
[13] A. Konar, Computational Intelligence: Principles, Techniques and Applications, Springer-Verlag, Berlin, 2005.
[14] L. Busoniu, R. Babushka, B. De Schutter, D. Ernst, Reinforcement Learning and Dynamic Programming Using Function Approximators, CRC Press, Taylor & Francis group, Boca Raton, FL, 2010.
[15] P.K. Das, S.C. Mandhata, H.S. Behera, S.N. Patro, An improved Q-learning algorithm for path-planning of a mobile robot, Int. J. Comput. Appl. 51 (No–9) (2012) 40–46.
[16] Z. Bien, J. Lee, A minimum–time trajectory planning method for two robots, IEEE Trans. Rob. Autom. 8 (3) (1992) 443–450.

[17] M. Moll, L.E. Kavraki, Path planning for minimal energy curves of constant length, Proceedings of the 2004 IEEE Int. Conf. on Robotics and Automation, pp.2826–2831, 2004.
[18] R. Regele, P. Levi, Cooperative multi-robot path planning by heuristic priority adjustment, Proceedings of the IEEE/RSJ Int Conf on Intelligent Robots and Systems, 2006.
[19] A. Konar, I.G. Chakraborty, S.J. Singh, L.C. Jain, A.K. Nagar, A deterministic improved Q-learning for path planning of a mobile robot, IEEE Trans. Syst. Man Cybern. Syst. 43 (5) (2013) 1141–1153.
[20] I. Goswami (Chakraborty), P.K. Das, A. Konar, R. Janarthanan. Extended q-learning algorithm for path-planning of a mobile robot, Eighth International Conference on Simulated Evolution And Learning (SEAL-2010), Indian Institute of Technology Kanpur, India, 2010.
[21] P. Rakshit, A. Konar, P. Bhowmik, I. Goswami, S. Das, L.C. Jain, 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.
[22] P. Rakshit, A. Konar, S. Das, A.K. Nagar, ABC-TDQL: an adaptive memetic algorithm, Hybrid Intelligent Models and Applications (HIMA), 2013 IEEE Workshop on. IEEE, 2013.
[23] C. Yi-Chang, S.-F. Lin, C.-Y. Hsu, Q-value based particle swarm optimization for reinforcement neuro-fuzzy system design, Int. J. Comp. Sci. Eng. 3 (10) (2011) 3477–3489.
[24] L. Jing, Z. Sheng, K.C. Ng, Multi-goal Q-learning of cooperative teams, Expert Syst. Appl. 38 (3) (2011) 1565–1574.
[25] P. Bhattacharjee, P. Rakshit, I. Goswami, A. Konar, A.K. Nagar, Multi-robot path-planning using artificial bee colony optimization algorithm, NaBIC (2011) 219–224, 19–21 Oct. 2011.
[26] J. Chakraborty, A. Konar, L.C. Jain, U.K. Chakraborty, Cooperative multi-robot path planning using differential evolution, J. Intell. Fuzzy Syst. 20 (1–2) (2009) 13–27.
[27] Y. Guo, L.E. Parker, A distributed and optimal motion planning approach for multiple mobile robots, in: Proc. 2002 IEEE International Conference on Robotics and Automation (ICRA'02), vol. 3, Washington, DC, USA, 2002, pp. 2612–2619.
[28] Y. Zhang, D.-W. Gong, J.-H. Zhang, Robot path planning in uncertain environment using multi-objective particle swarm optimization, Neurocomputing 103 (2013) 172–185.
[29] E. Masehian, D. Sedighizadeh, Multi-objective PSO-and NPSO based algorithms for robot path planning, Adv. Electr. Comput. Eng. 10 (2010) 69–76.
[30] N. Higashi, H. Iba, Particle swarm optimization with Gaussian mutation, IEEE Swarm Intelligence Symposium (2003) 72–79.
[31] X.-F. Xie, W.-J. Zhang, Z.L. Yang, Adaptive particle swarm optimization on individual level, Proceedings of International Conference on Signal Processing (2002), 1215–1218.
[32] C. Purcaru, R.-E. Precup, D. Iercan, L.-O. Fedorovici, R.-C. David, F. Dragan, Optimal robot path planning using gravitational search algorithm, Int. J. Comput. Intell. 10 (2013) 1–20.