Artificial Intelligence Review (2025) 58:142
https://doi.org/10.1007/s10462-025-11129-6

A novel reinforcement learning-based multi-operator differential evolution with cubic spline for the path planning problem

Mohamed Reda[1,2] · Ahmed Onsy[1] · Amira Y. Haikal[2] · Ali Ghanbari[1]

Accepted: 30 January 2025

Abstract

Path planning in autonomous driving systems remains a critical challenge, requiring algorithms capable of generating safe, efficient, and reliable routes. Existing state-of-the-art methods, including graph-based and sampling-based approaches, often produce sharp, suboptimal paths and struggle in complex search spaces, while trajectory-based algorithms suffer from high computational costs. Recently, meta-heuristic optimization algorithms have shown effective performance but often lack learning ability due to their inherent randomness. This paper introduces a unified benchmarking framework, named Reda's Path Planning Benchmark 2024 (RP2B-24), alongside two novel reinforcement learning (RL)-based path-planning algorithms: Q-Spline Multi-Operator Differential Evolution (QSMODE), utilizing Q-learning (Q-tables), and Deep Q-Spline Multi-Operator Differential Evolution (DQSMODE), based on Deep Q-networks (DQN). Both algorithms are integrated under a single framework and enhanced with cubic spline interpolation to improve path smoothness and adaptability. The proposed RP2B-24 library comprises 50 distinct benchmark problems, offering a comprehensive and generalizable testing ground for diverse path-planning algorithms. Unlike traditional approaches, RL in QSMODE/DQSMODE is not merely a parameter adjustment method but is fully utilized to generate paths based on the accumulated search experience to enhance path quality. QSMODE/DQSMODE introduces a unique self-training update mechanism for the Q-table and DQN based on candidate paths within the algorithm's population, complemented by a secondary update method that increases population diversity through random action selection. An adaptive RL switching probability dynamically alternates between these Q-table update modes. DQSMODE and QSMODE demonstrated superior performance, outperforming 22 state-of-the-art algorithms, including the IMODEII. The algorithms ranked first and second in the Friedman test and SNE-SR ranking test, achieving scores of 99.2877 (DQSMODE) and 93.0463 (QSMODE), with statistically significant results in the Wilcoxon test. The practical applicability of the algorithm was validated on a ROS-based system using a four-wheel differential drive robot, which successfully followed the planned paths in two driving scenarios, demonstrating the algorithm's feasibility and effectiveness for real-world scenarios. The source code for the proposed benchmark and algorithm is publicly available

Extended author information available on the last page of the article

Published online: 24 February 2025

for further research and experimentation at: https://github.com/MohamedRedaMu/RP2B24-Benchmark and https://github.com/MohamedRedaMu/QSMODEAlgorithm.

Keywords Path planning · Autonomous driving system (ADS) · Evolutionary algorithms · Reinforcement learning · Deep Q-network (DQN) · Cubic spline · Robotic operating system (ROS)

1 Introduction

1.1 Path planning in autonomous driving system (ADS)

Path planning is a crucial aspect of the ADS system, which is tasked with finding the optimal path for the vehicle while avoiding obstacles. The main objective of ADS is to develop a vehicle that can fully or at least partially operate without human driver (Buehler et al. 2009). The ADS pipeline consists of multiple stages including the perception, localization, mapping, path planning, and control tasks. The perception task is responsible for recognizing the obstacles in the surrounding environment such as traffic lights, road signs, and lanes (Velasco-Hernandez et al. 2020). The localization and mapping tasks generate a map of the environment using the sensor data while localizing the car within the generated map (Wong et al. 2020).

The path planning task generates a safe path from start point to the end point in the generated map while avoiding obstacles (Parekh et al. 2022). Finally, the next waypoint is converted to control actions to the vehicle's motor based on the kinematics of the vehicles' design (Reda et al. 2024). While many stages of the ADS system have been extensively researched, path planning and obstacle avoidance remain the most challenging (Yurtsever et al. 2020).

1.2 Challenges of the path planning problem

Numerous challenges in the path-planning problem demand continuous research to address them effectively. One of the biggest challenges is the conflicting multi-objective optimization nature, which requires the algorithm to minimize path length while simultaneously avoiding collision with obstacles (Gul et al. 2021). Achieving this trade-off requires sophisticated multi-objective optimization frameworks.

Environment complexity and the different representations of obstacles add more challenges to the path-planning problem. The obstacles can vary in shape, size, and representations that can be circular, rectangular, or grid-based (point-cloud) (Badrloo et al. 2022). These different representations require flexible path-planning algorithms that can adapt them without negatively affecting computational efficiency.

Accurate obstacle avoidance is another critical path-planning aspect that requires complex mathematical frameworks. These frameworks must consider all collisions resulting from the path points, the line segments connecting successive path points, and obstacles (Kunchev et al. 2006). The problem becomes even more challenging in large-scale environments with multiple obstacles, as increasing the number of obstacles necessitates scalable algorithms to handle long-range planning with multiple obstacles effectively. The implementation of these constraints adds significant complexity to the optimization problem.

Integrating the path planning algorithm with the other ADS components (e.g., perception, localization, and control) is another challenging task in real-time path planning. Moreover, the algorithm must seamlessly integrate with the control system, considering the hardware and firmware constraints for real-time processing (Butt et al. 2024). Practical execution of planned paths introduces an additional challenge that requires smooth paths to be executed by the ADS platforms to ensure maneuverability, passenger comfort, and mechanical efficiency. Guaranteeing smoothness while avoiding obstacles and minimizing other costs is a nontrivial task that demands advanced smoothing techniques within the path-planning framework.

Another issue is the validation of path-planning algorithms across diverse benchmark scenarios. Despite the good performance of many algorithms in simulated environments, these algorithms can face significant difficulties when applied to real-world scenarios due to some factors, such as sensor noise and hardware limitations. Moreover, each algorithm category in the literature requires the path-planning problem to be formulated in a specific way. This disparity hinders direct comparisons among algorithms across a unified set of benchmark problems. This issue makes the evaluation process of the state-of-the-art algorithms difficult, lacking the presence of comprehension comparative analyses among different categories of path-planning algorithms (Reda et al. 2024). Addressing these challenges requires innovative frameworks, robust mathematical formulations, and comprehensive benchmark libraries.

1.3 Path planning state-of-the-art

Various and wide ranges of state-of-the-art algorithms for the path planning problem are categorized as shown in Fig. 1. Graph-based algorithms, such as A* and Dijkstra, model the environment as a graph of connected nodes, each representing a possible location to move from the start point to the endpoint (Sun et al. 2021). Sampling-based algorithms, such as RRT and RRT*, generate random samples in the search space starting from the start point to build a map incrementally until it reaches the goal (Orthey et al. 2023).

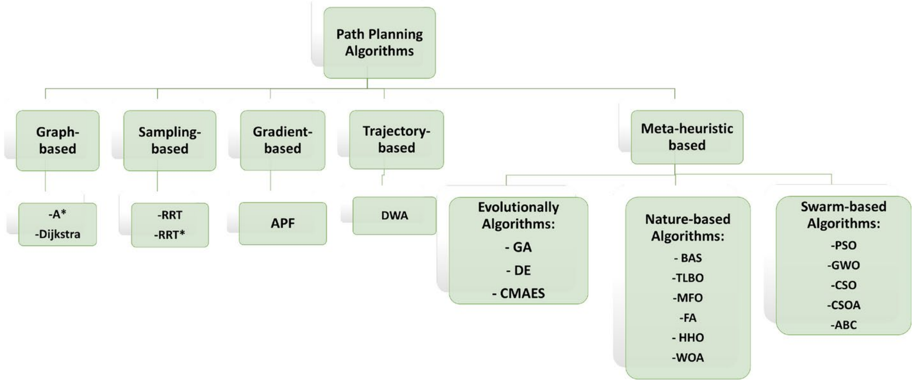


Fig. 1 Types of path planning algorithms

Gradient-based algorithms, such as the artificial potential field (APF) algorithm, use potential fields to guide the robot from the starting point to the endpoint, avoiding obstacles. The APF algorithm generates attractive force towards the goal and repulsive force from the obstacles to generate the obstacle-free path. These algorithms suffer from the local minimum traps because of the gradient behavior. Trajectory-based algorithms, like the Dynamic Window Approach (DWA), generate the path based on the robot's dynamic constraints, such as position, velocity, and acceleration (Dobrevski and Skočaj 2024). The DWA algorithm considers the velocity, acceleration limits, and obstacle positions to generate the path. It is widely used in ROS-based built-in libraries as a path-planning algorithm (ROS Wiki Contributors 2023).

Meta-heuristic optimization algorithms mimic principles from nature to solve optimization problems. They have been widely used to solve the path-planning problem by generating an initial population of candidate paths and improving them iteratively. These algorithms are considered general problem solvers and have proven their efficiency in solving optimization problems (Sood and Panchal 2020). They are categorized into three main categories: swarm-based, nature-based, and evolutionary algorithms.

Swarm-based algorithms imitate animals' natural behavior when searching for food in groups. The most popular Swarm-based algorithms in the path planning problem are the Artificial Bee Colony (ABC), Grey Wolf Optimization (GWO), Chicken Swarm Optimization (CSO), Particle Swarm Optimization (PSO), Cat Swarm Optimization Algorithm (CSOA) (Tang et al. 2021). The nature-based algorithms are inspired by natural phenomena that exist in humans and animals, such as the Firefly Algorithm (FA), the Moth-Flame Optimization (MFO), the Teaching-Learning-Based Optimization (TLBO), the Beetle Antennae Search (BAS), the Whale Optimization Algorithm (WOA), and Harris Hawks Optimization (HHO) (Houssein et al. 2019).

Evolutionary algorithms (EAs) mimic the survival of the fittest principle. The most common evolutionary algorithms in the path planning problem are Differential Evolution (DE), Genetic Algorithm (GA), and the Covariance matrix adaptation evolution strategy (CMA-ES) algorithm (Slowik and Kwasnicka 2020). The improved multi-operator differential evolution algorithm (IMODE) is an outstanding algorithm and the winner of the CEC2020 competition, outperforming all the other algorithms (Yue et al. 2019).

Multiple studies in the literature combined reinforcement learning (RL) with meta-heuristic optimization algorithms to improve the algorithm's efficiency (Goyal et al. 2019). Sallam et al. proposed the IMODEII algorithm (Sallam et al. 2022), in which the RL is used to pick the nest mutation method in the original IMODE algorithm. The IMODEII outperformed the original IMODE algorithm and other state-of-the-art winners to prove its superiority in the literature.

The EAs and meta-heuristic optimization algorithms are characterized by their strong exploration capability, which makes them less likely to get stuck in a local minimum (Slowik and Kwasnicka 2020). However, the solutions generated by meta-heuristic optimization algorithms are based on random behaviour in specific stages of the algorithms, which increases the computation time of the algorithm (Reda et al. 2024). Despite the trials of combining the RL with EAs, the RL is not fully utilized; it is just applied as an adaptive method to pick a rule or to tune a parameter in the original algorithm (Sallam et al. 2022). Therefore, the random behaviour in the EAs still dominates the performance without considering the ability to learn in the optimization problem.

1.4 Contributions

The contributions of this research are summarized as follows, aligned with the high-level workflow shown in Fig. 2:

- Development and integration of two reinforcement learning-based path-planning algorithms: Q-Spline Multi-Operator Differential Evolution (QSMODE), utilizing Q-learning (Q-tables), and Deep Q-Spline Multi-Operator Differential Evolution (DQS-MODE), based on Deep Q-networks (DQN). Both algorithms are unified under a single framework and enhanced with cubic spline interpolation to improve path smoothness and adaptability.
- Introduction of a novel reinforcement learning application in QSMODE/DQSMODE, where RL is directly employed to generate paths. Unlike existing literature, RL is not used just to update parameters; it is fully utilized to exploit accumulated experience during the search process for path generation.
- Proposal of a dual Q-table/DQN update mechanism:
  - Mode 1: Random action selection for path generation, enhancing population diversity.
  - Mode 2: A self-training method where Q-tables/DQNs are updated using candidate paths from the population, improving convergence and solution quality.
- Development of an adaptive RL switching probability mechanism to balance exploration (Mode 1) and exploitation (Mode 2) during Q-table/DQN updates, with sensitivity analysis recommending optimal bounds.
- Creation of Reda's Path Planning Benchmark 2024 (RP2B-24), a novel benchmark library comprising 50 diverse path-planning problems. This benchmark is generalizable

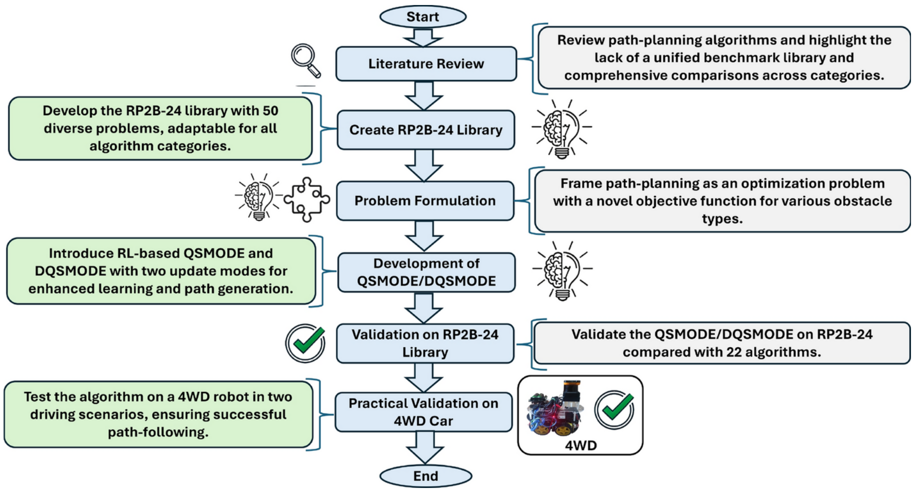

Start
Literature Review
Review path-planning algorithms and highlight the lack of a unified benchmark library and comprehensive comparisons across categories.
Develop the RP2B-24 library with 50 diverse problems, adaptable for all algorithm categories.
Create RP2B-24 Library
Problem Formulation
Frame path-planning as an optimization problem with a novel objective function for various obstacle types.
Introduce RL-based QSMODE and DQSMODE with two update modes for enhanced learning and path generation.
Development of QSMODE/DQSMODE
Validation on RP2B-24 Library
Validate the QSMODE/DQSMODE on RP2B-24 compared with 22 algorithms.
Test the algorithm on a 4WD robot in two driving scenarios, ensuring successful path-following.
Practical Validation on 4WD Car
4WD
End

Fig. 2 Proposed research workflow from literature review to algorithm development, validation, and practical implementation

and compatible with any path-planning algorithm from any category, providing a comprehensive testing framework for comparative evaluation.
- Design of an innovative objective function that frames path planning as an optimization problem. The function considers collisions between obstacles, path points, and line segments connecting successive path points, supporting circular, rectangular, and grid-based (point-cloud) representations of obstacles for real-world applications.
- Achievement of superior performance by DQSMODE and QSMODE over 22 state-of-the-art algorithms, including IMODEII, which outperformed winners of CEC competitions. The algorithms ranked first and second in the Friedman test and SNE-SR ranking, achieving scores of 99.2877 (DQSMODE) and 93.0463 (QSMODE) out of 100, with statistically significant p-values in all Wilcoxon signed rank tests across 50 path-planning problems.
- Validation of QSMODE in practical scenarios using a ROS-based autonomous driving system (ADS) on a four-wheel differential drive robot (4WD). The robot successfully followed planned paths in two distinct driving scenarios, demonstrating the real-world applicability and feasibility of the proposed framework.

1.5 Paper organization

The rest of the paper is organized as follows: Sect. 2 explores the state-of-the-art algorithms that have been applied recently for the path-planning problem. Section 3 illustrates the proposed objective function for the path planning problem. Section 4 introduces the proposed benchmark library, which consists of 50 problems with different structures. Section 5 explains the proposed QSMODE/DQSMODE algorithms and how reinforcement learning is integrated with the optimization algorithm. Section 6 explains the sensitivity of the switching probability parameter and the Q-learning impact. Section 8 presents the simulation results of the proposed algorithm and the meta-heuristic state-of-the-art algorithms, while Sect. 9 discusses the other categories of the path-planning algorithms. Section 10 demonstrates the practical validation of the proposed algorithm using a four-wheel differential drive car in two driving scenarios. Finally, the future work and overall summary are presented in Sect. 11.

2 Related work

This section explores the recent state-of-the-art algorithms applied to address the path planning problem. These algorithms are categorized as follows: graph-based algorithms, sampling-based algorithms, gradient-based algorithms, trajectory-based algorithms, and meta-heuristic optimization algorithms, as in Fig. 1 (Reda et al. 2024).

Li et al. applied the A* algorithm and the Dijkstra algorithm for the path planning of the automated guided vehicles (AGVs), and it was validated on two simulated types of obstacles, single and double-open obstacles (Li et al. 2020). Thoresen et al. applied the A* algorithm for unmanned ground vehicles (UGVs) with traversability to find the best path, and it is verified in real-time experiments in real terrain (Thoresen et al. 2021). Bhattacharyya et al. used the Dijkstra algorithm to find the shortest path to reduce energy consumption (Bhattacharyya and Karmakar 2022). Wang et al. solved the path planning problem using the Dijkstra algorithm to address the issue of localization drift error from the dead-reckoning. The algorithm is validated on a simulated map of 12 rectangular obstacles (Wang et al. 2022). Zhang et al. used the A* algorithm with a dynamic cost function for the automated guided vehicles (AGVs) (Zhang et al. 2024). Graph-based approaches obtain accurate paths for small search spaces. Nevertheless, they are sluggish for large search spaces and generate abrupt paths.

Niu et al. studied the path-planning of intelligent vehicles based on the RRT algorithm, which is validated on the robotic operating system (ROS)-based platform (Niu et al. 2021). Chen et al. integrated the RRT algorithm to reduce the search area in the path planning of autonomous vehicles (Chen et al. 2022). Shi et al. proposed an RRT algorithm for unmanned vehicles, in which the RRT algorithm generates the path, and the B-spline curve is applied to smooth it (Shi et al. 2022). Mao et al. proposed an RRT-based algorithm to address motion planning, where the RRT algorithm is used to find the path, and the cost is reduced using the elliptic sampling domain (Mao et al. 2023). Eshtehardian et al. applied the RRT* algorithm for mobile robots to find the path, and the algorithm is validated in a simulated environment of Webots (Eshtehardian and Khodaygan 2023). Sampling-based algorithms are faster than graph-based ones, but they generate too sharp solutions.

Hu et al. proposed a potential field-based algorithm for differential steering vehicles, and it was theoretically validated in an off-road environment with a single obstacle (Hu et al. 2019). Iswanto et al. introduced an APF-based algorithm for quadrotor applications, and it was validated theoretically in single and double circular obstacle environments (Iswanto et al. 2019). Batinovic et al. presented an APF-based algorithm for aerial vehicles, where the APF algorithm corrects the paths to avoid obstacles (Batinovic et al. 2022). Xie et al. used the APF algorithm to address overtaking scenarios of autonomous vehicles in multi-lanes, and it was validated in a simulation environment based on unreal engine (Xie et al. 2022). Wu et al. proposed an APF algorithm with an annealing strategy to increase obstacle avoidance efficiency, and it was validated on two single obstacles, two multiple obstacles, and two maze-like simulated scenarios (Wu et al. 2023). Sung et al. presented an APF algorithm for the ground forces environment, where the objective function implements possible enemy threads with penalty terms, and it was theoretically validated in an environment with six obstacles (Sung et al. 2023). Gradient-based algorithms can generate collision-free trajectories, but they are likely to get trapped at a local minimum due to gradient behavior.

Zhang et al. introduced a DWA-based algorithm for mobile robots, and it was validated in a ROS-based simulation platform using a two-wheel differential drive robot (Zhang et al. 2019). Hua et al. presented a DWA algorithm with an adaptive objective function, and it was validated on one simulated scenario with 16 circular obstacles (hua Zhang et al. 2021). Liu et al. used the DWA algorithm for local path planning with obstacle avoidance for the smart four-wheel robots. It was validated on one simulated environment with three rectangular obstacles and experimentally validated on a smart car with a depth camera in ROS-based platform (Liu et al. 2021). Kobayashi et al. proposed a DWA algorithm for mobile robots, and it was validated on a two-wheel differential drive robot in a single obstacle environment (Kobayashi and Motoi 2022). DWA algorithm considers the vehicle dynamics, but it has very high time complexity because it validates all possible velocity pairs while generating the trajectory.

Li et al. proposed a GA for solving robot path planning, and it was validated in a ROS-based platform in a simulated enterprise factory environment (Li et al. 2023). Zhang et al.

used the GA algorithm for mobile robots, where the environment is encoded in a grid-based format, and it was theoretically validated in one maze-like scenario with five rectangular obstacles (Zhang et al. 2023). Ab et al. proposed a GA-based algorithm for mobile robot global route planning, and it was theoretically validated on two scenarios: one with five rectangular obstacles and one with 20 rectangular obstacles (Ab Wahab et al. 2024). Yu et al. applied the DE algorithm for unmanned aerial vehicles as a constraint optimization problem, and it was theoretically validated on six scenarios (Yu et al. 2020). Chen et al. applied the DE algorithm for mobile robots with chaos initialization, and it was theoretically verified on one scenario with five circular obstacles and then validated on ROS-based simulation (Chen et al. 2020). Zhang et al. used the DE algorithm for autonomous underwater vehicles, where the problem is formulated as a constraint optimization problem (Zhang et al. 2022).

Zhao et al. applied the CMAES algorithm for unmanned ground vehicles, and it was theoretically validated on one scenario with four circular obstacles and practically validated on ROS-based four-wheel differential drive on one driving scenario (Zhao et al. 2021). Zhao et al. applied the CMAES to tracked robot terrains, and it was verified on a single scenario using the CoppeliaSim simulator (Zhao et al. 2023). Li et al. introduced a PSO-based algorithm with adaptive inertia weights for mobile robots, and the algorithm is theoretically validated on two scenarios: one with five circular obstacles and the other with 11 circular obstacles (Li et al. 2020). Qiuyun et al. proposed a PSO-based algorithm for automated guided vehicles in smart manufacturing workshops, and it was validated on two simulated scenarios (Qiuyun et al. 2021). Toufan et al. introduced a GWO-based algorithm for mobile robots with an objective function based on a laser range finder, and it was verified on two scenarios in a simulated environment using a differential drive robot (Toufan and Niknafs 2020). Kumar et al. proposed a GWO-based algorithm for autonomous robots, and it was theoretically validated vs. the PSO algorithm (Kumar et al. 2021).

Cheng et al. used the BAS algorithm in motion planning in robotic systems and proved reliable performance compared with PSO. GA, and FA (Cheng et al. 2020). Furthermore, Cui et al. applied the BAS algorithm for motion planning in industrial manipulators, and it proved its efficient performance (Cui and Li 2022). Lyu et al. applied the beetle swarm optimization, which was theoretically verified in a few scenarios (Lyu et al. 2023). Zan et al. applied the WOA algorithm for planning the robot's path with the aid of computer perception to map the surrounding environment (Zan 2022). Si and Li proposed a WOA-based algorithm, which is theoretically validated on two scenarios (Si and Li 2023). Similarly, Wang et al. proposed a WOA-based algorithm, which was validated on ten maze-like scenarios, outperforming the GWO and PSO algorithms (Wang et al. 2023). Chen et al. introduced an MFO-based algorithm for intelligent vehicles for the parking scenarios, and it is practically valid on one parking scenario (Chen et al. 2022). Dai and Wei proposed an MFO algorithm with the best flame average for mobile robots, and it was theoretically validated on three narrow passages and maze-like scenarios (Dai and Wei 2021).

Patle et al. applied the FA algorithm for mobile robot navigation, and it was validated in four theoretical and three practical scenarios (Patle et al. 2018). Achouri et al. deployed the FA algorithm for the two-wheel differential drive robots, and it was theoretically validated on four scenarios (Achouri and Zennir 2024). Majumder et al. applied the TLBO algorithm for path planning and task management for mobile robots, and it was validated on one maze-like scenario (Majumder et al. 2021). Sabiha et al. deployed the TLBO algorithm for tracked vehicles, and it was validated on a ROS-based system with one scenario (Sabiha et al. 2022). Shi et al. used the simulated annealing (SA) algorithm for path planning with boolean constraints for robotic systems (Shi et al. 2022). Yang and Vigneron applied the SA algorithm for the grid-based path planning for mobile robots (Yang and Vigneron 2022).

Xu et al. proposed an ABC-based algorithm guided by the global best path, and it was theoretically validated on two scenarios: one with two circular obstacles and the other with nine obstacles (Xu et al. 2020). Cui et al. applied the ABC algorithm for mobile robots' path planning, and it was validated on scenarios with four to ten obstacles (Cui et al. 2023). Huang et al. applied the HHO algorithm for the robot path planning and validated it on two scenarios with three and four circular obstacles (Huang et al. 2023a). Huang et al. proposed an ABC-based algorithm for grid-based path planning, and it was validated on three maze-like scenarios (Huang et al. 2023b). Fu et al. proposed a CSO-based algorithm for the hypersonic vehicles, validated on the PSOPT simulator with one scenario (Fu et al. 2019). Zhao et al. applied the CSOA algorithm for the navigation of the intelligent patrol car, and it was practically validated using a two-wheel differential drive car on one maze-like scenario (Zhao et al. 2020).

Sallam et al. proposed the superior improved multi-operator differential evolution (IMODE) algorithm (Sallam et al. 2020). The IMODE algorithm outperformed the entire meta-heuristic optimization algorithms, winning the CEC2020 competition (Yue et al. 2019). Meta-heuristic optimization algorithms are problem-independent and general problem-solvers that can handle complex scenarios. Due to the exploration capability of the search spaces, they rarely get trapped at a local minimum.

Several studies have merged reinforcement learning (RL) with meta-heuristic optimization algorithms to improve the RL performance (Goyal et al. 2019). Das et al. coupled the PSO algorithm with the Q-learning to reduce the RL's computational complexity (Das et al. 2016). Juang and Bui proposed an RL neural fuzzy surrogate-assisted model optimized with ant colony optimization (ACO) algorithm (Juang and Bui 2019). Tan et al. introduced a DE algorithm in which deep Q-network is used to pick the mutation rule from multiple DE mutation methods (Tan and Li 2021). Sallam et al. proposed a modified version of the IMODE algorithm, presented in (Sallam et al. 2020), called IMODEII (Sallam et al. 2022). In the IMODEII algorithm, the RL selects the best mutation method out of three based on a reward value and population state. IMODEII ranked first and outperformed the Spherical Search (SSA) algorithm, United Multi-operator EA (UMOEA) algorithm, IMODE-AGSK (winner of CEC2021 non-shifted), MadDE algorithm (winner of CEC2021), and IMODE algorithm (winner of CEC2020).

To the authors' best knowledge, the IMODEII is yet to be applied to the path planning problem. The usage of the RL in the meta-heuristic optimization algorithm in the literature is just picking one rule from a mixed pool of operators, which does not utilize the concept of learning using the RL. Nevertheless, the solutions generated by meta-heuristic optimization algorithms are based on random behaviour in specific stages of the algorithms, which increases the computation time of the algorithm. Therefore, the proposed QSMODE/DQSMODE algorithms utilize the RL method to generate solutions, using Q-learning (QSMODE) and DQN (DQSMODE) to build experience during the search process. The proposed algorithms modify the IMODE algorithm and integrate it with RL to add the ability to learn and decrease the response time in familiar scenarios while maintaining the exploration capability to avoid getting trapped in a local minimum in the path planning problem.

3 The proposed fitness-reward function of the path planning problem

The path starts at a starting point (x_s, y_s), ends at an endpoint (x_t, y_t), and includes *n* waypoints. Each point on the path is denoted as (x_i, y_i), where *i* ranges from 1 (start) to *n* (end). The objective is to solve a constraint optimization problem to find the shortest safe path between the start and endpoints. The shortest path minimizes the total Euclidean distance, a common metric in path planning, between successive points, as it represents the most direct route in Euclidean space (Sánchez-Ibáñez et al. 2021).

Beyond minimizing distance, the path must satisfy two constraints. First, all path points must remain within the boundaries, where $x_{\min} \leq x_i \leq x_{\max}$ and $y_{\min} \leq y_i \leq y_{\max}$. Second, the path must avoid obstacles, enforced by the condition $g(x_i, y_i, b) \geq 0$, where $g(x_i, y_i, b)$ is a general constraint function that evaluates the collision condition with the *b*-th obstacle and ensures no collision occurs (Nossier et al. 2021). The optimization problem is formulated as shown in Eq. (1).

$$minimize\ f(x, y) = \sum_{i=1}^{n-1} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2} \tag{1}$$

$$\begin{aligned} x_{min} \leq x_i \leq x_{max}, &\quad \forall i \in \{1, ..., n\} \\ y_{min} \leq y_i \leq y_{max}, &\quad \forall i \in \{1, ..., n\} \\ g(x_i, y_i, b) \geq 0, &\quad \forall i \in \{1, ..., n\},\ \forall b \in \text{Obstacles}, \end{aligned}$$

The fitness (cost) function evaluates the quality of a given path by considering both the path's total distance and any penalties that arise from collisions with obstacles. The penalty consists of two components: point-based penalties occur from collisions between path points and obstacles; segment-based penalties represent collisions between line segments connecting successive waypoints and obstacles. This formula ensures the path is both short and safe, guiding the optimization process effectively (Ma et al. 2020). The proposed cost function is designed to accommodate diverse obstacle types, with a primary focus on grid-based obstacles due to their generality and practical applicability.

The segment-based penalty is crucial in identifying issues that a point-based penalty cannot detect. Even if all waypoints lie outside obstacles, individual path segments may still intersect with them; hence, segment-based penalties are crucial for capturing these violations that a purely point-based check would overlook. Although a segment-based penalty can be enough to detect obstacle collisions, point-based checks are performed to assess or penalize the individual waypoints. Any waypoint within an obstacle indicates a high violation as the waypoint is unreachable and renders the path infeasible.

In contrast, if all waypoints are located outside the obstacles but some line segments connecting them intersect with obstacles, the path can still be considered a good candidate. These paths often require only minor refinements in subsequent iterations to eliminate the intersections. This combination of penalties ensures that paths with unreachable waypoints are penalized more heavily, while paths with minor issues can still be refined. Moreover, point-based penalties allow for a quick preliminary check, helping bypass unnecessary computations for segment-based penalties when a path is clearly invalid due to a waypoint being inside an obstacle, which decreases the computational complexity.

The cost function comprises two main components: the path length L and the total penalty p^T, as shown in Eq. (2). The path length L represents the Euclidean distance between all points on the path, defined in Eq. (3). The total penalty p^T accounts for collisions and is scaled by the weight β, set to 100 to prioritize safety by heavily penalizing unsafe paths For a collision-free path, the total cost equals the path length L, representing the minimum achievable cost.

The total penalty p^T represents the average penalty across all obstacles, including point- and line-segment-based penalties, as shown in Eq. (4), where B is the total number of obstacles For each obstacle b, the penalty includes two terms: $\sum_{i=1}^{n} p_i^b$, which evaluates penalties for collisions involving the n path points (x_i, y_i), and $\sum_{j=1}^{n-1} p_{\text{line},j}^b$, which evaluates penalties for collisions involving the $n-1$ line segments $L_j = [(x_j, y_j), (x_{j+1}, y_{j+1})]$. The penalty is normalized by dividing by B, the total number of obstacles, to ensure the penalty reflects the overall distribution and severity of collisions.

$$cost = L * (1 + \beta p^T) \tag{2}$$

$$L = \sum_{i=1}^{n-1} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2} \tag{3}$$

$$p^T = \frac{1}{B} \sum_{b=1}^{B} \left(\sum_{i=1}^{n} p_i^b + \sum_{j=1}^{n-1} p_{\text{line},j}^b \right) \tag{4}$$

For grid-based obstacles, the two components p_i^b and $p_{\text{line},j}^b$ are calculated using a grid representation. The search space is represented as a grid-based environment, where the grid is defined by its origin (x_g, y_g) and resolution $(x_{\text{res}}, y_{\text{res}})$. The x-coordinate ranges from $x_{\min}$ to $x_{\max}$ in steps of x_{res}, while the y-coordinate is discretized from $y_{\min}$ to $y_{\max}$ in steps of y_{res}. Each point on the path, (x_i, y_i), is mapped to a corresponding cell in the grid with indices (xc_i, yc_i), calculated using Eq. (5). Each cell in the grid is assigned a cost $M(xc_i, yc_i)$, where 0 represents a free cell, and 100 indicates a fully occupied cell, as determined by mapping algorithms such as SLAM. These occupancy values reflect the proximity of the cell to obstacles. For the point-based penalty p_i, the cost $M(xc_i, yc_i)$ of the cell corresponding to the point (x_i, y_i) is normalized by dividing by 100, yielding a penalty in the range [0, 1], as shown in Eq. (6).

$$xc_i = \text{round}\left(\frac{x_i - x_{og}}{x_{\text{res}}}\right), \quad yc_i = \text{round}\left(\frac{y_i - y_{og}}{y_{\text{res}}}\right) \tag{5}$$

$$p_i = \frac{M(xc_i, yc_i)}{100} \tag{6}$$

The segment-based penalty $p_{\text{line},j}$ is calculated by identifying all grid cells intersected by a line segment L_j connecting two consecutive waypoints. Bresenham's line algorithm determines these cells, where n_j is the number of these cells. Figure 3 illustrates the intersection between the line segment and the grid cells. The penalty is then aggregated across these intersected cells by averaging their normalized occupancy costs, as shown in Eq. (7). Equa-

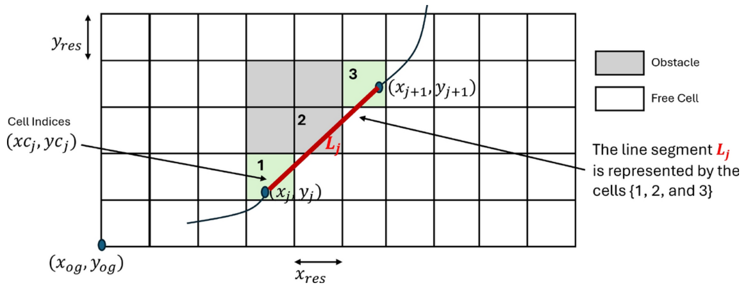


Fig. 3 Illustrative figure shows the segment-based cost in grid-based environment

tion (4) is modified to calculate the overall path penalty p^T in the grid-based environment as shown in Eq. (8), where the cost is not refereed to the obstacles because it is obtained directly from the grid. During the practical validation, the LiDAR sensor collects distance information, and then the SLAM algorithm is used to generate the occupancy grid that is used to evaluate the path cost.

$$p_{\text{line},j} = \frac{1}{n_j} \sum_{k=1}^{n_j} \frac{M(xc_k, yc_k)}{100} \tag{7}$$

$$p^T = \sum_{i=1}^{n} p_i + \sum_{j=1}^{n-1} p_{\text{line},j} \tag{8}$$

The proposed cost function supports various obstacle types, focusing here on grid-based obstacles for their generality and practical validation via SLAM-generated occupancy grids. Detailed formulations and calculations for p_i^b and $p_{\text{line},j}^b$ related to circular (Sect. S2.1) and rectangular (Sect. S2.2) obstacles, as well as conversion mechanisms between these types (Sect. S2.3), are provided in the supplementary materials, highlighting the framework's versatility.

4 The proposed library for path planning scenarios (RP2B-24)

This section introduces a proposed library of path-planning scenarios with different configurations that will be used in the path-planning comparative analysis. Bench-MR is an open-source benchmarking framework that generates path-planning scenarios. It is designed especially for sampling-based motion planning algorithms such as RRT and RRT* algorithms (Heiden et al. 2021). Arena-bench is another benchmark platform for generating path planning and obstacle avoidance scenarios based on neural networks (Kästner et al. 2022). These libraries are effective in generating the scenarios. Still, it is complex to integrate with the other algorithms to be tested on it, where each algorithm needs to be manually integrated into the library to be tested on it, which is time-consuming.

Therefore, a library of 50 challenging path-planning scenarios is proposed, called Reda's Path Planning Benchmark 2024 (RP2B-24). RP2B library evaluates the algorithm in a unified architecture that can be easily deployed for any path-planning algorithm. In the proposed library, the Bench-MR and Arena-bench benchmark platforms generate multiple scenarios as an initial configuration. Then, the obstacle configuration is collected and fine-tuned to make it more challenging. In addition, more complicated scenarios have been designed to create more challenging scenarios. These configurations have been structured into a proposed data structure that formulates a simple unified architecture that can be called with any path-planning algorithm.

Path planning with obstacle avoidance scenarios is categorized into five types in the proposed library: open field, single obstacle, multiple obstacle, narrow passages, and maze-like scenarios. For each type, there are multiple configurations: One for the open-field, two for the single obstacle, 13 configurations for the multiple obstacles case, 21 configurations for the narrow passage scenarios, and 13 configurations for the maze-like scenarios. The total number of configurations is 50, labeled from M1 to M50. Each configuration has three main attributes: the start point, the endpoint, and the configuration of the obstacles. In the open field, obstacles have smooth circular outlines and single-obstacle and multiple-obstacle scenarios. In the narrow passages and maze-like scenarios, obstacles have sharp rectangle shapes.

All the obstacles in this library are assumed to have the same priority and type. However, due to the dynamic structure of the proposed library, another attribute can be added to the obstacles to give more priority to some obstacles over others. The priority of obstacles requires an image processing layer that can identify the type of obstacle and assign a priority level based on its severity or danger. This point is beyond this study. However, this can be extended to the dynamic structure of our proposed architecture, and all the obstacles are assumed to have high priority and must be avoided.

The open field case is the simplest case, where there are no obstacles at all. It is required to find the path between the start point and the endpoint. This simple case shows how the algorithms perform in the basic simple scenario. In autonomous driving, the default scenario is the open field one because the car is supposed to move in a straight line in its lane. The obstacle avoidance case is the exception, where an object hinders the car's path. The shortest path between any two points in the open field is a straight line. However, some algorithms generate a curved path in this simple case. Therefore, it is crucial to consider the open-field scenario during testing.

The single obstacle case has only one obstacle between the start and endpoints. On the other hand, the multiple obstacle scenarios have more than one obstacle between the start and endpoints. The narrow passage scenarios have rectangle obstacles where they are too close to each other, making the path routing more challenging. The maze-like scenarios have thin, sharp barriers between the start and the endpoints. The boundaries for all 50 models are the same, where the x-coordinates lie between -10 and 10, and the y-coordinates lie between -10 and 10.

Figure 4 illustrates the distribution of the start points, endpoints, and obstacles for selected models in the proposed library. The complete set of configurations (M1-M50) and their detailed descriptions are available in the supplementary materials (Sect. S1; Fig. S1 and Table S1).

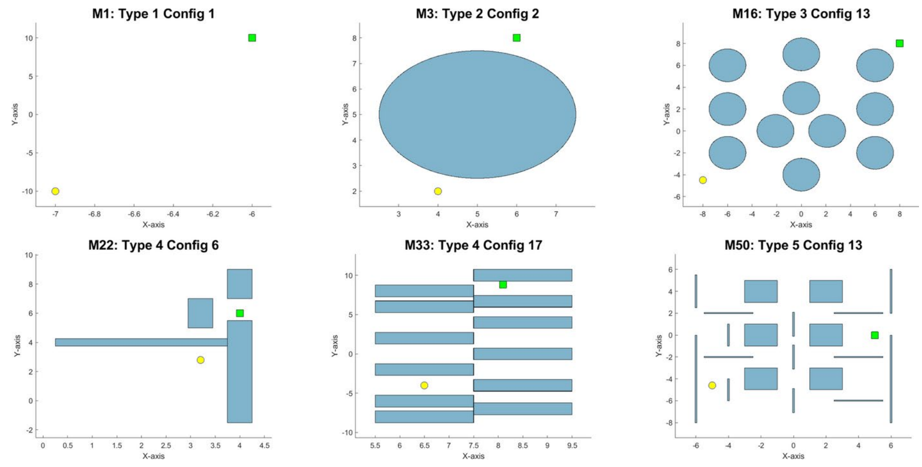


Fig. 4 Samples of model configurations for the proposed library of the path planning scenarios. The obstacles are blue in colour, the start point is a yellow circle, and the endpoint is a green square

5 The proposed path planning algorithm

This section explains the basis of the proposed QSMODE/DQSMODE algorithms for the path planning problem. Two significant aspects have been addressed here to solve two major issues for the state-of-the-art algorithms that deal with the path planning problem. First, the sharp solutions obtained from the algorithms in the literature have been addressed by adding a cubic-spline interpolation technique to fine-tune the paths generated by the algorithm. Second, the lack of the ability to learn in meta-heuristic optimization algorithms has been addressed by merging the reinforcement learning techniques in a novel manner with the proposed QSMODE and DQSMODE algorithms.

5.1 Population encoding

The proposed QSMODE/DQSMODE algorithm starts by generating an initial population of candidate solutions. These solutions must be encoded first to represent the solutions for the optimization problem being solved. In the path planning problem, the population P consists of a set of individuals I_i, each representing the candidate path from the start to the endpoint, as in Eq. (9), where n_{pop} is the number of individuals in the population.

Each individual I_i represents the candidate path, with pairs of waypoints as x and y coordinates. The path's waypoints dynamically change with the number of obstacles, as in Eq. (10), where the rate β is the waypoints rate (set to 1), and n_{obst} is the number of obstacles. The number of waypoints *nd* represents the dimension size of the optimization problem. The proposed QSMODE and DQSMODE algorithms are typically applied to the population like any other optimization problem.

$$P = \left\{ I_i = \left\{ (x_j^i, y_j^i) \right\}_{j=1}^{n_d} \right\}_{i=1}^{n_{pop}} \tag{9}$$

$$nd = \max(\text{round}(\beta \times n_{obst}), 1) \tag{10}$$

5.2 Smoothing technique using cubic spline interpolation

Path planning algorithms in the literature suffer from sharp paths, especially in tight corners and turns. These path-planning scenarios are challenging, and sharp paths are not feasible for the car to follow as they may lead to a collision. In the proposed algorithms, a cubic spline technique is used to make the path generated by the algorithm smoother. In practice, the waypoints generated from the QSMODE/DQSMODE algorithms are the main points to be followed. Additionally, the spline's smooth path is stored for use in case of sharp turns and corners. This advantage gives the proposed algorithm more reliability when dealing with more challenging scenarios.

The cubic spline interpolation method creates a smooth path through a set of waypoints Li et al. (2020). The generated path from the QSMODE/DQSMODE algorithms consists of *nd* waypoints, including the start and endpoints. This path is smoothed by breaking it into two groups of segments. The first group represents the main segments that define the knots and the junction points between each piece of the cubic spline curve, defining the main structure of the path. The number of main segments is $nd - 1$, which consists of the main waypoints of the path (Lian et al. 2020). The main segments of the entire path have an evaluation parameter T that ranges from 0 to 1. This parameter is broken into a set of $nd - 1$ continuous segments t_i, such that the main waypoints are uniformly distributed within the whole range of parameter T, as in Eq. (11).

The second group is the secondary segments representing the discrete spline segments over the entire path from the start to the endpoints. The entire path is discretized into a set of *ns* spline points, dividing into $ns - 1$ segments. The entire path will have another evaluation parameter, K, that ranges from 0 to 1. This parameter is broken into $ns - 1$ continuous segments k_i such that the spline points are uniformly distributed within the whole range of parameter K, as in Eq. (12).

Each secondary segment $S_i(k)$ parameterized by k_i is expressed by the cubic spline piecewise polynomial function in Eq. (13), where (a_i, b_i, c_i, d_i) are the coefficients defining the cubic polynomial of the i-th segment of the spline. This function interpolates between the spline points by obtaining the coefficient of each secondary segment. These segments generate a smoother path than the originally generated path, giving more flexibility in handling the tight and sharp turns.

Figure 5 illustrates the difference between the primary segments and their evaluation parameter ranges and the secondary spline segments and their corresponding range. The number of the main waypoints nd is assumed to be 3, while the number of spline points ns is assumed to be 6.

$$T = \{t_i\}_{i=1}^{nd-1}, \quad \text{where } t_i \in \left[\frac{i-1}{nd-1}, \frac{i}{nd-1}\right]. \tag{11}$$

$$K = \{k_i\}_{i=1}^{ns-1}, \quad \text{where } k_i \in \left[\frac{i-1}{ns-1}, \frac{i}{ns-1}\right]. \tag{12}$$

$$S_i(k) = a_i(k - k_i)^3 + b_i(k - k_i)^2 + c_i(k - k_i) + d_i \tag{13}$$

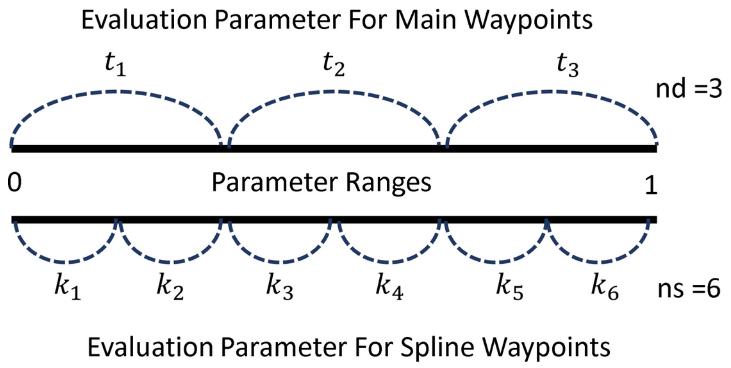

Evaluation Parameter For Main Waypoints
t_1
t_2
t_3
nd =3
0
Parameter Ranges
1
k_1
k_2
k_3
k_4
k_5
k_6
ns =6
Evaluation Parameter For Spline Waypoints

Fig. 5 The ranges of the evaluation parameters in the cubic spline interpolation

5.3 Reinforcement learning: Q-learning and deep Q-network (DQN) techniques

The proposed QSMODE and DQSMODE algorithms enhance the optimization process by integrating reinforcement learning techniques, enabling them to learn iteratively from the population experience and adapt to complex scenarios. This learning ability distinguishes them from traditional optimization methods, where each run starts afresh without prior experience. In contrast, QSMODE and DQSMODE accumulate experience, improving convergence speed for real-time applications and their ability to handle challenging path planning scenarios.

5.3.1 Formulation of states, actions, and rewards in QSMODE and DQSMODE

Reinforcement learning in both QSMODE and DQSMODE is modeled using the Markov Decision Process (MDP), which comprises a set of states S, a set of actions A, and a reward function $R(s, a)$. If an action $a \in A$ is taken at a state $s \in S$, the system transitions to a new state $s^n \in S$, which is evaluated using the reward function $R(s, a)$. The reward value is represented by $Q(s, a)$, which quantifies the long-term expected reward of taking action a at state s, guiding the agent to optimize decision-making (Jang et al. 2019). For QSMODE, these $Q(s, a)$ values are stored in a Q-table, accumulating knowledge over iterations. For DQSMODE, the Q-table is replaced by a deep Q-network (DQN), which approximates the same knowledge as a neural network.

In the path-planning problem, the state s represents the car's location as (x, y) coordinates. The state space is discretized into a grid with resolutions $(x_{\text{res}}, y_{\text{res}})$, where each grid cell corresponds to a unique state. The complete set of states S is defined in Eq. (14), where $x_{\min}$ and $y_{\min}$ are the lower bounds of the grid, while $x_{\max}$ and $y_{\max}$ are the upper bounds. The total number of grid cells along the x- and y-axes are $n_{cx} = \frac{x_{\max}-x_{\min}}{x_{\text{res}}}$ and $n_{cy} = \frac{y_{\max}-y_{\min}}{y_{\text{res}}}$, respectively.

$$S = \{(x_i, y_j) \mid x_i = x_{\min} + i \cdot x_{\text{res}},\, y_j = y_{\min} + j \cdot y_{\text{res}}\}_{i=1,j=1}^{i=n_{cx},j=n_{cy}} \tag{14}$$

In QSMODE, this discrete grid representation is more suitable for the state-action mapping in the Q-table. For DQSMODE, the input space can represent discrete or continuous (x, y) coordinates, offering more flexibility. However, using a discrete input space accelerates the process and aligns with real-time maps generated by SLAM algorithms, making it suitable for real-time path planning applications.

The set of actions A represents discrete rotation angles θ that control the car's steering direction, as in Eq. (15), where n_a is the number of possible actions. Each action θ_i specifies the car's direction to reach the next state. The resolution of n_a balances decision precision and computational complexity, enabling effective path planning for both QSMODE and DQSMODE.

$$A = \left\{ \theta_i = \frac{i \cdot 360}{n_a} \right\}_{i=1}^{n_a} \tag{15}$$

The reward function in the Q-learning framework is designed to encourage progress toward the goal while penalizing obstacle violations. It is derived from the fitness function in Eq. (2) and reformulated to maximize the reward by negating the cost. The reward for taking an action a from the current state $s = (x, y)$ is expressed in Eq. (16), where $d(s^n, s_t)$ denotes the Euclidean distance between the next state $s^n = (x_n, y_n)$ (after taking action a) and the goal $s_t = (x_t, y_t)$, as in Eq. (17). The term $p^T(s, a)$ represents the total penalty associated with the action a from state s Eq. (18), where $p_i^b(s^n)$ accounts for the s^n point-collision with obstacles, and $p_{\text{line},j}^b(s, s^n)$ penalizes collision between the line segment (s, s^n) and obstacles. These terms are computed as described in Sect. 3.

$$r(s, a) = -\left(d(s^n, s_t) + p^T(s, a) \right), \tag{16}$$

$$d(s^n, s_t) = \sqrt{(x_n - x_t)^2 + (y_n - y_t)^2}, \tag{17}$$

$$p^T(s, a) = \sum_{b=1}^{B} \left(p_i^b(s^n) + p_{\text{line},j}^b(s, s^n) \right). \tag{18}$$

5.3.2 QSMODE: Q-table structure and update

QSMODE incorporates a Q-table to store and update the Q-values for all state-action pairs. The Q-table in QSMODE is a two-dimensional matrix, where each row corresponds to a state $s = (x, y)$, and each column corresponds to a possible action $a = \theta$. The table's entries $Q(s, a)$ store the long-term expected rewards for taking action a at state s.

First, the agent (car) selects the action $a \in A$ with the highest Q-value at the current state $s \in S$, where the Q-value $Q(s, a)$ is retrieved from the Q-table. This action corresponds to the steering angle θ that directs the car from its current location $s = (x, y)$ to the next location $s^n = (x^n, y^n)$, and its direct reward r is calculated using Eq. (16). Figure 6 illustrates the concept and the relationships among s, a, and r.

The Q-value for the current state-action pair $Q(s, a)$ is updated based on the Q-learning update rule Eq.(19), where $\alpha \in [0, 1]$ is the learning rate, which controls how much of the new information overwrites the old Q-value, and $\gamma \in [0, 1]$ is the discount factor,

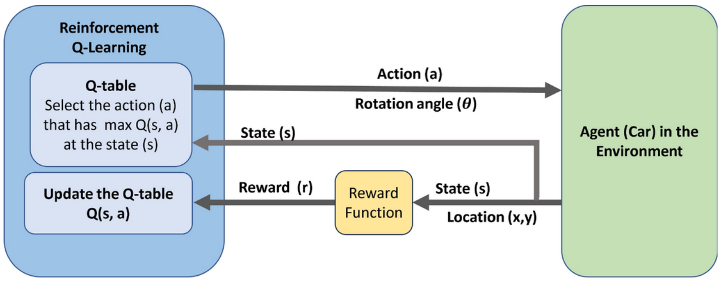


Fig. 6 Concept of reinforcement learning based on state s, action a, and reward r

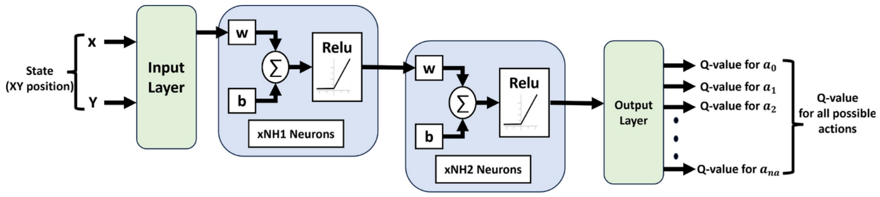


Fig. 7 The DQN network architecture for the proposed DQSMODE algorithm

which balances the immediate reward r and the future reward $\max_{a^n} Q(s^n, a^n)$. The term $\max_{a^n} Q(s^n, a^n)$ represents the maximum future reward achievable from the new state s^n for all possible actions a^n (Clifton and Laber 2020).

$$Q(s,a) = Q(s,a) + \alpha \left[r + \gamma \max_{a^n} Q(s^n, a^n) - Q(s,a) \right]. \tag{19}$$

5.3.3 DQSMODE: deep Q-network (DQN) structure and update

DQSMODE incorporates the deep learning concept by replacing the Q-table in the QSMODE algorithm with a deep Q-network (DQN). The QSMODE algorithm's Q-table explicitly lists all states and their possible actions, leading to significant memory consumption in complex search spaces. The DQSMODE algorithm addresses this issue using a DQN, which approximates the entire Q-table as a neural network. This approach enables continuous state representation, reduces memory requirements, and eliminates the scalability challenges associated with the Q-table (Fan et al. 2020). DQSMODE preserves the RL fundamentals of QSMODE while extending its capability to address larger and more complex problems. This improvement allows the DQSMODE to manage expansive state-action spaces effectively, ensuring both efficient learning and high-quality solutions.

5.3.4 DQN structure

The DQN receives the current state and predicting the expected reward (Q-value) for each possible action in that state. The structure of the proposed DQN is illustrated in Fig. 7. The input layer accepts the state (position) represented by continuous x and y coordinates, with

$x \in [x_{\min}, x_{\max}]$ and $y \in [y_{\min}, y_{\max}]$. This approach avoids the discrete indices (required by a Q-table) and can be adapted for both continuous and discrete state spaces, enabling flexibility in representing various environments.

Two hidden layers are employed, with the size of the first hidden layer *NH*1 determined as $\frac{2}{3} \times \text{Input size} + n_a$, where n_a is the number of possible actions. For the second hidden layer, the size *NH*2 is set to approximately half of the first hidden layer to balance computational efficiency and model generalization (Heaton 2008). The ReLU activation function is used for both hidden layers to enhance the network's ability to capture non-linear relationships while maintaining computational efficiency (Lv et al. 2019). This configuration minimizes training complexity while ensuring suitability for real-time path planning and balancing computational efficiency and learning capability. The output layer predicts the Q-value for each possible action, with the number of outputs corresponding to the set of actions defined in Eq. (15).

5.3.5 DQN training

The DQN is trained iteratively, using a replay memory to store the history of state transitions during optimization. Each entry in the replay memory contains the current state *s*, the selected action *a*, the next state s^n, and the immediate reward *r* obtained after taking action *a* in state *s*. A replay memory of size n_m ensures adequate history without excessive computational overhead (Liu and Zou 2018).

During training, a subset (batch) of replay memory of size n_b, is sampled randomly, serving as the training dataset for each cycle. For every state *s* in the batch, predicted Q-values for all possible actions are computed using the primary Q-network $Q_{\text{main}}(s, a)$. Future Q-values for the next states s^n are predicted using a target Q-network $Q_{\text{target}}(s^n, a^n)$, which is updated less frequently to stabilize learning (Halat and Ebadzadeh 2021). Using separate networks to predict future Q-values enhances stability and avoids the moving target problem. Since the primary network undergoes frequent updates, relying on it to evaluate future Q-values would lead to inconsistent results for the next states s^n. Conversely, updated less frequently, the target Q-network provides a stable reference for evaluating future Q-values (Halat and Ebadzadeh 2021).

The training data consists of input–output pairs, in which the inputs are the (*x*, *y*) coordinates of the states in the batch, resulting in a dataset of $n_b \times 2$ dimensions. The outputs represent the Q-values for all possible actions at each state, with a size of $n_b \times n_a$. For each state *s*, one action $a \in A$ is selected, while the remaining actions, denoted as $A^* \subset A$ where $a \notin A^*$, are updated separately. For each action a^* in the remaining actions set A^* , the Q-values, represented as $Q_{\text{outputs}}(s, a^*)$, are directly computed using the primary Q-network $Q_{\text{main}}(s, a^*)$ (second branch in Eq. (20)), reflecting the predictions made by the primary Q-network during the current training cycle.

In contrast, the Q-value for the selected action *a*, denoted as $Q_{\text{outputs}}(s, a)$, is updated using the Bellman update rule (first branch in Eq. (20)) to incorporate the learning dynamics of reinforcement learning (Zhao 2022). Where *r* represents the immediate reward after taking action *a* in state *s*, γ is the discount factor (a value between 0 and 1), and $\max_{a^n}\left(Q_{\text{target}}(s^n, a^n)\right)$ denotes the maximum predicted Q-value of the next state s^n across all possible actions a^n as evaluated by the target Q-network.

$$Q_{\text{outputs}}(s,a) = \begin{cases} r + \gamma \max_{a^n} \left(Q_{\text{target}}(s^n, a^n)\right), & \text{Where } a \text{ is the selected action,} \\ Q_{\text{main}}(s, a = a^*), & \forall a^* \in A^*,\ a^* \neq a, a \notin A^*. \end{cases} \tag{20}$$

The Mean Squared Error (MSE) loss is obtained for all current states in the batch (Lv et al. 2019). For each state s, the error is calculated for the selected action a by comparing the predicted Q-value $Q_{\text{main}}(s,a)$, obtained from the primary Q-network with the Q-value $Q_{\text{outputs}}(s,a)$, computed using the Bellman update rule. The MSE loss is given by Eq. (21), where n_b is the batch size, s_i denotes the state for the i^{th} sample, and a_i is the selected action for that state.

$$\text{MSE} = \frac{1}{n_b} \sum_{i=1}^{n_b} \left(Q_{\text{main}}(s_i, a_i) - Q_{\text{outputs}}(s_i, a_i)\right)^2 \tag{21}$$

The Adam gradient descent algorithm is applied to minimize the MSE loss by updating the weights of the primary Q-network (Vieillard et al. 2020). The overall training process is illustrated in Algorithm 1. The proposed DQSMODE algorithm retains the core structure of the QSMODE algorithm, as shown in Fig. 8, with the Q-table replaced by a Q-network and the iterative training process integrated.

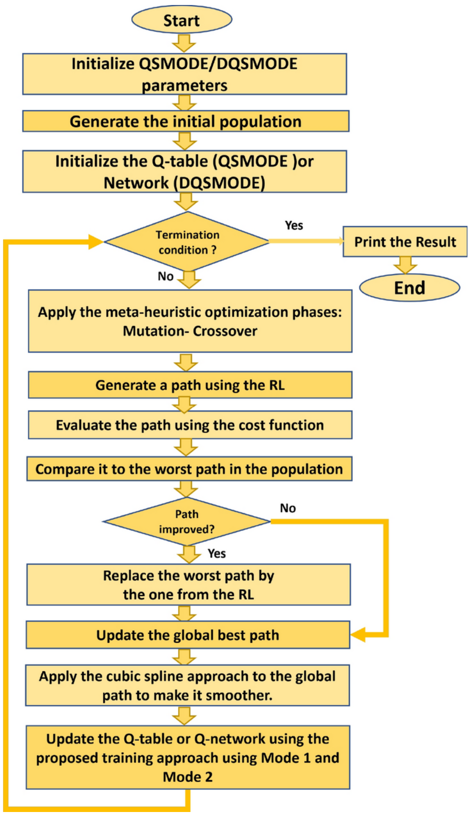

Start
Initialize QSMODE/DQSMODE parameters
Generate the initial population
Initialize the Q-table (QSMODE)or Network (DQSMODE)
Termination condition ?
Yes
Print the Result
End
No
Apply the meta-heuristic optimization phases: Mutation- Crossover
Generate a path using the RL
Evaluate the path using the cost function
Compare it to the worst path in the population
Path improved?
No
Yes
Replace the worst path by the one from the RL
Update the global best path
Apply the cubic spline approach to the global path to make it smoother.
Update the Q-table or Q-network using the proposed training approach using Mode 1 and Mode 2

Fig. 8 The flowchart of the proposed QSMODE/DQSMODE algorithm

```
1: Receive batch entries, main Q-network, and target Q-network.
2: for each entry in the batch $< s, a, r, s^n >$ do        ▷ Construct the training dataset
3:     Compute $Q_{\mathrm{main}}(s, a)$ for all actions at current state $s$ via main Q-net.
4:     Compute $Q_{\mathrm{target}}(s^n, a^n)$ for all actions at next state $s^n$ via target Q-net.
5:     Compute $Q_{\mathrm{outputs}}(s, a)$ for the selected action $a$ using the Bellman rule (Eq. (20)).
6:     Set $Q_{\mathrm{outputs}}(s, a = a*)$ for the remaining actions $\forall a^* \in A^*$ to $Q_{\mathrm{main}}(s, a)$.
7:     Append the current state $s$ to the inputs dataset.
8:     Append $Q_{\mathrm{outputs}}(s, a)$ and $Q_{\mathrm{outputs}}(s, a = a^*)$ to the outputs dataset.
9: end for
10: Calculate the Mean Squared Error (MSE) for all states in the batch (Eq.(21)).
11: Apply backpropagation using the Adam optimizer to update the main Q-network.
12: Return the trained Q-network.
```

Algorithm 1 Training process of the DQN network

5.3.6 Updating the Q-table/DQN in the QSMODE/DQSMODE algorithms

The primary objective of integrating the QSMODE and DQSMODE algorithms with reinforcement learning is to enable the proposed frameworks to learn and adapt to improve the optimization process. The training process differs slightly for QSMODE, which uses a Q-table, and DQSMODE, which uses a DQN network. Both processes are controlled by an exploration probability P_{rl} to dynamically balance exploration and exploitation.

The exploration probability P_{rl} governs the balance between two modes during the training process: Mode 1 for exploration and Mode 2 for exploitation. Notably, P_{rl} does not determine whether the Q-network or Q-table is updated but rather the updating mode. At every iteration, the reinforcement learning process generates a solution to accelerate convergence, particularly in real-time path planning scenarios.

Initially, P_{rl} is set to a value $P_{rl,\mathrm{init}}$ and decreases linearly over iterations until it reaches a minimum value $P_{rl,\mathrm{min}}$. The P_{rl} is updated using Eq. (22), where $P_{rl,\mathrm{init}}$ is the initial probability, $P_{rl,\mathrm{min}}$ is the minimum probability, *t* is the current iteration number, and $\mathrm{max\_iter}$ is the maximum number of iterations. This linear decay ensures a smooth transition from exploration to exploitation as training progresses.

$$P_{rl}^t = P_{rl,\mathrm{min}} + (P_{rl,\mathrm{init}} - P_{rl,\mathrm{min}}) \left(1 - \frac{t}{\mathrm{max\_iter}}\right) \tag{22}$$

A random number is generated and compared to P_{rl} at each iteration. If the random number is less than P_{rl}, Mode 1 is activated, focusing on exploration by randomly selecting actions. Otherwise, Mode 2 is activated, emphasizing exploitation by utilizing the population generated by the optimization process to train the Q-table or Q-network. Algorithm 2 illustrates the proposed RL-learning update approach for both QSMODE and DQSMODE, showing the interaction between exploration (Mode 1) and exploitation (Mode 2).

Mode 1 emphasizes exploration by randomly selecting actions $a \in A$ for the current state s to generate the next state s^n. Mode 2, on the other hand, focuses on exploitation by utilizing paths generated during the optimization process to train the Q-table or Q-network. For each waypoint s and its following waypoint s^n in the path, the action a is computed as the angle between s and s^n using Eq. (23).

$$\theta_r = \tan^{-1}\left(\frac{y_n - y_r}{x_n - x_r}\right) \tag{23}$$

The transition from s to s^n using the action a is evaluated with a reward $r(s, a)$ using Eq.(16), which considers the distance to the goal and obstacle penalties. In QSMODE, the transition tuple (s, a, r, s^n) is directly used to update the Q-table using the Q-learning rule in Eq. (19). In DQSMODE, the tuple (s, a, r, s^n) is stored in the replay memory, and the Q-network is trained using a subset of this memory.

Mode 1 is designed for exploration; it expands the search space by exploring new states and actions to ensure a diverse search space for reinforcement learning. On the other hand, Mode 2 is designed for exploitation; it exploits the optimization-generated data to train the model toward goal-oriented solutions, enhancing convergence efficiency. Moreover, there is a difference in the update mechanism between QSMODE and DQSMODE; QSMODE depends on direct Q-table update, while DQSMODE relies on replay memory and Q-network for training. Algorithm 2 integrates both modes, dynamically balancing exploration and exploitation using the switching probability P_{rl}.

```
1: Update $P_{rl}$ using Eq.(22).
2: Generate a random number $r \in [0, 1]$.
3: if $r < P_{rl}$ then                                          ▷ Mode 1: Exploration Phase
4:     Set the current state $s = (x_r, y_r)$ as the starting state.
5:     while Max training iterations $C1_{max}$ not reached do
6:         Select a random action $a \in A$, within the action set in Eq.(15).
7:         Compute the next state $s^n = (x_n, y_n)$ by applying action $a$ on state $s$ via Eq. (25).
8:         Evaluate the direct reward $r$ for the next state $s^n$ using Eq.(16).
9:         if QSMODE then
10:            Update the Q-table using Eq. (19).
11:        else if DQSMODE then
12:            Store the entry $(s, a, r, s^n)$ in replay memory.
13:            Obtain the batch of size $n_b$ as subset of the replay memory.
14:            Train the Q-network using the batch as in Algorithm 1.
15:            if Current iteration is a multiple of $N_{freq}$ cycles then
16:                Update the target Q-network by copying from the main Q-network.
17:            end if
18:        end if
19:        Set the next state $s^n$ as the current state $s$.
20:    end while
21: else                                                          ▷ Mode 2: Exploitation Phase
22:    for Each path in the population do
23:        for Each waypoints $i$ in the path do
24:            Set the current state $s$ to the first waypoint $(x_i, y_i)$.
25:            Set the next state $s^n$ to the following waypoint in the path $(x_{i+1}, y_{i+1})$.
26:            Compute the action angle $a = \theta_r$ using Eq.(23).
27:            Evaluate the direct reward $r$ for the next state $s^n$ using Eq.(16).
28:            if QSMODE then
29:                Update the Q-table using Eq. (19).
30:            else if DQSMODE then
31:                Store the entry $(s, a, r, s^n)$ in replay memory.
32:                Obtain the batch of size $n_b$ as subset of the replay memory.
33:                Train the Q-network using the batch as in Algorithm 1.
34:                if Current iteration is a multiple of $N_{freq}$ cycles then
35:                    Update the target Q-network by copying from the main Q-network.
36:                end if
37:            end if
38:            Set the next state $s^n$ as the current state $s$.
39:        end for
40:    end for
41: end if
42: return Updated Q-table or (Q-main, Q-target, and replay memory).
```

Algorithm 2 Training stage in QSMODE and DQSMODE (Mode 1 and Mode 2)

5.3.7 Generating the path using Q-learning in QSMODE/DQSMODE

In the QSMODE and DQSMODE algorithms, Q-learning is employed to generate a candidate path from the start point to the endpoint. The process begins by initializing the path with the start point and setting it as the current state $s = (x_r, y_r)$. The main loop continues until a path is found, the goal state is reached, or the maximum number of iterations is exceeded. At each iteration, the action a with the maximum Q-value for the current state s is selected, as in Eq. (24). This action represents the optimal rotation angle θ_r to transition from the current state s to the next state $s^n = (x_n, y_n)$. The next state is computed using Eq. (25).

$$a = \arg\max_{a \in A} Q(s, a) \tag{24}$$

$$\begin{aligned} x_n =& x_r + \cos(\theta_r), \\ y_n =& y_r + \sin(\theta_r), \end{aligned} \tag{25}$$

For QSMODE, the Q-table is directly used to determine the best action. In DQSMODE, the Q-network predicts the Q-values, and the action with the highest predicted value is chosen. The path is returned if the new state matches the goal state. Otherwise, the new state is appended to the path vector, set as the current state, and the loop proceeds. Algorithm 3 outlines the detailed steps for generating a path in both QSMODE and DQSMODE.

```
1: Initialize the current state $s = (x_r, y_r)$ as the start state.
2: Initialize the path vector with the start state.
3: while Termination condition not met ($C2_{max}$) do
4:     if QSMODE then
5:         Select action $a$ with the highest Q-value from the Q-table for $s$ using Eq.(24).
6:     else if DQSMODE then
7:         Predict Q-values by DQN for $s$ and select action $a$ with the highest value (Eq.(24)).
8:     end if
9:     Compute the next state $s^n = (x_n, y_n)$ using Eq.(25).
10:    if Goal state reached then
11:        return the path vector.
12:    else
13:        Append the new state $s^n$ to the path vector.
14:        Set $s^n$ as the current state $s$.
15:    end if
16: end while
17: return No Path Found.
```

Algorithm 3 Path generation using Q-learning in QSMODE/DQSMODE

5.3.8 The overall proposed QSMODE/DQSMODE algorithm

The QSMODE and DQSMODE algorithms enhance the original MODE framework by incorporating reinforcement learning and cubic spline interpolation to improve path quality and adaptability. RL integrates either Q-learning (QSMODE) or deep Q-networks (DQSMODE) for iterative learning, while cubic spline interpolation smoothens paths for better applicability in real-world scenarios. The high-level workflow of the algorithm is shown in Algorithm 4 and Fig. 8.

The algorithm begins with parameter initialization, including the Q-learning parameters and the Q-table or Q-network. The main loop iterates until the termination condition is met. In each iteration, standard optimization phases (mutation, crossover, and optional local search) are applied to improve the population. Next, a candidate path is generated using Q-learning (for QSMODE) or a DQN-based approach (for DQSMODE) using Algorithm 3. This path is evaluated using the defined cost function in Eq. (2), and if it improves upon the worst path in the population, it replaces it.

The global best path is updated after each iteration to refine the global solution, and cubic spline interpolation is applied to ensure smoothness. Finally, RL is applied to update the Q-table or train the Q-network using Algorithm 2, adapting the algorithm to learn from the population experience that enhances its convergence and solution quality.

```
1: Initialize the parameters for QSMODE/DQSMODE, including the initial population.
2: Initialize the Q-table (QSMODE) or Q-network (DQSMODE).
3: while Termination condition not met do
4:     Apply the mutation, crossover, and optional local search operators.
5:     Generate a path using Q-learning or DQN (Algorithm 3).
6:     Evaluate the path cost using Eq. (2).
7:     if The path cost improves the worst individual in the population then
8:         Replace the worst individual with the new path generated from Algorithm 3.
9:     end if
10:    Update the global best path.
11:    Apply cubic spline interpolation to smooth the global best path.
12:    Update the Q-table (QSMODE) or train the DQN (DQSMODE) using Algorithm 2.
13: end while
14: return Global best path.
```

Algorithm 4 The overall QSMODE/DQSMODE algorithm

6 Sensitivity and convergence analysis of the DQSMODE and Q-learning effect

This section examines the role of reinforcement learning (RL) in the DQSMODE algorithm, focusing on its impact on convergence and parameter tuning for the switching probability P_{rl}. The P_{rl} parameter governs the balance between exploration (Mode 1) and exploitation (Mode 2) in updating the Q-network/table. Importantly, P_{rl} does not control whether the Q-network/table is updated but rather dictates the mode of updating. Regardless of the value of P_{rl}, the RL process produces a solution at every iteration, ensuring accelerated convergence in real-time path planning scenarios.

6.1 Sensitivity analysis of P_{rl}

A sensitivity analysis was conducted on $P_{rl,init}$ ([0.1, 0.3, 0.5, 0.7, 0.9]) and $P_{rl,min}$ ([0, 0.01, 0.05, 0.1]) with 20 configurations tested on 50 benchmark problems from the RP2B-24 library. Each configuration was run independently 8 times, with a Friedman test performed on median error results. Results, visualized using heatmaps and bar charts (Fig. 9), show that the best configuration is $P_{rl,\text{init}} = 0.5$ and $P_{rl,\text{min}} = 0$, achieving the

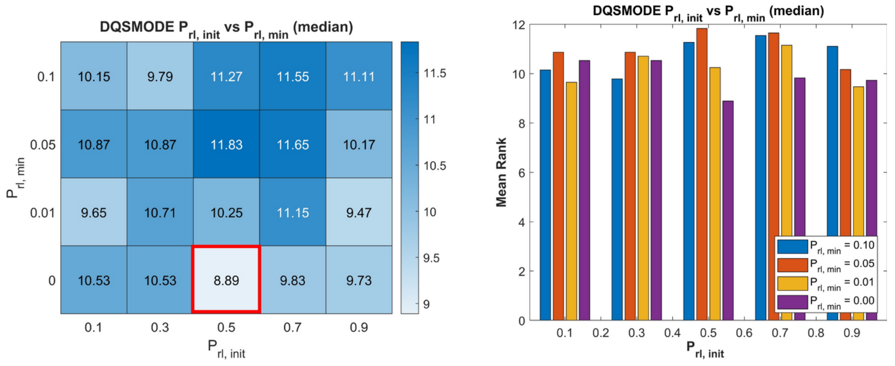


Fig. 9 The heatmap and the bar chart for the sensitivity analysis

minimum mean rank in the Friedman test. This configuration balances exploration (Mode 1) and exploitation (Mode 2) early in the learning process while prioritizing exploitation during later iterations.

Larger values of $P_{rl,min}$ result in higher mean ranks, indicating the importance of prioritizing Mode 2 during later stages for effective learning and convergence. As P_{rl} decreases linearly over iterations, the Q-table/network updates increasingly rely on Mode 2. Furthermore, the DQSMODE algorithm incorporates an archive to preserve population diversity, ensuring that the learning process remains robust and avoids premature convergence to local minima, even when $P_{rl,min} = 0$.

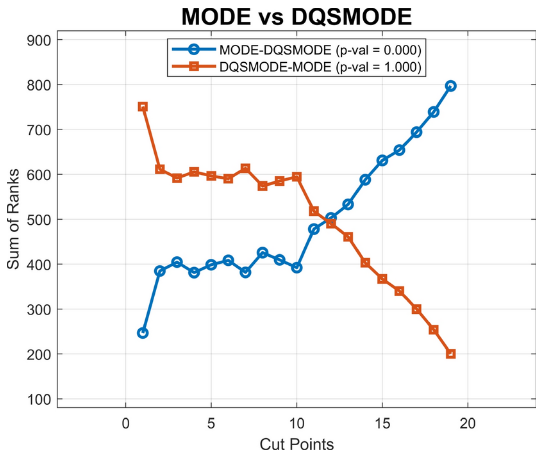
6.2 Convergence comparison: DQSMODE vs. MODE

Reinforcement learning plays a dual role in enhancing the DQSMODE algorithm. For unfamiliar scenarios, it accelerates convergence by deploying evolutionary operators (from differential evolution) alongside Q-learning updates. The combination of population-based search and reinforcement learning enables faster identification of optimal paths than traditional evolutionary algorithms. For familiar scenarios, the pre-trained Q-network/table can generate paths more efficiently by bypassing the need to explore the search space. It is highly effective in scenarios requiring long-term learning and rapid, accurate path planning.

The Page test results further validate the superior convergence of DQSMODE compared to the baseline MODE algorithm. The test examines the median error differences over $N = 19$ cut points across iterations and $k = 10$ benchmark problems (Carrasco et al. 2020). Statistical significance (p-value < 0.05) and an increasing trend in performance differences demonstrate that DQSMODE achieves faster convergence rates, as shown in Fig. 10 (More details are in the supplementary materials, Sect. S3). This advantage is due to the RL-guided exploitation of learned knowledge, which directs the optimization process more effectively than random evolutionary exploration.

An additional strength of DQSMODE is its hybrid capability, combining the general problem-solving nature of evolutionary algorithms with the learning-based adaptability of reinforcement learning. While traditional machine-learning models rely on pre-existing training data and struggle with unknown environments, and conventional evolutionary algorithms require more time and rely on random search strategies for unfamiliar scenarios,

Fig. 10 Page test convergence trend graph between the DQSMODE and MODE algorithms



DQSMODE excels in both. It can effectively address entirely new problems while simultaneously retaining and leveraging knowledge for similar future tasks.

This flexibility allows DQSMODE to function as both a general solver for unknown challenges and a specialized solver for recurring scenarios. The convergence plots in Fig. 11 (and additional figures in the supplementary materials, Figure S3) further highlight this strength, demonstrating DQSMODE's ability to generate solutions almost instantaneously. In cases where prior learning is applicable, it effectively transforms the problem into a near-deterministic one. This adaptability positions DQSMODE as an ideal solution for real-time path planning, offering a unique combination of speed, accuracy, and long-term learning within a single robust framework.

7 Parameters settings of the path planning algorithms

In this validation phase, the algorithm will be validated on path-planning problems. DQSMODE and QSMODE will be compared with six other path-planning algorithms in the literature to evaluate its performance in path-planning scenarios, including A*. Dijkstra, RRT, RRT*, APF, and DWA algorithms. The DQSMODE and QSMODE will also be compared with 16 meta-hectic optimization algorithms for the path-planning problems, including GA, DE, SA, PSO, ABC, CSO, CSOA, FA, BAS, TLBO, GWO, MFO, WOA, HHO, CMAES, and IMODEII algorithms. Table 1 summarizes the parameters settings recommended in the original work for all the 22 state-of-the-art path planning algorithms used in the comparison.

The search space for all 50 benchmark problems varies between -10 and 10 in the x- and y-coordinates. The path-planning algorithms are validated in two stages. First, the DQSMODE and QSMODE are compared with 16 meta-heuristic optimization algorithms. All the meta-heuristic optimization algorithms have the same principle of operation, where the termination condition is set to a maximum function of evaluations (maxFEs) equals 10000 multiplied by the number of obstacles of the model.

Second, the DQSMODE and QSMODE are compared with the six path-planning algorithms (A*. Dijkstra, RRT, RRT*, APF, and DWA). These algorithms work differently to generate the path. Therefore, the primary termination condition is based on convergence, where the algorithm terminates when the best path is not improving for 100 successive iterations.

A trial run is performed for each algorithm for all 50 path planning scenarios to estimate the maximum number of iterations required to generate a path for each problem. The A*, RRT, and RRT* require 12000 iterations, the APF requires 50000 iterations, the Dijkstra needs 25000 iterations, and the DWA algorithm requires 1000 iterations. The DQSMODE,

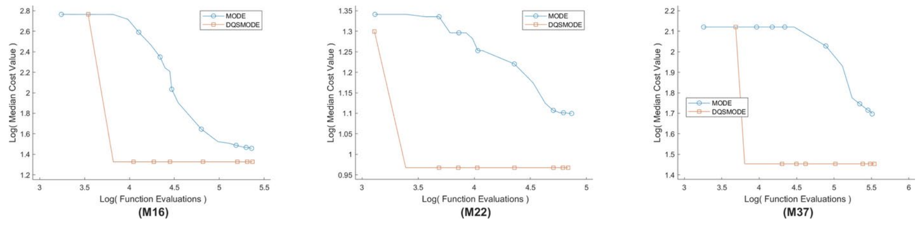


Fig. 11 Convergence plots after Q-learning for the median error between MODE and DQSMODE

Table 1 Summary of parameter settings for path planning algorithms

Algorithm	Parameters
A*	Grid resolution = 0.15
Dijkstra	Grid resolution = 0.15
RRT	Step size = 0.15
RRT*	Step size = 0.15, rewire radius = 1.5 * stepSize
APF Yao et al. (2020)	Step Size Δt = 0.005, attractive force coeff. k_{att} = 1
	repulsive force coeff. k_{rep} = 20, influence range of repulsive force ρ_0 = 3
DWA Kong and Cheng (2023). Chang et al. (2021)	v_{max} = 1, v_{min} = −0.5, linear velocity resolution =0.01
	ω_{max} = $\pi/4$, ω_{min} =$-\pi/4$, angular velocity resolution = 0.1, Δt = 0.1
GA Koohi et al. (2018)	selection: tournament, tournament size = 3
	P_c = 0.8, P_m =0.3, mutation rate μ = 0.02
DE Simon (2008)	*CR* = 0.5, *F* = 0.5
SA Heris (2015), Fischetti and Stringher (2019)	T_0 = 0.1, α = 0.99, mutation rate μ = 0.5
	No. of neighbors per individual = 5, max no. of sub-iteration = 20
PSO Heidari et al. (2019), Simon (2008)	*w* = 0.3, c_1 = 1, c_2 = 1
ABC Koohi et al. (2018)	*a* = 1, No. of onlooker bees = popSize
	abandon limit = 0.6 * dimSize * popSize
CSO Meng et al. (2014)	roosters *NR* = 20%, hens *NH* = 60%, chicks *NC* = 20%
	mothers *NM* = 10%, *G* = 10%, FL_{min} = 0.5, FL_{max} = 0.9
CSOA Thomas et al. (2018)	*c* = 2.05, w_{max} = 0.9, w_{min} = 0.4, *MR* = 0.8, *SMP* = 15
	CDC = 0.8, *SRD* = 0.1, initial cat velocity = 25% of problem bounds
FA Koohi et al. (2018)	γ = 1, β_0 = 2, α = 0.2, damping ratio = 0.98
	exponential constant *m* = 2, uniform mutation range = 0.05 * (ub - lb)
BAS Jiang and Li (2017)	initial step size δ^1 = 0.5, η_δ = 0.95, δ_0 = 0.00
	initial antennae length d^1 = 2, η_d = 0.95, d_0 = 0.01
TLBO Rao et al. (2011)	teacher factor *TF* = 1 or 2 (randomly chosen)
GWO Mirjalili et al. (2014)	convergence parameter *a*: linearly decreases from 2 to 0
MFO Mirjalili (2015)	*b* = 1, convergence parameter *a*: linearly decreases from -2 to -1
WOA Mirjalili and Lewis (2016)	*b* = 1, convergence parameter *a*: linearly decreases from 2 to 0
HHO Heidari et al. (2019)	escaping energy randomly varies between [-2,2]
CMAES Hansen and Ostermeier (2001)	μ = *popSize*/2, damping rate $d_\sigma = c_\sigma^{-1} + 1$
	$C_c = \frac{4}{nd+4}$, $c_{cov} = \frac{2}{(nd+\sqrt{2})^2}$
IMODEII Sallam et al. (2022)	$N_{init} = 10 * nd$, $N_{min} = 4$, $CR_{init} = F_{init} = 0.2$
	A = 1.4, H = 18 $* nd$, P_{ls} = 0.1
	for RL: γ = 0.85, α = 0.25
QSMODE	maximum spline points (ns) = 100, number of actions n_a = 8
	α = 0.6, γ = 0.9, P_{rl} = 0.5 to 0
	$C1_{max}$ = 20, $C2_{max}$ = 500, $x_{res} = y_{res}$ = 0.15
DQSMODE	maximum spline points (ns) = 100, number of actions n_a = 8
	α = 0.6, γ = 0.9, P_{rl} = 0.5 to 0
	$C1_{max}$ = 20, $C2_{max}$ = 500, $x_{res} = y_{res}$ = 0.15
	Replay memory size n_m = 1000, Batch size n_b = 32,, N_{freq} = 5
Population size	nPop = 30, all meta-heuristic alg. except CMAES, IMODEII, (D)QSMODE

Table 1 (continued)

Algorithm	Parameters
Models	Number of models (scenarios) = 50
	Range of the x-coordinate = [-10,10]
	Range of the y-coordinate = [-10,10]
Termination Cond.	Convergence (successive number of iterations) = 100
	Max. function evaluations (MaxFEs) = 10000 * max(nObstacles, 1)

QSMODE and the other meta-heuristic optimization algorithms need 500 iterations to guarantee a path for all the problems.

8 Simulation results part 1: meta-heuristic optimization algorithms

The DQSMODE and QSMODE algorithms are compared with 16 other state-of-the-art meta-heuristic optimization algorithms: GA, DE, SA, PSO, ABC, CSO, CSOA, FA, BAS, TLBO, GWO, MFO, WOA, HHO, CMAES, and IMODEII algorithms. All these algorithms have been executed for 30 independent runs on the 50 path-planning problems of the RP2B-24 library. All the paths generated from each algorithm have been evaluated using the same cost function proposed in Eq. (2).

8.1 Results collection and graphical representation via box and violin plots

The best, worst, mean, median, and standard deviation (SD) metrics for the path costs generated by each algorithm across 30 runs have been calculated for all 50 path-planning models (M1 to M50). These results are included in the Supplementary Materials (Table S3). The violin plots for the models (M14, M20, M37, and M46) are shown in Fig. 12, providing a concise visual representation of the results. These models are selected to demonstrate diverse scenarios, including multiple obstacles (M14), narrow passages (M20 and M37), and maze-like environments (M46). The BAS, CMAES, and PSO algorithms have been excluded because their costs are higher than those of the other algorithms, causing the plots to be unclear for the rest of the algorithms.

The violin plots highlight the distribution of results for each algorithm. The black line within each violin represents the mean value, while the dashed red line indicates the median (Molina et al. 2022). The QSMODE and DQSMODE algorithms show compact distributions with tightly aligned mean and median values, indicating high repeatability and reliability. The violin plots clearly demonstrate the ability of QSMODE and DQSMODE to produce consistent and repeatable results across varied path-planning challenges. Full results, including all box and violin plots for the 50 models, are available in the Supplementary Materials (Figs. S4 and S5).

8.2 Statistical analysis of the results: group comparisons

The performance of the DQSMODE and QSMODE algorithms are evaluated using the Friedman test and the SNE-SR ranking method to compare all 18 algorithms across 50 path-planning models, providing a comprehensive evaluation of the algorithms, ranking

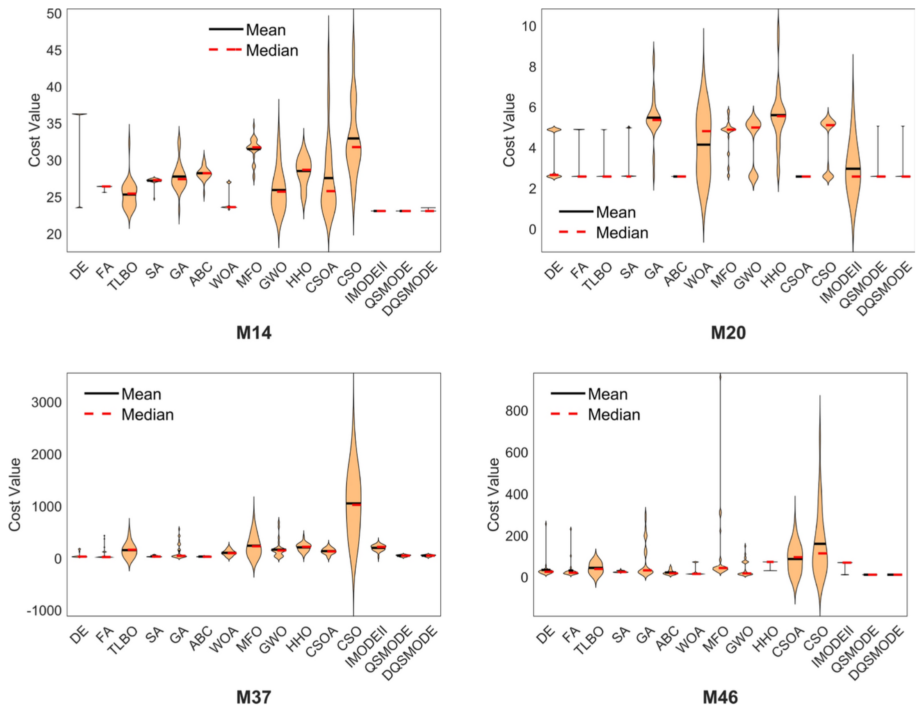


Fig. 12 Violin plots for all the algorithms in 30 runs for M14, M20, M37, and M46

them based on median and mean path cost metrics. The Friedman test results, summarized in Table 2, show the ranks of the algorithms based on median costs (Derrac et al. 2011). The DQSMODE ranks first with a mean rank of 2.71, followed by QSMODE at 3.11. The top seven algorithms are DQSMODE, QSMODE, IMODEII, TLBO, ABC, DE, and FA. The p-value in the Friedman test ($2.12E-114$) confirms the statistical significance of these rankings, emphasizing the superior performance of the proposed algorithms (QSMODE and DQSMODE).

The SNE-SR ranking method evaluates the algorithms using two metrics: sum normalized best error/cost (SNE) and mean cost sum ranks (SR), each contributing 50% to the total score, as introduced (Yue et al. 2019; Kumar et al. 2021). DQSMODE achieves the highest total score (99.29%), followed by QSMODE (93.05%) and IMODEII (77.82%), as shown in Table 2. The top seven algorithms align with those obtained from the Friedman test, further emphasizing the significant performance of DQSMODE and QSMODE.

8.3 Statistical analysis of the results: paired comparisons

The Wilcoxon test was conducted for paired comparisons between the proposed DQSMODE algorithm and each of the other algorithms across 50 models. This test evaluates not only which algorithm has better mean values but also the magnitude of the differences (Divine et al. 2013). Positive differences (+) indicate that DQSMODE performs better, negative differences (−) indicate the other algorithm is better, and ties (=) show no significant difference.

Table 2 Statistical analysis for all 50 models for all the 18 meta-heuristic optimization algorithms

Alg.	Friedman test			SNE-SR ranking test				Wilcoxon test				
	sumR	meanR	rank	Score1	Score2	Score	Rank	Sign Test	R+	R-	p-val	H
DE	323.5	6.47	6	48.7623	20.9428	69.7051	6	+45/=1/-4	1158	67	5.76E-08	TRUE
FA	331	6.62	7	49.0671	20.4683	69.5354	7	+47/=0/-3	1242	33	5.37E-09	TRUE
TLBO	309	6.18	4	49.6505	21.9256	71.5761	4	+44/=0/-6	1208	67	3.65E-08	TRUE
SA	397	7.94	8	47.5896	17.0655	64.6551	10	+43/=0/-7	1162	113	4.12E-07	TRUE
GA	611	12.22	13	48.4491	11.0884	59.5375	12	+50/=0/-0	1275	0	7.56E-10	TRUE
CMAES	853	17.06	18	38.7905	7.9426	46.7331	17	+50/=0/-0	1275	0	7.56E-10	TRUE
PSO	783.5	15.67	16	44.3882	8.6471	53.0353	16	+48/=1/-1	1224	1	1.18E-09	TRUE
BAS	840	16.8	17	22.5147	8.0655	30.5802	18	+50/=0/-0	1275	0	7.56E-10	TRUE
ABC	314	6.28	5	48.1314	21.5764	69.7079	5	+43/=0/-7	1119	156	3.35E-06	TRUE
WOA	540	10.8	11	48.5891	12.5463	61.1354	11	+48/=0/-2	1267	8	1.23E-09	TRUE
MFO	598.5	11.97	12	46.8354	11.3200	58.1553	14	+47/=1/-2	1220	5	1.51E-09	TRUE
GWO	403	8.06	9	49.5683	16.8114	66.3797	8	+47/=0/-3	1253	22	2.82E-09	TRUE
HHO	634	12.68	15	45.6522	10.6861	56.3383	15	+49/=0/-1	1273	2	8.53E-10	TRUE
CSOA	445	8.9	10	49.6661	15.2247	64.8908	9	+42/=0/-8	1176	99	2.01E-07	TRUE
CSO	633	12.66	14	47.6333	10.7030	58.3363	13	+49/=0/-1	1224	51	1.18E-09	TRUE
IMODEII	243.5	4.87	3	50.0000	27.8234	77.8234	3	+29/=1/-20	899	326	4.37E-03	TRUE
QSMODE	155.5	3.11	2	49.4772	43.5691	93.0463	2	+31/=1/-18	707	518	2.22E-01	FALSE
DQSMODE	135.5	2.71	1	49.2877	50.0000	99.2877	1	NA	NA	NA	NA	NA
	p-val	2.12E-114										

In the Friedman test, the best algorithm is the one with the minimum mean rank. In SNE-SR ranking test, SNE is the sum of normalized error, SR is the sum rank, the score is out of 100, and the best algorithm is the one with the highest score. In Wilcoxon Test, DQSMODE is the reference algorithm in all paired comparisons. '+' means the number of models in which the DQSMODE is better, and '=' means draw. R+ > R- means the DQSMODE is better. The significance level α is between 0.05 and 0.1. The results are significant if p-value $< \alpha$

Differences are ranked, with the smallest receiving rank 1 and the largest receiving rank 50. The ranks are summed for positive ($R+$) and negative ($R-$) differences, and the p-value is calculated to determine statistical significance at a threshold $\alpha = 0.05$ (Woolson 2007). Table 2 shows that DQSMODE consistently achieves a p-value below 0.05, significantly outperforming all other algorithms, including IMODEII. Notably, there is no significant difference between DQSMODE and QSMODE, confirming their similar performance.

Figures 13 and 14 illustrate the detailed Wilcoxon test results. From an algorithm perspective, DQSMODE demonstrates superior performance across the 50 models, achieving consistently better costs in multi-obstacle, narrow-passage, and maze-like scenarios. In open-field cases, DQSMODE ties with competitive algorithms such as DE, PSO, MFO, IMODEII, and QSMODE.

For single-obstacle cases, DQSMODE outperforms most algorithms, achieving significant wins against GA, CMAES, and BAS. In multi-obstacle scenarios, DQSMODE excels in all 13 models, outperforming IMODEII in 10 models. In comparison, QSMODE outperforms DQSMODE in 7 multi-obstacles models. For narrow passages, DQSMODE is in the lead for most cases, with QSMODE and IMODEII as its closest competitors. In maze-like cases, DQSMODE outperforms all algorithms in 12-13 cases, with IMODEII performing slightly better in some scenarios. These results confirm the robustness and reliability of DQSMODE in various path-planning scenarios, especially in challenging environments with obstacles.

8.4 Convergence graphs

Figure 15 illustrates the convergence behavior of the top seven algorithms (DQSMODE, QSMODE, IMODEII, TLBO, ABC, DE, and FA) on models representing diverse scenarios: multiple obstacles (M12), narrow passages (M25 and M34), and maze-like environments (M43). The vertical axis displays the logarithmic median cost across 30 runs, while the horizontal axis represents the logarithmic value of function evaluations, indicating algorithm iterations.

The DQSMODE algorithm shows a superior convergence performance compared to the top seven algorithms, where it starts with the highest cost compared with the other algorithms. Then, it improves until it reaches a cost equal to or better than the other algorithms. The DQSMODE algorithm demonstrates superior convergence performance, starting with the highest initial cost compared to the other algorithms and rapidly improving to achieve a final cost equal to or better than its competitors.

In most models, the algorithm can achieve low cost in shorter function evaluations, showing its fast convergence. QSMODE shows a similar convergence pattern, which DQSMODE slightly advantages. The convergence performance of the DQSMODE and QSMODE are balanced, as they are sometimes slower than the other algorithms in complex scenarios to avoid premature convergence, resulting in superior final costs compared to other algorithms. Convergence graphs for all 50 models are available in Supplementary materials (Figure S6) for further analysis.

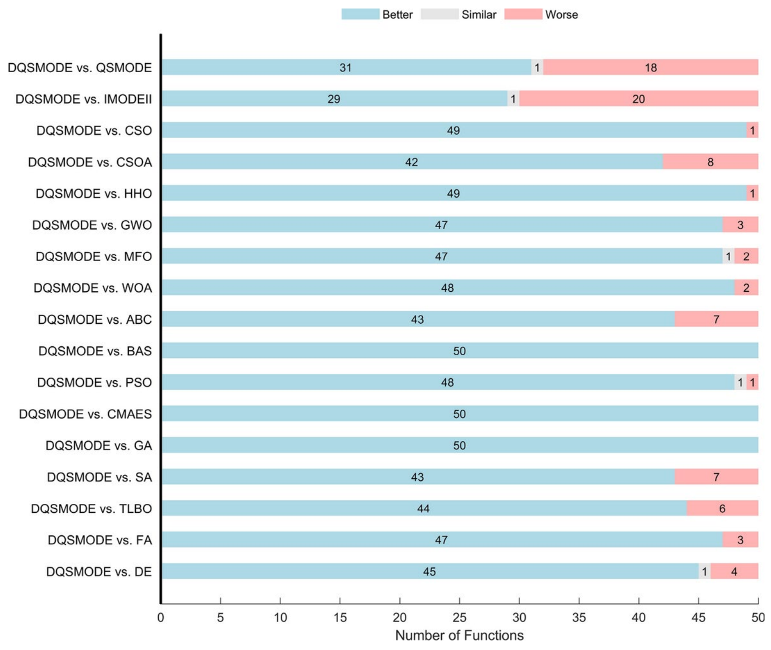


Fig. 13 Detailed Wilcoxon test results distributions for all the 50 models for the algorithms vs. the DQSMODE algorithm (Algorithm perspective)

8.5 Path plots

This section shows the path found by each algorithm, from the start point to the goal, in some selected path planning problems (M22 and M49), as shown in Fig. 16. In this figure, the median path for each algorithm is plotted to visualize the actual path. These plots give an intuition about the feasibility of the path because the cost of the path is not enough to evaluate the path's feasibility and smoothness, and any collision can be noticed as well.

Figure 16a shows the paths for a narrow-passage case (M22). The QSMODE algorithm got the best median path with a cost of 9.2716, and DQSMODE got the second best path with a cost of 9.2734, where these paths are smooth compared to the ones obtained by the other algorithms. DE, FA, IMODEII, and TLBO algorithms achieved similar routes with challenging costs of 12.3940, 12.4140, 123940, and 12.3940, respectively. However, these paths have higher costs and are sharper than those obtained by the DQSMODE and QSMODE algorithms.

Figure 16b is a challenging maze-like scenario with a larger number of obstacles. This scenario shows the superiority of the DQSMODE algorithm in dealing with one of the most challenging scenarios, achieving the best path with a cost of 9.9618, followed by the QSMODE with a cost of 9.9623, both maneuvering through the maze. The BAS, CMAES, CSO, CSOA, GWO, HHO, IMODEII, MFO, PSO, and WOA algorithms solved the maze by hitting the obstacles. The GA algorithms found an overlapping route, which is collision-free but infeasible with higher distance due to cycles. The ABC, DE, FA, SA, and TLBO algorithms solved the maze by leaving it and returning following an outer route. These results prove the superiority of the QSMODE and DQSMODE algorithms compared with

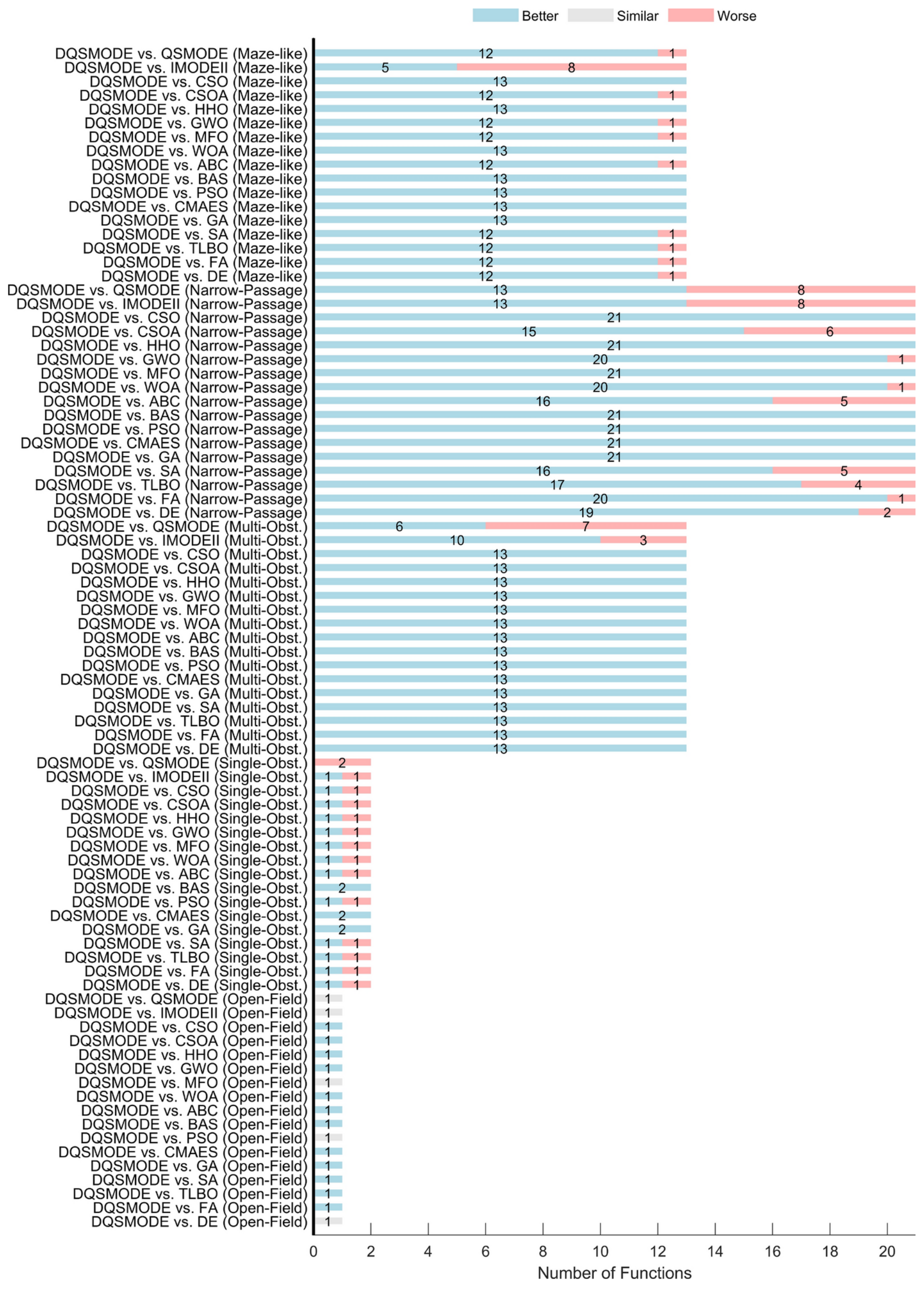


Fig. 14 Detailed Wilcoxon test results distributions for all the 50 models for the algorithms vs. the DQSMODE algorithm for each category of the models (Model categories perspective)

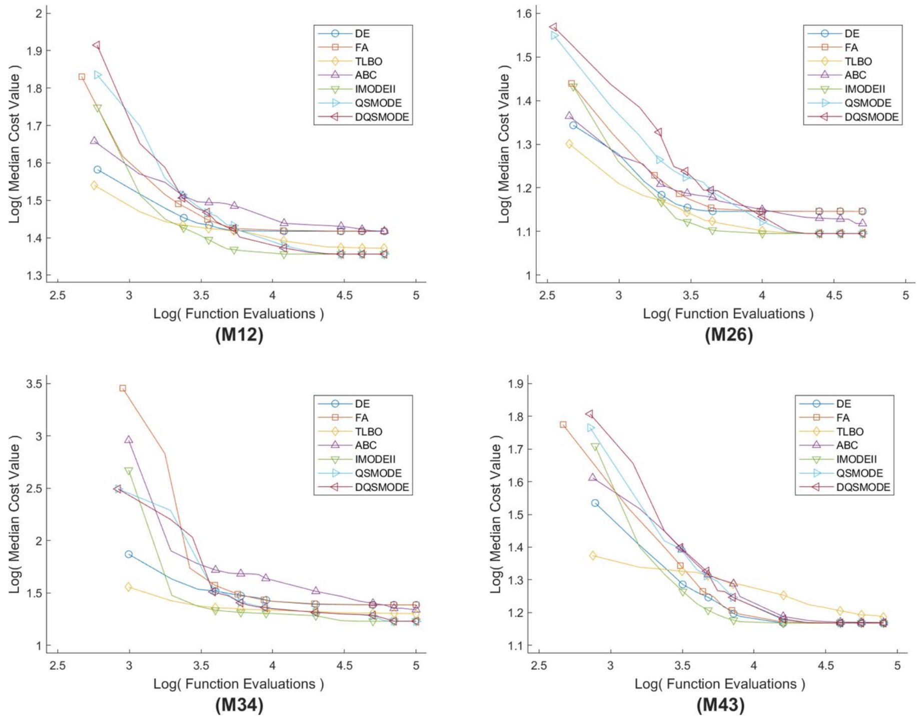


Fig. 15 Convergence plots for the median error of the top seven algorithms in 30 runs for M12, M26, M34, and M43

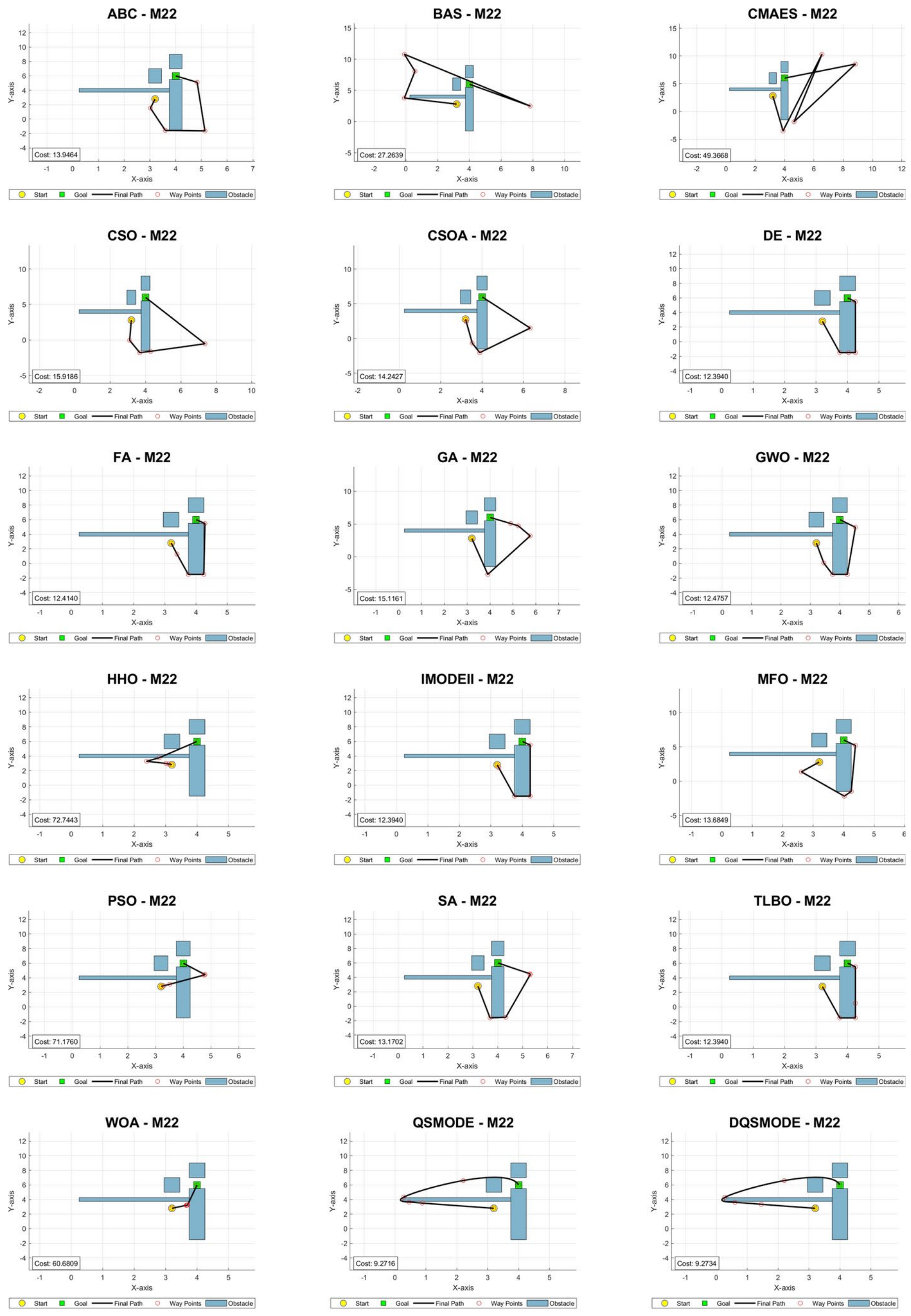
the other algorithms in the literature by achieving the best cost and a feasible smooth path. The path plots for all the 50 models obtained by all the algorithms are presented in the supplementary Materials (Fig. S9).

9 Simulation results part 2: path-planning algorithms

The QSMODE and DQSMODE algorithms are compared with state-of-the-art algorithms: A*, Dijkstra, RRT, RRT*, DWA, and APF algorithms. All these algorithms have been executed for 30 independent runs on the 50 path-planning problems of the RP2B-24 library. All the paths generated from each algorithm have been evaluated using the same cost function proposed in Eq. (2).

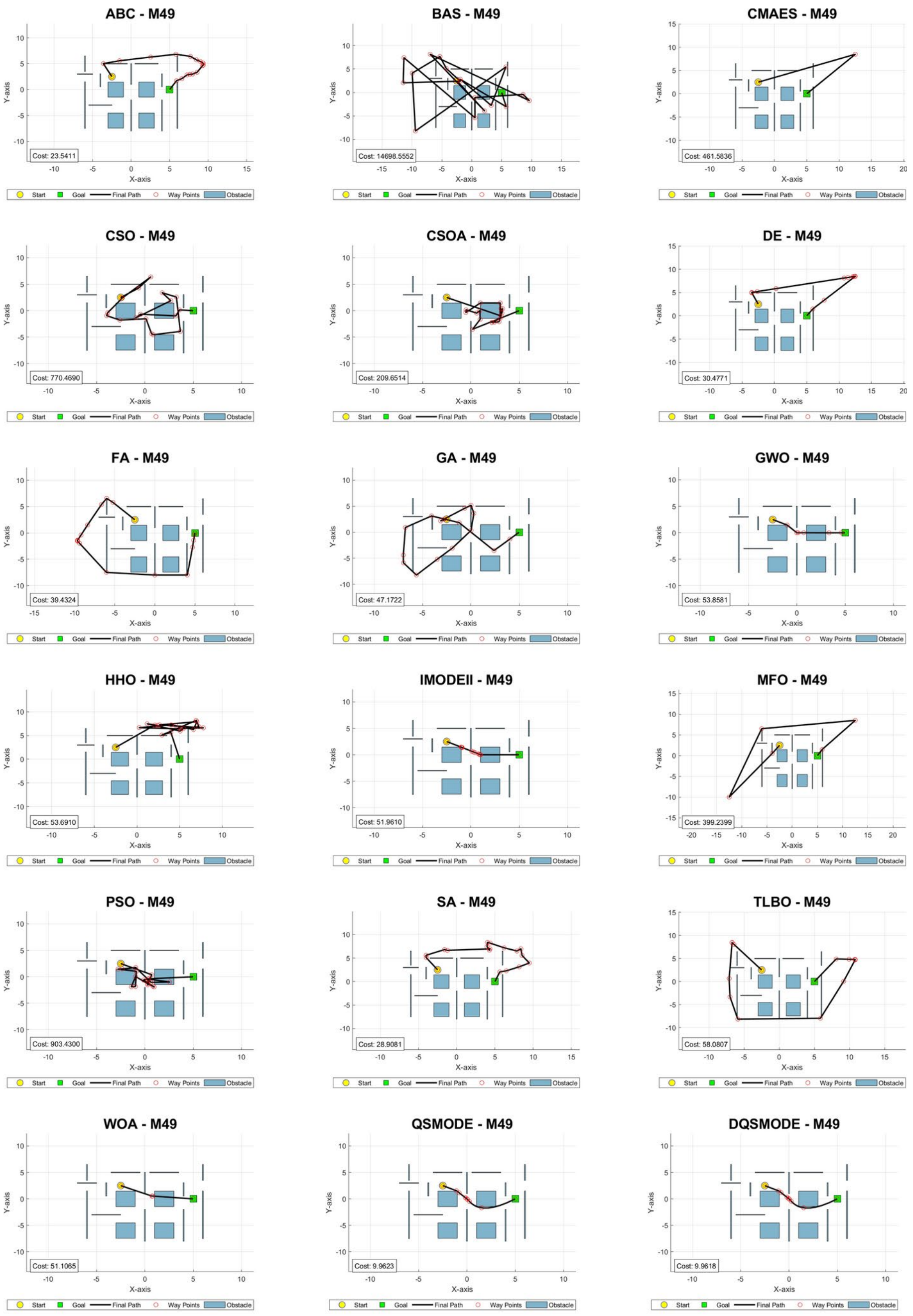
9.1 Results collection and graphical representation via box and violin plots

The results of the eight path-planning algorithms (A*, Dijkstra, RRT, RRT*, DWA, APF, QSMODE, and DQSMODE) are collected for all 50 models from M1 to M50 across the 30 runs. These results are provided in the Supplementary Materials (Table S6). The violin plots for the models (M14, M26, M34, and M43) are shown in Fig. 17, providing a concise visual representation of the results. These models are selected to demonstrate diverse scenarios,



(a) M22 Model

Fig. 16 Median Path plots for all the 18 meta-heuristic optimization algorithms. Median Path plots for all the 18 meta-heuristic optimization algorithms



(b) M49 Model

Figure 16 (continued)

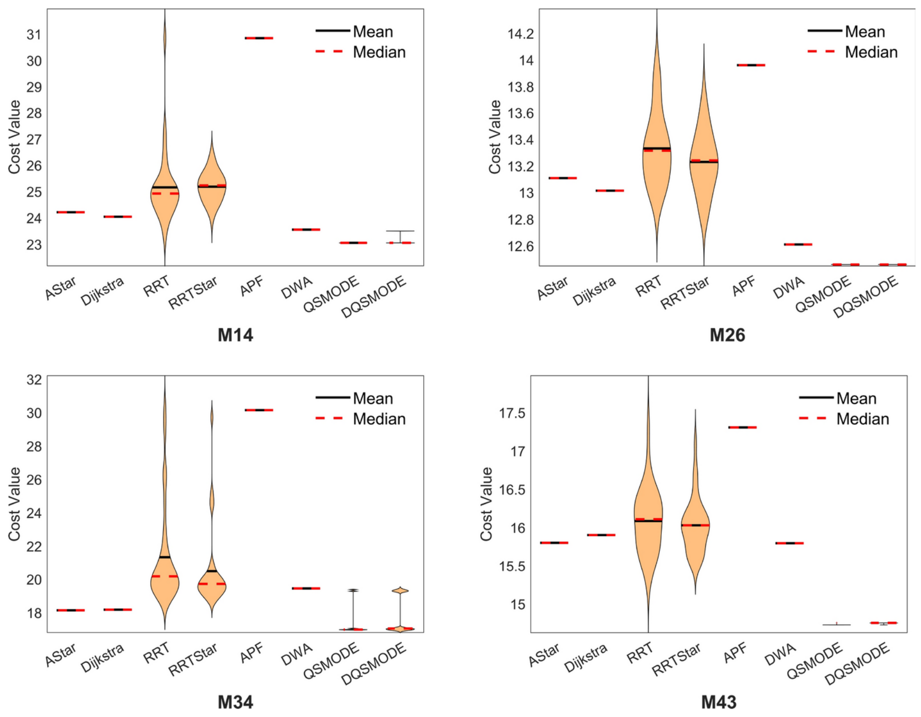


Fig. 17 Violin plots for all the 8 path-planning algorithms in 30 runs

including multiple obstacles (M14), narrow passages (M26 and M34), and maze-like environments (M43).

The plots show that the proposed QSMODE and DQSMODE algorithms have the best compact distribution of the results across the 30 runs compared to the other six algorithms. It shows a high degree of repeatability similar to the A* and Dijkstra algorithms, but QSMODE and DQSMODE outperform them with better path costs and fewer path waypoints. Full results, including all box and violin plots for the 50 models, are available in the Supplementary Materials (Figs. S7 and S8).

Table 3 shows the success rate for each algorithm across the 30 runs and the best number of waypoints for each model. The A*, Dijkstra, RRT, RRT*, QSMODE, and DQSMODE algorithms achieved a 100% success rate to obtain a path in all 30 runs for all 50 models. The APF algorithm achieved a 100% success rate in only M2 and M11. APF achieved variable success rates of 50%, 80%, 90%, and 86.67% for the models M1, M4, M6, and M18, while it failed in all the other models with a 0% success rate. The DWA algorithm achieved a 100% success rate for all models of types: open-field, single obstacle, multiple obstacles, and maze-like models. In contrast, DWA achieved 0% success rate for some of the narrow passages models M19, M21, M22, M24, M25, M30, M37, and M38.

The QSMODE and DQSMODE algorithms show the minimum number of path waypoints in all the models compared to the other algorithms. The RRT and RRT* come third and fourth after the proposed algorithms, with slightly fewer points for the RRT* algorithms than the RRT algorithm. The Dijkstra and A* algorithms come in the fifth and sixth places,

Table 3 Success rate (SR) and the best number of waypoints (n) for eight algorithms across 30 runs for all models (M1-M50)

M	SR/n	A*	DJ	RRT	RRT*	APF	DWA	QS DQS
M1	SR	100	100	100	100	50	100	100
	n	136	136	30	28	0	272	3
M2	SR	100	100	100	100	100	100	100
	n	51	51	27	26	5	142	3
M3	SR	100	100	100	100	0	100	100
	n	54	54	26	28	0	148	3
M4	SR	100	100	100	100	80	100	100
	n	92	92	36	32	0	232	4
M5	SR	100	100	100	100	0	100	100
	n	102	102	35	31	0	238	5
M6	SR	100	100	100	100	90	100	100
	n	49	48	26	25	0	142	5
M7	SR	100	100	100	100	0	100	100
	n	155	155	41	40	0	357	7
M8	SR	100	100	100	100	0	100	100
	n	70	70	27	28	0	185	6
M9	SR	100	100	100	100	0	100	100
	n	157	157	44	39	0	360	6
M10	SR	100	100	100	100	0	100	100
	n	101	101	37	34	0	239	6
M11	SR	100	100	100	100	100	100	100
	n	98	98	34	31	6	224	7
M12	SR	100	100	100	100	0	100	100
	n	119	119	34	32	0	293	8
M13	SR	100	100	100	100	0	100	100
	n	168	168	44	40	0	364	7
M14	SR	100	100	100	100	0	100	100
	n	125	125	33	32	0	296	9
M15	SR	100	100	100	100	0	100	100
	n	130	130	38	34	0	309	11
M16	SR	100	100	100	100	0	100	100
	n	125	125	38	35	0	279	13
M17	SR	100	100	100	100	0	100	100
	n	23	23	13	12	0	110	5
M18	SR	100	100	100	100	86.67	100	100
	n	93	93	25	29	0	214	6
M19	SR	100	100	100	100	0	0	100
	n	29	29	21	20	0	0	4
M20	SR	100	100	100	100	0	100	100
	n	36	36	15	15	0	104	4
M21	SR	100	100	100	100	0	0	100
	n	35	35	22	21	0	0	5
M22	SR	100	100	100	100	0	0	100
	n	90	90	41	32	0	0	6
M23	SR	100	100	100	100	100	100	100
	n	78	78	30	29	5	185	6

Table 3 (continued)

M	SR/n	A*	DJ	RRT	RRT*	APF	DWA	QS DQS
M24	SR	100	100	100	100	0	0	100
	n	57	57	22	26	0	0	7
M25	SR	100	100	100	100	0	0	100
	n	50	50	30	25	0	0	7
M26	SR	100	100	100	100	0	100	100
	n	75	75	29	27	0	189	7
M27	SR	100	100	100	100	0	100	100
	n	90	90	35	32	0	217	11
M28	SR	100	100	100	100	0	100	100
	n	69	69	30	30	0	559	7
M29	SR	100	100	100	100	0	100	100
	n	70	70	30	24	0	564	8
M30	SR	100	100	100	100	0	0	100
	n	55	55	35	31	0	0	9
M31	SR	100	100	100	100	0	100	100
	n	122	122	32	33	0	470	8
M32	SR	100	100	100	100	0	100	100
	n	89	89	27	26	0	212	10
M33	SR	100	100	100	100	0	100	100
	n	89	89	37	36	0	587	14
M34	SR	100	100	100	100	0	100	100
	n	108	108	35	36	0	256	12
M35	SR	100	100	100	100	0	100	100
	n	101	95	30	27	0	253	16
M36	SR	100	100	100	100	0	100	100
	n	41	41	16	20	0	152	12
M37	SR	100	100	100	100	0	0	100
	n	74	73	30	33	0	0	16
M38	SR	100	100	100	100	0	0	100
	n	79	78	35	33	0	0	4
M39	SR	100	100	100	100	0	100	100
	n	95	95	34	31	0	227	8
M40	SR	100	100	100	100	0	100	100
	n	72	69	28	28	0	189	8
M41	SR	100	100	100	100	0	100	100
	n	88	82	28	28	0	206	12
M42	SR	100	100	100	100	0	100	100
	n	72	72	30	28	0	191	10
M43	SR	100	100	100	100	0	100	100
	n	89	89	32	29	0	222	10
M44	SR	100	100	100	100	0	100	100
	n	85	85	31	23	0	863	14
M45	SR	100	100	100	100	0	100	100
	n	107	107	34	30	0	254	16
M46	SR	100	100	100	100	0	100	100
	n	61	61	24	21	0	174	16

Table 3 (continued)

M	SR/n	A*	DJ	RRT	RRT*	APF	DWA	QS DQS
M47	SR	100	100	100	100	0	100	100
	n	63	62	19	18	0	203	16
M48	SR	100	100	100	100	0	100	100
	n	79	79	23	25	0	206	19
M49	SR	100	100	100	100	0	100	100
	n	60	59	21	21	0	189	19
M50	SR	100	100	100	100	0	100	100
	n	69	69	22	17	0	195	22

Here, DJ refers to Dijkstra, QS is the proposed QSMODE, and DQS is the proposed DQSMODE

where the number of waypoints of the A* and Dijkstra algorithms is the same in all models except M6, M35, M37, M38, M40, M41, M47, and M49 with slightly more waypoints for the A* compared to Dijkstra. The DWA algorithm has the maximum number of waypoints; for example, the number of waypoints in M18 is 6 and 214 for the QSMODE/DQSMODE and DWA, respectively.

9.2 Statistical analysis of the results: group comparisons

The statistical analysis for all eight algorithms (A*, Dijkstra, RRT, RRT*, DWA, APF, QSMODE, and DQSMODE) are summarized in Table 4. The Friedman test results show that the DQSMODE ranks first with a mean rank of 2.31, followed by QSMODE at 2.69. The ranks of the eight algorithms are DQSMODE, QSMODE, Dijkstra, DWA, A*, RRT*, RRT, and APF. The p-value in the Friedman test ($1.06203E-36$) confirms the statistical significance of these rankings, emphasizing the superior performance of the proposed algorithms (QSMODE and DQSMODE).

In the SNE-SR ranking method, DQSMODE achieves the highest total score (99.9258%), followed by QSMODE (92.9368%) and Dijkstra (80.6348%), as shown in Table 4. The ranks of the eight algorithms are DQSMODE, QSMODE, Dijkstra, RRT*, RRT, A*, DWA, and APF. The DWA algorithm falls from fourth place in the Friedman test to seventh place in the SNE-SR ranking test because this test partially considers the best cost, and DWA fails in multiple models with a 0% success rate. The DQSMODE and QSMODE achieved top ranks in both tests, outperforming the other six algorithms.

9.3 Statistical analysis of the results: paired comparisons

The Wilcoxon test in Table 4 shows that DQSMODE consistently achieves a p-value below 0.05, significantly outperforming all other algorithms. Notably, there is no significant difference between DQSMODE and QSMODE (p-value > 0.05), confirming their similar performance. Figures 18 and 19 illustrate the detailed Wilcoxon test results. The DQSMODE significantly outperforms the APF algorithm in all 50 comparisons.

In all the open-field and single-obstacle scenarios, the DQSMODE performs significantly compared to all the other algorithms (except QSMODE). In the 13 multi-obstacle models, the DQSMODE performs better in 12 models compared with (A*, Dijkstra, and RRT*) algorithms, 13 models compared with RRT, and eight models compared with DWA. In the narrow-passages case, DQSMODE achieves better results in 14 models compared with the

Table 4 Statistical analysis for all 50 models for all the 8 path-planning algorithms

Alg.	Friedman test			SNE-SR ranking test				Wilcoxon Test				
	SumR	MeanR	Rank	Score1	Score2	Score	Rank	Sign Test	R+	R-	p-Val	H
A*	229	4.58	5	39.5143	25.2183	64.7327	6	+41/=0/-9	978	297	1.01E-03	TRUE
Dijkstra	174	3.48	3	47.4451	33.1897	80.6348	3	+39/=0/-11	903	372	1.04E-02	TRUE
RRT	287	5.74	7	48.8374	20.1220	68.9593	5	+44/=0/-6	1084	191	1.63E-05	TRUE
RRT*	277	5.54	6	48.9072	20.8484	69.7556	4	+42/=0/-8	1046	229	8.03E-05	TRUE
APF	385	7.7	8	6.3630	15.0000	21.3630	8	+50/=0/-0	1275	0	7.56E-10	TRUE
DWA	198	3.96	4	21.7324	29.1667	50.8991	7	+37/=0/-13	994	281	5.79E-04	TRUE
QSMODE	134.5	2.69	2	50.0000	42.9368	92.9368	2	+31/=1/-18	707	518	2.22E-01	FALSE
DQSMODE	115.5	2.31	1	49.9258	50.0000	99.9258	1	NA	NA	NA	NA	NA
	p-val	1.06203E-36										

In the Friedman test, the best algorithm is the one with the minimum mean rank. In SNE-SR ranking test, SNE is the sum of normalized error, SR is the sum rank, the score is out of 100, and the best algorithm is the one with the highest score. In Wilcoxon Test, DQSMODE is the reference algorithm in all paired comparisons. '+' means the number of models in which the DQSMODE is better, and '=' means draw. R+ > R- means the DQSMODE is better. The significance level α is between 0.05 and 0.1. The results are significant if p-value $< \alpha$

(RRT* and DWA), better in 15 compared with RRT, better in 13 models compared with A*, and better in 12 models compared with the Dijkstra algorithm. In the maze-like case, the DQSMODE is better than all the other algorithms in 12 to 13 cases out of the 13 cases compared with all the algorithms. For the overall performance, the QSMODE achieved a p-value less than 0.05 in all the paired comparisons, significantly outperforming all the other algorithms.

9.4 Path plots

This section shows the median path plot found by each algorithm in some path planning problems selected from each category: M1, M16, M24, and M48, as shown in Fig. 20. Figure 20a shows the path generated for the model M1 of the open-field scenario. Despite being a straightforward scenario, the QSMODE and DQSMODE are the only algorithms that successfully found a straight-line path with a minimum median cost of 20.0250. Figure 20b shows the problem M16 as a multi-obstacles scenario, where the QSMODE obtained the shortest smoothest paths followed by the DQSMODE with the minimum number of path waypoints and a median costs of 21.1119 and 21.2073. The APF failed in M16 due to the increased complexity of the model.

Figure 20c presents the path of M24 as a narrow-passage scenario, where APF and DWA failed to find a path. The A*, Dijkstra, RRT, and RRT* algorithms found a path by moving around the obstacles, following the outer path. In contrast, the DQSMODE succeeded in finding the shortest and smoothest path from the start to the end by moving through the obstacles with a median cost of 5.6842. The QSMODE achieved the second shortest path with a median cost of 6.0994.

Figures 20d shows the paths for a maze-like scenario (M48). The QSMODE, DQSMODE, and DWA algorithms achieve smooth paths compared to the A*, Dijkstra, RRT, and RRT*, but the DQSMODE and QSMODE achieve these paths with fewer path waypoints than the DWA algorithm. The QSMODE and DQSMODE have a success rate of 100% in all the models, while DWA failed in some models: M19, M21, M22, M24, M25, M30, M37, and M38. The results proved that the DQSMODE and QSMODE demonstrate the shortest, smoothest collision-free path compared with the A*, Dijkstra, RRT, RRT*, APF, and DWA algorithms. The path plots obtained by all the algorithms for all 50 models are presented in the supplementary materials (Fig. S9).

10 Practical validation using 4WD robot

This section deploys the proposed algorithm within a ROS-based ADS to validate the path-planning algorithm using a four-wheel differential drive (4WD) robotic car. The car is validated on two simulated scenarios, built using chicanes, falling under the narrow passages category in the RP2B-24 library but with different configurations to test the algorithm's adaptability to real-world environments. The main objective of this experiment is to validate the integration of the path-planning algorithm with the ADS, demonstrating its feasibility in real-time navigation tasks.

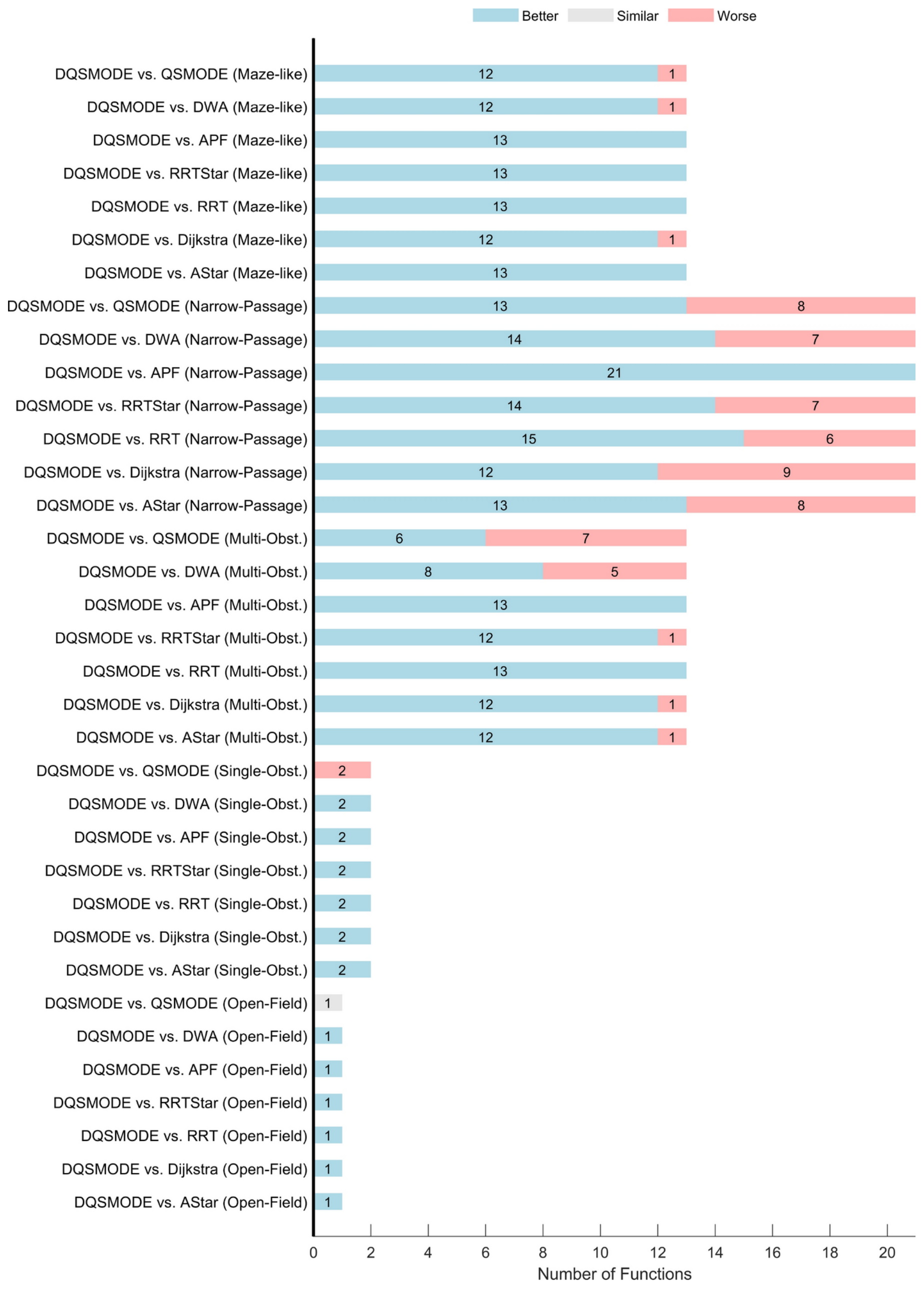


Fig. 18 Detailed Wilcoxon test results distributions for all the 50 models for the algorithms vs. the DQS-MODE algorithm for each category of the models (Model categories perspective)

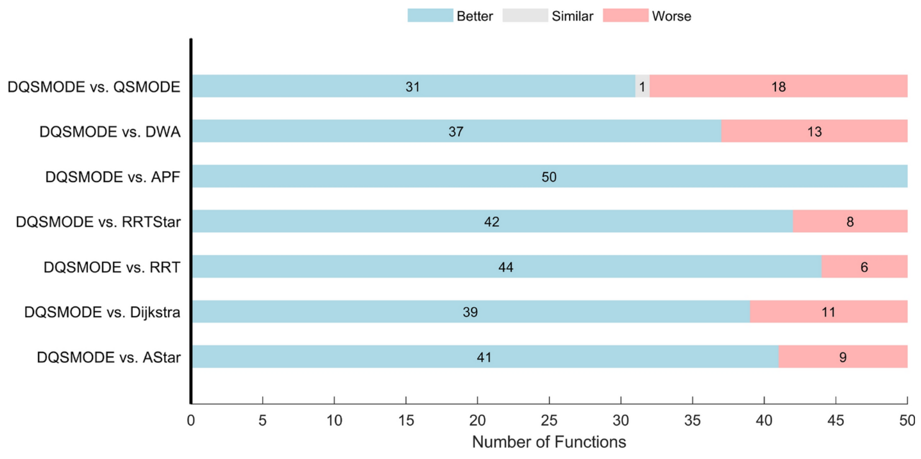


Fig. 19 Detailed Wilcoxon test results distributions for all the 50 models for the algorithms vs. the DQS-MODE algorithm (Algorithm perspective)

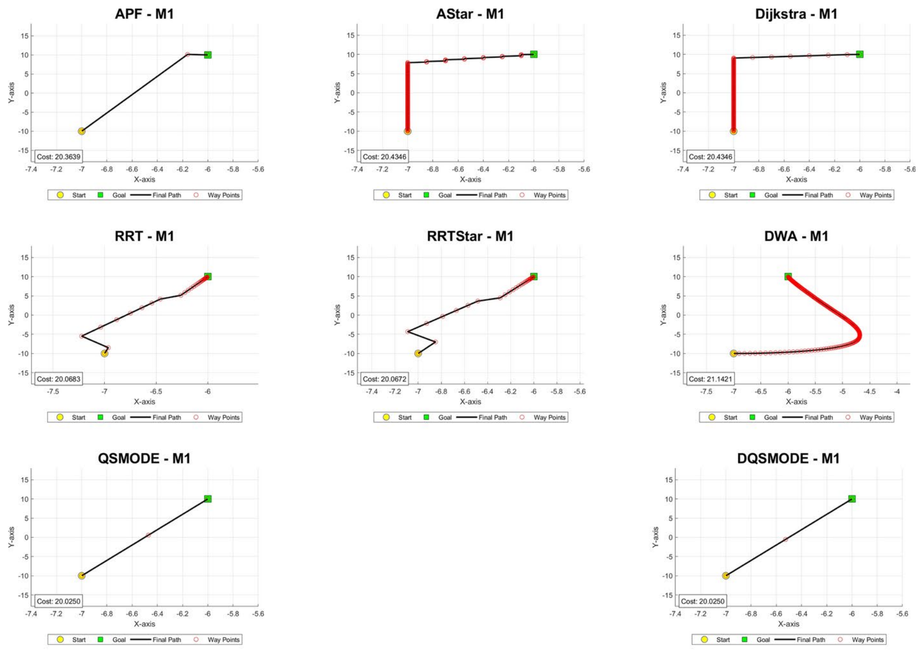
10.1 Hardware of the 4WD car

The system is validated on a 4WD with hardware indicated in Fig. 21. This car is based on the chassis of the Elegoo smart car V4 (ELEGOO Inc 2021). The car consists of 4 DC motors powered by a 7.4 lithium battery. Four motor drivers of type L298N have been used to generate the required signals for the DC motors (STMicroelectronics, L298N Dual Full-Bridge Driver (n.d.)). Four incremental encoders have been mounted in the car to measure the RPM speed of each motor (JOY-IT, KY-040 Rotary Encoder Datasheet (n.d.)). IMU sensor of type MPU-6050 is used to measure the car's orientation (InvenSense Inc. Mpu-6050 six-axis (gyro + accelerometer) mems motiontrackingdevices (n.d.)). A LiDAR of type HOKUYO UTM-30LX is used to gather distance data about the surroundings, which are used to generate the map (HOKUYO 2018). The LiDAR is powered with a dedicated 11.1 Lipo battery with a capacity of 2000 mAh (Allbattries 2023).

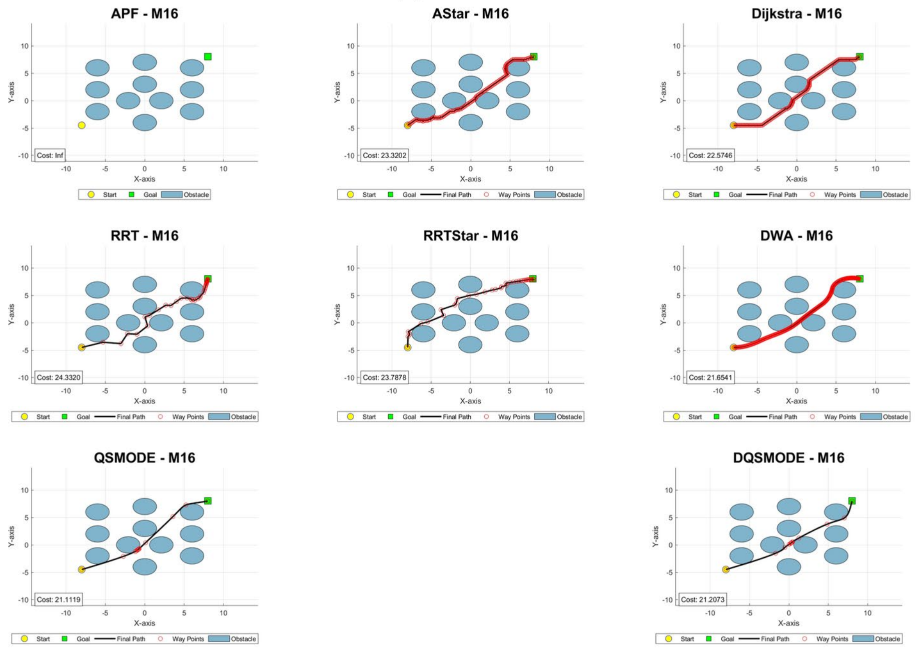
Two microcontrollers have been used in the car: Arduino Mega and Raspberry Pi 4B. The Arduino is responsible for low-level control such as motor direction control, speed control, and reading IMU and encoder data (Arduino, Arduino Mega 2560 Datasheet (n.d.)). The Raspberry Pi controller is responsible for the high-level control, and the ROS system is installed on it to implement the ADS stages in the form of ROS nodes (Raspberry Pi Foundation, Raspberry Pi 4 Model B specifications (n.d.)). The Raspberry Pi is powered by a UPS circuit (X728 V2.3), supplied by two 18650 batteries (GEEKWORM 2022). The Arduino communicates with the Raspberry Pi via UART protocol and USB cable.

10.2 ROS software nodes

This section explains the ROS software that implements the ADS system in the form of ROS nodes, and it is installed on the Raspberry Pi controller. Figure 22 shows the overall connections between all the ROS nodes. The Arduino node represents the Arduino firmware written in C++ and installed on the Arduino. It is responsible for reading the IMU and the encoder sensors data, and it also receives the desired velocities and applies PID control to

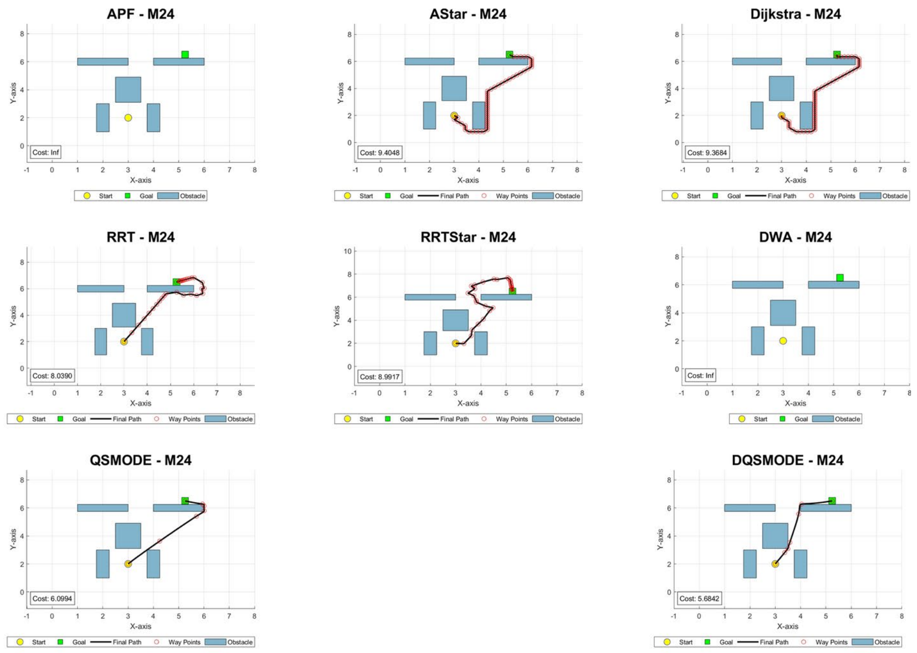


(a) M1 Model

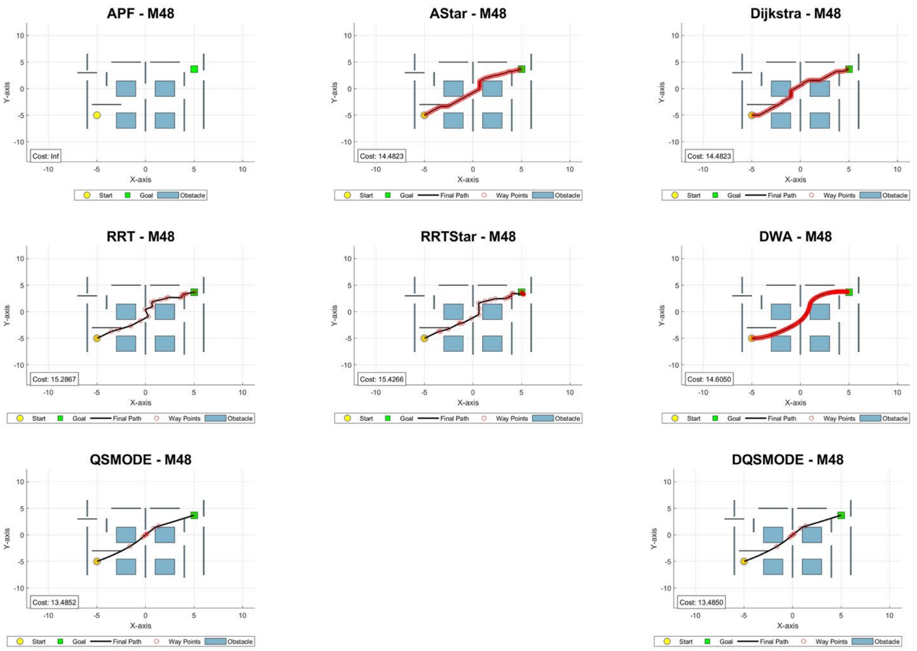


(b) M16 Model

Fig. 20 Median path plots for all the 8 path-planning algorithms. Median path plots for all the 8 path-planning algorithms



(c) M24 Model



(d) M48 Model

Figure 20 (continued)

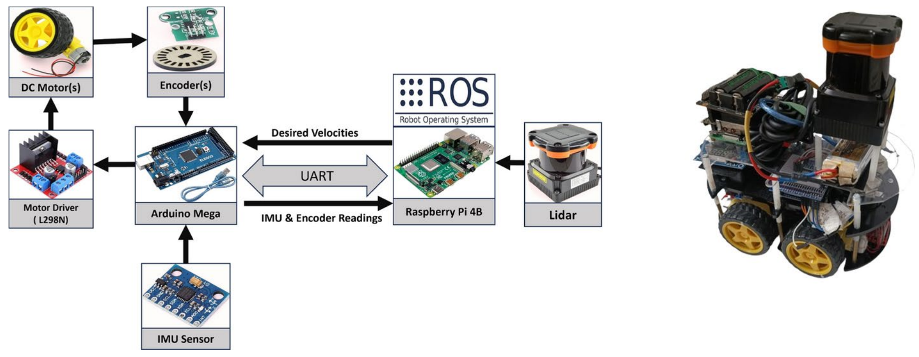
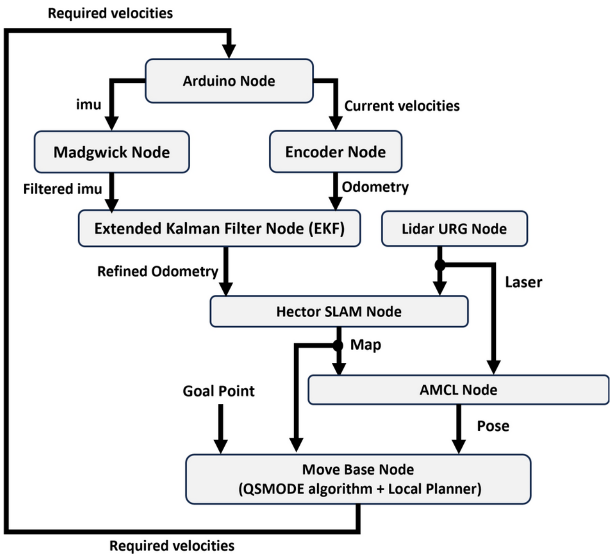


Fig. 21 The block diagram of the 4WD car's hardware

Fig. 22 The ROS block diagram of ADS



the motors to reach the desired velocities. Finally, it publishes the current encoders and the IMU readings. The encoder odometry node determines the car's location based on the dead reckoning of the encoder readings, publishing the raw odometry data. A Madgwick filter node merges the gyroscope and the accelerometer frames to generate a filtered IMU signal. This node is implemented using the imu_filter_madgwick node (Dryanovski and Günther 2022).

A LiDAR node reads the distances and data from the LiDAR sensors. The LiDAR node is implemented using the urg_node (Baltovski et al. 2024). A Kalman filter node is used to merge the raw odometry data from the encoder node and the filtered IMU data from the Madgwick filter node to generate the car's location for refined odometry. This node is implemented using robot_localization node (Moore 2024). Hector SLAM mapping is applied to the LiDAR data and the odometry data to generate the occupancy grid map, and it is implemented using the hector_slam node (Kohlbrecher et al. 2022). The AMCL node is used to estimate the car's location within the map using the map generated from the hector SLAM and the laser scan data from LiDAR, which produces the refined position of the car. This node is implemented using the AMCL node (ROS Wiki Contributors 2020a).

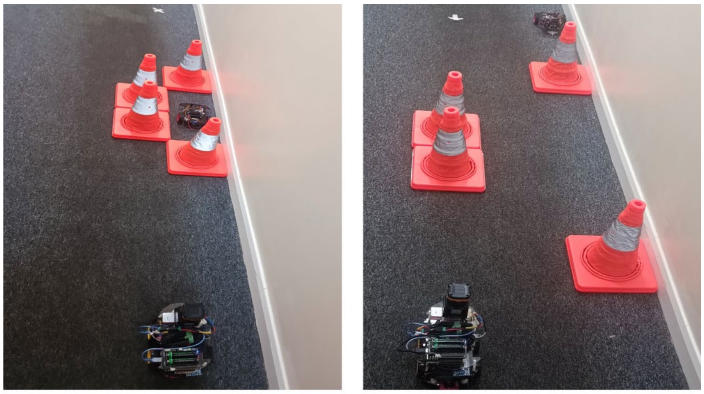
The move base node is responsible for path planning and generating the required velocities for the car to reach the desired goal. This node is implemented using the move_base node (ROS Wiki Contributors 2020b). The move_base node receives the position from the AMCL node and a desired goal point, and it runs a path planning algorithm (A* by default) to generate the path. Then, a local planner converts this path, and the following way is to point to the required velocities that will be sent to the Arduino node. In this node, the default path planning algorithm is replaced by the proposed QSMODE algorithm to validate its integration with the ADS system.

10.3 Validation on driving scenarios

The ADS system is validated using the 4WD car in two simulated scenarios built using chicanes as presented in Fig. 23. The main objective of this experiment is to ensure that the car can follow the route generated by the QSMODE algorithm to show the integration between the QSMODE and the other nodes.

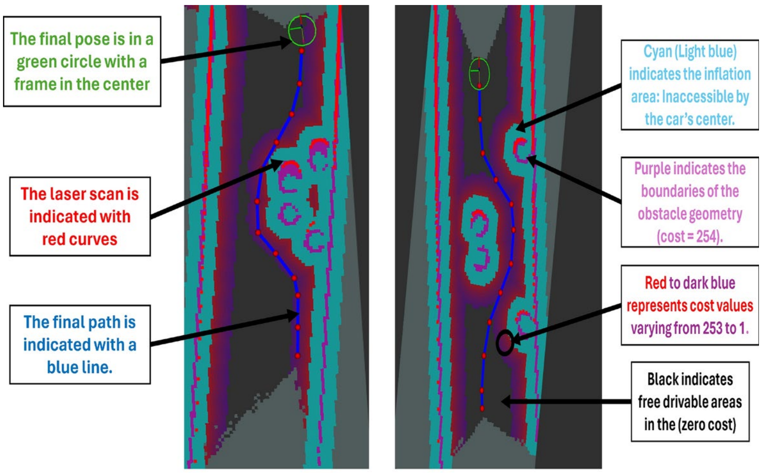
RViz is a ROS node used to visualize real-time data published by the ROS nodes, such as the map from the Hector SLAM and the pose of the car (ROS Wiki Contributors 2023a). This data is visualized on a PC connected to the vehicle's Raspberry Pi. The Raspberry Pi is connected to the PC via an SSH connection. The PC and Raspberry Pi must be connected to the same WiFi, and the ROS must be installed on both. The ROS Master is a central service that enables the communication between ROS nodes on different devices (ROS Wiki Contributors 2023b). On the Raspberry Pi, the nodes publish the topics such as the pose of the car, the map, and the waypoints. On the PC, the RViz node is running to visualize the data received from the Raspberry Pi nodes via the ROS Master.

The path followed by the 4WD is visualized on RViz for scenarios 1 and 2, as seen in Figs. 24a and 24b, respectively. The PC received the laser scan topic from the LiDAR node, the map topic from the Hector SLAM node, the pose from the AMCL node, and the waypoints from the move base node. The car's final path is represented by the blue line, which is divided into equally spaced waypoints denoted by red circles.



(a) Scenario 1. (b) Scenario 2.

Fig. 23 The simulated scenario for validation


The final pose is in a green circle with a frame in the center
The laser scan is indicated with red curves
The final path is indicated with a blue line.
Cyan (Light blue) indicates the inflation area: Inaccessible by the car's center.
Purple indicates the boundaries of the obstacle geometry (cost = 254).
Red to dark blue represents cost values varying from 253 to 1.
Black indicates free drivable areas in the (zero cost)

(a) Path of scenario 1. (b) Path of scenario 2.

Fig. 24 Final path and cost map of the proposed system on the driving scenarios

The real-time scan data from the LiDAR is displayed as red curves that surround the outer sides of the obstacles (chicanes) facing the LiDAR sensor. The car's current location is visualized as a frame attached to a green circle to indicate the position and orientation of the vehicle. The occupancy grid of the map is displayed with different colours based on the cost value for each grid cell. The highest cost is 253, representing a collision with probability 100%, while the lowest cost is zero, representing a free empty cell. The map's colours vary from red to black, corresponding to costs from 253 to 0.

The final path followed by the 4WD car in both scenarios proved the car's successful overtaking of the chicanes, following a collision-free path generated by the proposed algorithm. Therefore, these results showed the successful integration among all the ROS nodes, including the QSMODE algorithm, proving the feasibility of the proposed algorithm.

11 Conclusion

This study introduced two novel reinforcement learning RL-based algorithms, QSMODE and DQSMODE, to address path-planning challenges in autonomous driving systems (ADS). By integrating Q-learning and DQN with the cubic spline smoothing technique, QSMODE/DQSMODE overcomes the limitations of classical algorithms, such as sharp paths and high computational demands, while addressing the randomness issues in meta-heuristic optimization methods. RL in QSMODE/DQSMODE generates paths based on accumulated experience, improving path quality and reliability. The proposed algorithms feature a self-training update mechanism for the Q-table and DQN, an adaptive RL switching probability, and a secondary update method to enhance population diversity. This study also proposed a unified benchmarking framework, RP2B-24, providing 50 challenging comprehensive scenarios with diverse configurations, including open fields, obstacles, and maze-like paths, as a robust and generalizable testing platform.

In extensive validation, QSMODE and DQSMODE consistently outperformed 16 state-of-the-art meta-heuristic optimization algorithms, achieving top ranks in Friedman and SNE-SR tests with scores of 99.2877% (DQSMODE) and 93.0463 % (QSMODE). They also outperformed classical path-planning algorithms such as A*, Dijkstra, RRT, RRT*, and DWA, achieving superior SNE-ranking scores of 99.9258% (DQSMODE) and 92.9368% (QSMODE). Their RL framework enhanced search-space exploration, enabling faster and more reliable pathfinding with minimal waypoints. Moreover, the algorithm was practically validated in a ROS-based ADS system consisting of eight nodes using a four-wheel differential drive robot. The successful navigation through two driving scenarios demonstrated the algorithm's feasibility, with the car completing collision-free paths, showcasing the integration capabilities of QSMODE/DQSMODE within ADS frameworks. These results highlight the significant contributions of QSMODE and DQSMODE to the field of path planning, offering robust performance, generalizability, and practical applicability.

The proposed algorithms in this study achieved significant results compared to the state-of-the-art algorithms, leading to improved ADS performance. However, they also open the door to various opportunities for future research direction in ADS and artificial intelligence. In the future, the RL strategy in the DQSMODE algorithm can be improved further. One improvement is to perform long-term training for the DQN in the DQSMODE to see the effect of long-term training on the performance of the path planning of the DQSMODE algorithm. This point can pave the way for a general AI-based path-planning platform with the DQSMODE as its core. In addition, more mutation operators can be tested in the QSMODE/DQSMODE, and their effect can be studied compared to the original DQSMODE algorithm. Moreover, deploying alternative deep reinforcement learning models, such as actor-critic methods, could address the challenges of continuous action spaces, enhancing the flexibility and scalability of the algorithm. The sensitivity analysis can also be extended to further parameters in the QSMODE/DQSMODE algorithms. Finally, hybridizing QSMODE/DQSMODE with other algorithms may yield more advanced variants, broadening its applicability beyond ADS to other robotic and industrial optimization challenges.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s10462-025-11129-6.

Acknowledgements Not applicable.

Author contributions Conceptualization, M.R.; taxonomy, M.R.; methodology, M.R.; software, M.R.; hardware, M.R.; visualization, M.R.; formal analysis, M.R.; validation, M.R.; data curation, M.R.; investigation, M.R.; resources, A.O.; writing—original draft preparation, M.R; writing—review and editing, M.R.; supervision, A.O, A.G., and A.Y.H.; project administration, A.O, A.G., and A.Y.H.; funding acquisition, A.O. and A.G., All authors have read and agreed to the published version of the manuscript.

Funding Not applicable.

Data availability No datasets were used during the current study. The supplementary materials related to this paper will be available on the a GitHub repository after the manuscript has been approved.

Declarations

Conflict of interest There is no Conflict of interest between the authors to publish this manuscript.

Ethical approval Not applicable.

Consent to participate Not applicable.

Consent for publication Not applicable.

References

Ab Wahab MN, Nazir A, Khalil A, Ho WJ, Akbar MF, Noor MHM, Mohamed ASA (2024) Improved genetic algorithm for mobile robot path planning in static environments. Expert Syst Appl 249:123762

Achouri M, Zennir Y (2024) Path planning and tracking of wheeled mobile robot: using firefly algorithm and kinematic controller combined with sliding mode control. J Braz Soc Mech Sci Eng 46(4):1

Allbattries, Lithium Ion Battery Manual (2023). Available: https://www.allbatteries.co.uk/lithium-ion-battery-efest-imr18650-li-mn-hd-3-7v-3ah-ft-acl9045.html. Accessed: 2023-12-23

Arduino, Arduino Mega 2560 Datasheet (n.d.). Available: https://docs.arduino.cc/hardware/mega-2560/. Last accessed: 2023-12-14

Badrloo S, Varshosaz M, Pirasteh S, Li J (2022) Image-based obstacle detection methods for the safe navigation of unmanned vehicles: A review. Remote Sensing 14(15):3824

Baltovski T, Rockey C, O'Driscoll M (2024) urg_node: A ROS package for Hokuyo URG laser scanners (2024). https://wiki.ros.org/urg_node. Accessed: 2024-02-26

Batinovic A, Goricanec J, Markovic L, Bogdan S (2022) 2022 International conference on unmanned aircraft systems (ICUAS), IEEE, pp 394–401

Bhattacharyya S, Karmakar M (2022) Human-centric smart computing: proceedings of ICHCSC 2022. Springer, pp 283–291

Buehler M, Iagnemma K, Singh S (2009) The DARPA urban challenge: autonomous vehicles in city traffic, vol 56. Springer

Butt MZ, Nasir N, Rashid RBA (2024) A review of perception sensors, techniques, and hardware architectures for autonomous low-altitude UAVS in non-cooperative local obstacle avoidance. Robot Autonom Syst 173:104629

Carrasco J, García S, Rueda M, Das S, Herrera F (2020) Recent trends in the use of statistical tests for comparing swarm and evolutionary computing algorithms: practical guidelines and a critical review. Swarm Evol Comput 54:100665

Chang L, Shan L, Jiang C, Dai Y (2021) Reinforcement based mobile robot path planning with improved dynamic window approach in unknown environment. Auton Robot 45(1):51

Chen R, Hu J, Xu W (2022) An rrt-dijkstra-based path planning strategy for autonomous vehicles. Appl Sci 12(23):11982

Chen Y, Wang L, Liu G, Xia B (2022) Automatic parking path optimization based on immune moth flame algorithm for intelligent vehicles. Symmetry 14(9):1923

Cheng Y, Li C, Li S, Li Z (2020) Motion planning of redundant manipulator with variable joint velocity limit based on beetle antennae search algorithm. IEEE Access 8:138788

Chen J, Liang J, Tong Y (2020) 2020 16th International conference on control, automation, robotics and vision (ICARCV) IEEE, pp 811–816

Clifton J, Laber E (2020) Q-learning: theory and applications. Ann Rev Stat Appl 7:279

Cui Q, Liu P, Du H, Wang H, Ma X (2023) Improved multi-objective artificial bee colony algorithm-based path planning for mobile robots. Front Neurorobot 17:1196683

Cui J, Li Z (2022) Handbook of nature-inspired optimization algorithms: the state of the art: Volume I: solving single objective bound-constrained real-parameter numerical optimization problems (Springer), pp 117–134

Dai X, Wei Y (2021) Application of improved moth-flame optimization algorithm for robot path planning. IEEE Access 9:105914

Das P, Behera H, Panigrahi B (2016) Intelligent-based multi-robot path planning inspired by improved classical q-learning and improved particle swarm optimization with perturbed velocity. Eng Sci Technol Int J 19(1):651

Derrac J, García S, Molina D, Herrera F (2011) A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. Swarm Evol Comput 1(1):3

Divine G, Norton HJ, Hunt R, Dienemann J (2013) A review of analysis and sample size calculation considerations for Wilcoxon tests. Anesthesia Analgesia 117(3):699

Dobrevski M, Skočaj D (2024) Dynamic adaptive dynamic window approach. IEEE Trans Robot 40:3068

Dryanovski Ivan, Günther Martin (2022) imu_filter_madgwick - ROS Wiki. https://wiki.ros.org/imu_filter_madgwick. [Online; accessed 20-Dec-2023]

ELEGOO Inc., Smart Robot Car V4.0 With Camera Assembly Tutorial (2021). https://www.elegoo.com/en-gb/blogs/arduino-projects/elegoo-smart-robot-car-kit-v4-0-tutorial. Available from ELEGOO Inc. Website

Eshtehardian S, Khodaygan S (2023) A continuous rrt*-based path planning method for non-holonomic mobile robots using b-spline curves. J Ambient Intell Humaniz Comput 14(7):8693

Fan J, Wang Z, Xie Y, Yang Z (2020) Learning for dynamics and control PMLR, pp 486–489

Fischetti M, Stringher M (2019) Embedded hyper-parameter tuning by simulated annealing, arXiv preprint arXiv:1906.01504

Fu W, Wang B, Li X, Liu L, Wang Y (2019) Ascent trajectory optimization for hypersonic vehicle based on improved chicken swarm optimization. IEEE Access 7:151836

GEEKWORM, X728 V2.3 Raspberry Pi UPS Manual (2022). https://wiki.geekworm.com/X728#X728_V2.3. Accessed: 2023-12-23

Goyal P, Malik H, Sharma R (2019) Application of evolutionary reinforcement learning (erl) approach in control domain: a review. Smart Innov Commun Comput Sci: Proc 2:273–288

Gul F, Mir I, Abualigah L, Sumari P, Forestiero A (2021) A consolidated review of path planning and optimization techniques: technical perspectives and future directions. Electronics 10(18):2250

Halat S, Ebadzadeh MM (2021) Modified double dqn: addressing stability, arXiv preprint arXiv:2108.04115

Hansen N, Ostermeier A (2001) Completely derandomized self-adaptation in evolution strategies. Evol Comput 9(2):159

Heaton J (2008) Introduction to neural networks for java. Heaton Research

Heidari AA, Mirjalili S, Faris H, Aljarah I, Mafarja M, Chen H (2019) Harris hawks optimization: algorithm and applications. Future Gener Comput Syst 97:849

Heiden E, Palmieri L, Bruns L, Arras KO, Sukhatme GS, Koenig S (2021) Bench-mr: a motion planning benchmark for wheeled mobile robots. IEEE Robot Autom Lett 6(3):4536

Heris MK (2015) Simulated annealing in matlab. https://yarpiz.com/223/ypea105-simulated-annealing

HOKUYO, Hokuyo UTM-30LX Lidar Manual (2018). https://www.hokuyo-aut.jp/search/single.php?serial=169. Accessed: 2023-12-23

Houssein EH, Younan M, Hassanien AE (2019) Nature-inspired algorithms: a comprehensive review. Hybrid Comput Intell. https://doi.org/10.1201/9780429453427-1

hua Zhang J, Feng Q, di Zhao A, He W, Hao X (2021) Journal of Physics: Conference Series, vol. 1905-1 IOP Publishing, vol. 1905-1, p 012019

Huang L, Fu Q, Tong N (2023) Solving robot path planning problem by adaptively adjusted Harris hawk optimization algorithm. J Comput Appl 43(12):3840

Huang L, Fu Q, Tong N (2023) An improved Harris hawks optimization algorithm and its application in grid map path planning. Biomimetics 8(5):428

Hu J, Hu Y, Liu K, Wang W, Chen H (2019) 2019 IEEE intelligent transportation systems conference (ITSC) (IEEE), pp 4140–4145

InvenSense Inc. Mpu-6050 six-axis (gyro + accelerometer) mems motiontracking devices (n.d.). https://invensense.tdk.com/products/motion-tracking/6-axis/mpu-6050/. Accessed: 2023-12-14

Iswanto I, Ma'arif A, Wahyunggoro O, Cahyadi AI (2019) Artificial potential field algorithm implementation for quadrotor path planning. Int J Adv Comput Sci Appl 10(8):1

Jang B, Kim M, Harerimana G, Kim JW (2019) Q-learning algorithms: a comprehensive classification and applications. IEEE Access 7:133653

Jiang X, Li S (2017) Bas: beetle antennae search algorithm for optimization problems, arXiv preprint arXiv:1710.10724

JOY-IT, KY-040 Rotary Encoder Datasheet (n.d.). https://www.alldatasheet.com/datasheet-pdf/pdf/1648739/JOY-IT/KY-040.html. Accessed: 2023-12-14

Juang CF, Bui TB (2019) Reinforcement neural fuzzy surrogate-assisted multiobjective evolutionary fuzzy systems with robot learning control application. IEEE Trans Fuzzy Syst 28(3):434

Kästner L, Bhuiyan T, Le TA, Treis E, Cox J, Meinardus B, Kmiecik J, Carstens R, Pichel D, Fatloun B et al (2022) Arena-bench: a benchmarking suite for obstacle avoidance approaches in highly dynamic environments. IEEE Robot Autom Lett 7(4):9477

Kobayashi M, Motoi N (2022) Local path planning: dynamic window approach with virtual manipulators considering dynamic obstacles. IEEE Access 10:17018

Kohlbrecher S, Meyer J et al (2022) hector_slam - ros wiki. http://wiki.ros.org/hector_slam. Accessed: 2024-03-02

Kong J, Cheng J (2023) 2023 IEEE 6th information technology, networking, electronic and automation control conference (ITNEC), (IEEE), vol. 6, pp 968–973

Koohi SZ, Hamid NAWA, Othman M, Ibragimov G (2018) Raccoon optimization algorithm. IEEE Access 7:5383

Kumar R, Singh L, Tiwari R (2021) Path planning for the autonomous robots using modified grey wolf optimization approach. J Intell Fuzzy Syst 40(5):9453

Kumar A, Price KV, Mohamed AW, Hadi AA, Suganthan PN (2021) Problem definitions and evaluation criteria for the CEC 2022 special session and competition on single objective bound constrained numerical optimization. Technical Report, Nanyang Technological University. https://github.com/P-N-Suganthan/2022-SO-BO

Kunchev V, Jain L, Ivancevic V, Finn A (2006) In: Knowledge-based intelligent information and engineering systems: 10th international conference, KES 2006, Bournemouth, UK, October 9–11, 2006. Proceedings, Part II 10 Springer, pp 537–544

Li X, Wu D, He J, Bashir M, Liping M (2020) An improved method of particle swarm optimization for path planning of mobile robot. J Control Sci Eng 2020:1

Li W, Tan M, Wang L, Wang Q (2020) A cubic spline method combing improved particle swarm optimization for robot path planning in dynamic uncertain environment. Int J Adv Rob Syst 17(1):1729881419891661

Li D, Wang L, Cai J, Wang A, Tan T, Gui J (2023) Research on path planning of mobile robot based on improved genetic algorithm. Int J Model, Simul, Sci Comput 14(06):2341030

Lian J, Yu W, Xiao K, Liu W (2020) Cubic spline interpolation-based robot path planning using a chaotic adaptive particle swarm optimization algorithm. Math Probl Eng 2020:1

Li B, Dong C, Chen Q, Mu Y, Fan Z, Wang Q, Chen X (2020) In: Proceedings of the 2020 4th high performance computing and cluster technologies conference & 2020 3rd international conference on big data and artificial intelligence, pp 49–53

Liu LS, Lin JF, Yao JX, He DW, Zheng JS, Huang J, Shi P (2021) Path planning for smart car based on dijkstra algorithm and dynamic window approach. Wirel Commun Mob Comput 2021:1

Liu R, Zou J (2018) 2018 56th Annual Allerton conference on communication, control, and computing (Allerton) IEEE, pp 478–485

Lv L, Zhang S, Ding D, Wang Y (2019) Path planning via an improved dqn-based learning policy. IEEE Access 7:67319

Lyu Y, Mo Y, Yue S, Hong L (2023) A mobile robot path planning using improved beetle swarm optimization algorithm in static environment. J Intell Fuzzy Syst 45:11453

Majumder A, Majumder A, Bhaumik R (2021) Teaching-learning-based optimization algorithm for path planning and task allocation in multi-robot plant inspection system. Arab J Sci Eng 46(9):8999

Ma J, Liu Y, Zang S, Wang L (2020) Robot path planning based on genetic algorithm fused with continuous Bezier optimization, Computational intelligence and neuroscience 2020

Mao S, Yang P, Gao D, Bao C, Wang Z (2023) A motion planning method for unmanned surface vehicle based on improved rrt algorithm. J Marine Sci Eng 11(4):687

Meng X, Liu Y, Gao X, Zhang H (2014) International conference in swarm intelligence. Springer, pp 86–94

Mirjalili S (2015) Moth-flame optimization algorithm: a novel nature-inspired heuristic paradigm. Knowl-Based Syst 89:228

Mirjalili S, Lewis A (2016) The whale optimization algorithm. Adv Eng Softw 95:51

Mirjalili S, Mirjalili SM, Lewis A (2014) Grey wolf optimizer. Adv Eng Softw 69:46

Molina E, Viale L, Vázquez P (2022) 2022 IEEE 4th workshop on visualization guidelines in research, design, and education (VisGuides) IEEE, pp 1–7

Moore T (2024) robot_localization. https://wiki.ros.org/robot_localization. Accessed: 2024-03-20

Niu C, Li A, Huang X, Xu C (2021) 2021 Global reliability and prognostics and health management (PHM-Nanjing), IEEE, pp 1–7

Nossier E, Ibrahim F, Abdelwahab M, Abdelaziz M (2021) 2021 16th International conference on computer engineering and systems (ICCES). https://doi.org/10.1109/ICCES54031.2021.9686134

Orthey A, Chamzas C, Kavraki LE (2023) Sampling-based motion planning: a comparative review. Ann Rev Control, Robot, Autonom Syst 7:285

Parekh D, Poddar N, Rajpurkar A, Chahal M, Kumar N, Joshi GP, Cho W (2022) A review on autonomous vehicles: progress, methods and challenges. Electronics 11(14):2162

Patle B, Pandey A, Jagadeesh A, Parhi DR (2018) Path planning in uncertain environment by using firefly algorithm. Defence Technol 14(6):691

Qiuyun T, Hongyan S, Hengwei G, Ping W (2021) Improved particle swarm optimization algorithm for agv path planning. IEEE Access 9:33522

Rao RV, Savsani VJ, Vakharia D (2011) Teaching-learning-based optimization: a novel method for constrained mechanical design optimization problems. Comput Aided Des 43(3):303

Raspberry Pi Foundation, Raspberry Pi 4 Model B specifications (n.d.). Available: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/. Last accessed: 2023-12-14

Reda M, Onsy A, Haikal AY, Ghanbari A (2024) Path planning algorithms in the autonomous driving system: a comprehensive review. Robot Autonom Syst 174:104630

Reda M, Onsy A, Haikal AY, Ghanbari A (2024) Optimizing the steering of driverless personal mobility pods with a novel differential Harris hawks optimization algorithm (dhho) and encoder modeling. Sensors (Basel, Switzerland) 24(14):4650

ROS Wiki Contributors. amcl - ros wiki (2020). https://wiki.ros.org/amcl. Accessed: 2024-07-29

ROS Wiki Contributors. move_base - ros wiki (2020). https://wiki.ros.org/move_base. Accessed: 2024-07-29

ROS Wiki Contributors. Ros master - ros wiki. http://wiki.ros.org/Master (2023b). [Last Accessed; 14-01-2024]

ROS Wiki Contributors. Ros noetic ninjemys (2023). https://wiki.ros.org/noetic. Accessed: 2024-02-17

ROS Wiki Contributors. rviz - ros wiki (2023a). http://wiki.ros.org/rviz. Accessed: 15-01-2024

Sabiha AD, Kamel MA, Said E, Hussein WM (2022) Real-time path planning for autonomous vehicle based on teaching-learning-based optimization. Intel Serv Robot 15(3):381

Sallam KM, Abdel-Basset M, El-Abd M, Wagdy A (2022) 2022 IEEE congress on evolutionary computation (CEC) IEEE, pp 1–8

Sallam KM, Elsayed SM, Chakrabortty RK, Ryan MJ (2020) In: 2020 IEEE congress on evolutionary computation (CEC) IEEE, pp 1–8

Sánchez-Ibáñez JR, Pérez-del Pulgar CJ, García-Cerezo A (2021) Path planning for autonomous mobile robots: a review. Sensors 21(23):7898

Shi W, He Z, Tang W, Liu W, Ma Z (2022) Path planning of multi-robot systems with Boolean specifications based on simulated annealing. IEEE Robot Autom Lett 7(3):6091

Shi P, Liu Z, Liu G (2022) Proceedings of the 2022 4th asia pacific information technology conference, pp 231–239

Si Q, Li C (2023) Indoor robot path planning using an improved whale optimization algorithm. Sensors 23(8):3988

Simon D (2008) Biogeography-based optimization. IEEE Trans Evol Comput 12(6):702

Slowik A, Kwasnicka H (2020) Evolutionary algorithms and their applications to engineering problems. Neural Comput Appl 32:12363

Sood M, Panchal VK (2020) Meta-heuristic techniques for path planning: recent trends and advancements. Int J Intell Syst Technol Appl 19(1):36

STMicroelectronics, L298N Dual Full-Bridge Driver (n.d.). https://www.alldatasheet.com/datasheet-pdf/pdf/22440/STMICROELECTRONICS/L298N.html. Accessed: 2023-12-14

Sun Y, Fang M, Su Y (2021) Journal of Physics: Conference Series, vol 1746–1. IOP Publishing, p 012052

Sung N, Kim S, Cho N (2023) An efficient path planning algorithm using a potential field for ground forces. Computation 11(1):12

Tan Z, Li K (2021) Differential evolution with mixed mutation strategy based on deep reinforcement learning. Appl Soft Comput 111:107678

Tang J, Liu G, Pan Q (2021) A review on representative swarm intelligence algorithms for solving optimization problems: applications and trends. IEEE/CAA J Autom Sin 8(10):1627

Thomas A, Majumdar P, Eldho T, Rastogi A (2018) Simulation optimization model for aquifer parameter estimation using coupled meshfree point collocation method and cat swarm optimization. Eng Anal Boundary Elem 91:60

Thoresen M, Nielsen NH, Mathiassen K, Pettersen KY (2021) Path planning for ugvs based on traversability hybrid a. IEEE Robot Autom Lett 6(2):1216

Toufan N, Niknafs A (2020) Robot path planning based on laser range finder and novel objective functions in grey wolf optimizer. SN Appl Sci 2(8):1324

Velasco-Hernandez G, Barry J, Walsh J, et al (2020) 2020 IEEE 16th international conference on intelligent computer communication and processing (ICCP) IEEE, pp 315–321

Vieillard N, Pietquin O, Geist M (2020) Munchausen reinforcement learning. Adv Neural Inf Process Syst 33:4235

Wang C, Cheng C, Yang D, Pan G, Zhang F (2022) Path planning in localization uncertaining environment based on dijkstra method. Front Neurorobot 16:821991

Wang CH, Chen S, Zhao Q, Suo Y (2023) An efficient end-to-end obstacle avoidance path planning algorithm for intelligent vehicles based on improved whale optimization algorithm. Mathematics 11(8):1800

Wong K, Gu Y, Kamijo S (2020) Mapping for autonomous driving: opportunities and challenges. IEEE Intell Transp Syst Mag 13(1):91

Woolson RF (2007) Wilcoxon signed-rank test. Wiley encyclopedia of clinical trials, pp 1–3

Wu Z, Dai J, Jiang B, Karimi HR (2023) Robot path planning based on artificial potential field with deterministic annealing. ISA Trans 138:74

Xie S, Hu J, Bhowmick P, Ding Z, Arvin F (2022) Distributed motion planning for safe autonomous vehicle overtaking via artificial potential field. IEEE Trans Intell Transp Syst 23(11):21531

Xu F, Li H, Pun CM, Hu H, Li Y, Song Y, Gao H (2020) A new global best guided artificial bee colony algorithm with application in robot path planning. Appl Soft Comput 88:106037

Yang H, Vigneron A (2022) Coordinated path planning through local search and simulated annealing. ACM J Exp Algorithm (JEA) 27:1

Yao Q, Zheng Z, Qi L, Yuan H, Guo X, Zhao M, Liu Z, Yang T (2020) Path planning method with improved artificial potential field-a reinforcement learning perspective. IEEE Access 8:135513

Yu X, Li C, Zhou J (2020) A constrained differential evolution algorithm to solve uav path planning in disaster scenarios. Knowl-Based Syst 204:106209

Yue CT, Price KV, Suganthan PN, Liang JJ, Ali MZ, Qu BY, Awad NH, Biswas PP (2019) Problem definitions and evaluation criteria for the cec 2020 special session and competition on single objective bound constrained numerical optimization. Technical Report, Nanyang Technological University, Singapore. https://github.com/P-N-Suganthan/2020-Bound-Constrained-Opt-Benchmark

Yurtsever E, Lambert J, Carballo A, Takeda K (2020) A survey of autonomous driving: common practices and emerging technologies. IEEE Access 8:58443

Zan J (2022) Research on robot path perception and optimization technology based on whale optimization algorithm. J Comput Cognit Eng 1(4):201

Zhang J, Liu M, Zhang S, Zheng R (2022) Auv path planning based on differential evolution with environment prediction. J Intell Robot Syst 104(2):23

Zhang D, Chen C, Zhang G (2024) 2024 IEEE 7th advanced information technology, electronic and automation control conference (IAEAC), IEEE, vol. 7, pp 1590–1595

Zhang F, Li N, Xue T, Zhu Y, Yuan R, Fu Y (2019) 2019 IEEE international conference on robotics and biomimetics (ROBIO) IEEE, pp 2873–2878

Zhang Y, Ou B, Xu Y, Dai C (2023) 2023 8th international conference on cloud computing and big data analytics (ICCCBDA) IEEE, pp 501–505

Zhao Z (2022) 2nd International conference on artificial intelligence, automation, and high-performance computing (AIAHPC 2022) (SPIE), vol. 12348, pp 470–481

Zhao D, Yu H, Fang X, Tian L, Han P (2020) A path planning method based on multi-objective Cauchy mutation cat swarm optimization algorithm for navigation system of intelligent patrol car. IEEE Access 8:151788

Zhao J, Zhang J, Liu H, Wang J, Chen Z (2023) Path planning for a tracked robot traversing uneven terrains based on tip-over stability. Asian J Control 25(5):3569

Zhao J, Zhang J, Wang J, Zhang X, Wang Y (2021) 2021 33rd Chinese control and decision conference (CCDC) IEEE, pp 1085–1091

Authors and Affiliations

Mohamed Reda[1,2] · Ahmed Onsy[1] · Amira Y. Haikal[2] · Ali Ghanbari[1]

✉ Mohamed Reda
mohamed.reda.mu@gmail.com; mramohamed@uclan.ac.uk; mohamed.reda@mans.edu.eg

[1] School of Engineering, University of Central Lancashire, Preston PR1 2HE, UK

[2] Computers and Control Systems Engineering Department, Faculty of Engineering, Mansoura University, Mansoura 35516, Egypt