Advantages and disadvantages of various robot algorithms and their comparison

Hongxiang Lai *

Department of International Education College, Jiangsu University of Technology, Changzhou, China

* Corresponding Author Email: 2023701124@smail.jstu.edu.cn

Abstract. In recent years, robotics technology has developed rapidly around the world and has been widely used in many fields, greatly promoting the automation and intelligence of related industries. In this context, robotic algorithms have played a key role in path planning, industrial production of robotic arms, and warehouse management. This study systematically reviews the five key indicators of various robotic algorithms (such as Dijkstra algorithm, A* algorithm, RRT algorithm, etc.) from the time dimension - environmental adaptability, learning efficiency, safety, real-time and robustness, and focuses on analysing the performance of these algorithms in path planning applications in complex obstacle environments. By combining and analysing relevant domestic and foreign literature, the basic principles and performance indicators of various robotic algorithms are systematically summarized, and the advantages and limitations of each algorithm are deeply discussed in combination with actual application scenarios. Finally, the future research directions and development trends of these algorithms are expected to provide a useful reference for related research.

Keywords: Algorithms, Path planning, Obstacle avoidance.

1. Introduction

With the deepening of Industry 4.0 and the intelligent transformation of emerging fields such as medical care and logistics, the environment faced by robots is changing from static scenes to dynamic scenes. In smart manufacturing workshops, mobile robots need to respond to environmental changes in real time, such as forklifts and workers; in minimally invasive surgery scenarios, medical robotic arms need to operate safely in human tissue; in warehousing and logistics systems, automatic guided vehicles need to deal with dynamic changes such as package piles and original routes that are impassable. Such scenarios pose a dual challenge to the core capabilities of algorithms: on the one hand, the algorithm needs to achieve faster responses in dynamic environments to cope with real-time changes in dynamic environments; on the other hand, it is necessary to achieve a balance between path optimality, computational efficiency, and system stability. In high-precision application scenarios, the safety fault tolerance rate needs to be controlled below 0.1%. Therefore, some traditional algorithms gradually reveal their limitations under such complex requirements. For example, although the A* algorithm method can guarantee optimality, it relies on static environment modelling and is difficult to handle dynamic obstacles. Although Hybrid A* improves path feasibility by introducing vehicle dynamics constraints, its discretized state space characteristics cause the computational complexity to grow exponentially with the increase of dimensions. Although the RRT series of algorithms is adapted to high-dimensional spaces, their random sampling mechanism is prone to planning failures in narrow channels or dynamic scenarios. More importantly, the existing algorithms generally lack the closed-loop architecture of "environmental perception-decision-control", making it difficult to achieve autonomous online learning in a dynamic environment, resulting in the inability of the response to emergencies to meet engineering requirements. Due to sensor noise, environmental unpredictability, and real-time processing limitations, perception is still a major research bottleneck. This study constructs a five-dimensional evaluation system of environmental adaptability, learning efficiency (sample complexity), safety (collision probability), real-time (decision delay) and robustness (interference tolerance), quantitatively compares the

performance changes of the algorithms at each stage, reveals the evolution law of the algorithms, and provides solutions for the real-time-safety-interpretability of the next generation of robot systems. The research results have theoretical value and engineering guidance significance in fields such as robot-intelligent manufacturing.

2. Algorithm Analysis

2.1. Dijkstra

The Dijkstra algorithm was proposed by Dutch computer scientist Edsger W. Dijkstra in 1956. Its core purpose is to find the shortest path from a single source node to all other nodes in a graph with non-negative weights [1]. It gradually determines the shortest path through a greedy strategy to ensure that the node with the shortest known distance is selected at each step. The greedy strategy means that each time the node closest to the starting point is selected, the shortest distance to its neighbouring nodes is updated. The relaxation operation is to update the shortest distance of the node by comparing the length of the known path with the length of the new path. For an edge(u,v), if the currently known short distance from the starting point to v is greater than the shortest distance from the starting point to u plus the weight of the edge (u,v), then the shortest distance of v is updated [1]. That is for all neighbouring nodes v of node u, check whether it is possible to reach v through u to make the path shorter, that is, judge whether

$$\mathrm{dist}(\mathrm{v}) > \mathrm{dist}(\mathrm{u}) + \mathrm{w}(\mathrm{u},\mathrm{v}) \tag{1}$$

$\mathrm{w}(\mathrm{u},\mathrm{v})$ is the weight of the edge u to v. If so, update the value of $\mathrm{dist}(\mathrm{v})$[2]. The Dijkstra algorithm performs stably in static, low-dimensional, known structure environments and is suitable for simple robot path navigation. It has poor real-time update capabilities for map changes, so it is difficult to handle dynamic obstacles or high-dimensional continuous spaces (such as robotic arm joint space). From the perspective of learning efficiency, Dijkstra does not require training data or environmental models. Dijkstra is directly based on graph structure calculations. Because the algorithm logic is simple, it can be quickly solved in small-scale scenarios, with a time complexity of:

$$O(|E| + |V| \log V) \tag{2}$$

Nodes V or the number of edges E is too large, so most nodes need to be explored. So, the decision time increases significantly, and the calculation efficiency drops sharply, making it difficult to meet real-time requirements and not suitable for large-scale scenarios [2]. In practical applications, the Dijkstra algorithm guarantees the shortest path. If the environment is completely static, the obstacle information is accurate, and the collision probability approaches zero, it cannot cope with real-time obstacle changes or external interference due to the lack of a dynamic obstacle avoidance mechanism. In terms of robustness, the algorithm is sensitive to small mapping errors and has extremely low tolerance for dynamic interference. The full map path needs to be recalculated after the environment changes. Further improvements, such as combining heuristic search to narrow the search scope or using hierarchical structure technology, can improve its real-time performance and robustness. In GPS navigation systems, the Dijkstra algorithm efficiently solves the shortest path from the starting point to the end point by converting the urban road network into a weighted graph and using the priority queue to perform greedy relaxation operations. It has good efficiency and stability in most application scenarios and provides drivers with the best driving route in real-time navigation.

2.2. A* algorithm

The A* algorithm was proposed by Peter Hart, Nils Nilsson, and Bertram Raphael in 1968. It was born to solve the problem of more efficient path search. By introducing a heuristic function to search the space, the path planning was greatly accelerated. Based on the traditional Dijkstra algorithm, the

A* algorithm uses the heuristic function $f(n) = g(n) + h(n)$ to balance the actual cost from the starting point to the current node and the exponential cost from the current node to the target, making the search more efficient. g(n) is the actual cost from the starting point to the node n, h(n) is the heuristic estimated cost from the node n to the target node, such as the Manhattan distance.

$$h(n) = |x_n - x_{end}| + |y_n - y_{end}| \tag{3}$$

$$h(n) = \sqrt{(x_n - x_{end})^2 + (y_n - y_{end})^2} \tag{4}$$

On a square grid that allows 4 directions of movement, the Manhattan distance is used; on a square grid that allows 8 directions of movement, the diagonal distance is used; On a hexagonal grid that allows arbitrary angle movement, a Manhattan distance adapted to the hexagonal grid is used [3]. And f(n) is the total estimated cost from the starting point through node n to the target node [4]. The A* algorithm is suitable for regular, structured environments (such as grid maps) and significantly improves the search efficiency through heuristic functions. A* does not rely on sample training and directly uses the problem structure for reasoning, so it performs well in sample utilization. However, it is difficult to design effective heuristic functions in high-dimensional or continuous spaces, and their scalability is limited. In path planning, the number of search nodes is reduced by using heuristic functions, and the sample complexity is significantly lower than that of Dijkstra. In terms of safety, if the heuristic function meets the acceptability, the optimality of the path can be guaranteed, and the risk of collision is low. However, since it does not consider the dynamic constraints of the vehicle itself, it needs to be replanned according to the external environment and cannot be effectively applied to engineering. In terms of real-time performance, A* can usually converge quickly under the guidance of the heuristic function, but there is still a problem of computational delay in large-scale or high-complexity environments, especially in the context of real-time update of map information, the delay problem will be more obvious. The robustness of the A* algorithm depends on the robustness of the heuristic function design. When encountering unexpected interruptions or sudden changes in the environment, the heuristic information may cause the search to enter the wrong path, and the algorithm's tolerance for abnormal information is reduced. In general, the advantages of the A* algorithm in improving search have achieved remarkable results in many path planning problems, but in dynamic environments or tasks with higher real-time requirements, improved technology is still needed to improve overall performance. In intelligent digital production workshops, AGV is widely used for material handling. Through the improvement of the A algorithm and the integration with other algorithms, AGV can achieve efficient and smooth path planning in complex indoor environments to meet the needs of real-time navigation and obstacle avoidance.

2.3. PRM (Probabilistic Roadmap)

The PRM (Probabilistic Roadmap) algorithm was proposed by Lydia in 1996. The purpose of this algorithm is to efficiently generate collision-free paths in high-dimensional continuous space. It is particularly suitable for path obstacle avoidance problems in complex environments such as robot path planning and autonomous driving. The core idea is to discretize the continuous space into a roadmap through random sampling, and then quickly find a feasible path in combination with a graph search algorithm (such as A*, Dijkstra). Since the traditional grid method has an extremely high computational cost in complex or high-dimensional spaces (such as robotic arms), PRM significantly reduces the complexity through sparse sampling. The PRM algorithm is very suitable for complex, high-dimensional continuous environments, can handle unstructured, random map scenes, and quickly verify the connectivity of the path [5]. However, since the search path depends on the sampling points and the number of sampling points is limited, it cannot completely cover the entire free space, and the distribution is random, the final path is often not the optimal path, but a suboptimal path [6]. In terms of learning efficiency, the PRM algorithm requires multiple random samplings to build a comprehensive roadmap, so the number of samples is closely related to the spatial complexity. By adjusting the sampling strategy and parameters, it can adapt to different environmental characteristics (such as narrow channels, obstacle edges, etc.). In terms of safety, the path safety of

PRM depends on the accuracy of collision detection when sampling nodes and local connections. Although it can generate smooth collision-free paths, ensure the safety of sampling points and edges through collision detection, and have high path reliability, it is assumed that in a configuration space, there are many obstacles very close to each other. The PRM algorithm randomly generates nodes, but the probability of generating nodes between these gaps is very small, so it may generate dangerous paths close to obstacles. In terms of real-time performance, since the core steps of PRM are usually divided into two stages: offline sampling and online sampling, offline sampling and online sampling are fast, but when the environment changes dynamically, the local graph must be resampled or updated, affecting the real-time decision-making ability [6]. In terms of robustness, PRM has a certain tolerance to environmental changes. The graph structure constructed offline can provide a certain path, but in the face of sudden interference or obvious changes, the original graph structure may not be able to respond effectively, so it is necessary to combine online update strategies. In general, PRM performs well in planning in high-dimensional complex environments, but a node enhancement method can be used to gradually replace the original path nodes with new nodes, reduce the number of path turning points, thereby shortening the path length, and use arcs to replace the original path turning points and adjacent broken lines to achieve the purpose of smoothing the path[6]. The PRM method is also widely used in the motion planning of industrial robots and manipulators. Through offline sampling and composition, PRM can effectively cope with complex obstacle environments. In the online stage, it only needs to perform a quick search on the pre-built map to achieve real-time and efficient path planning. This method is widely used in many industrial automation fields, and the planning efficiency and operation safety are well guaranteed.

2.4. RRT (Rapidly Exploring Random Trees)

The RRT algorithm was proposed by Steven in 1998. The original intention was to quickly and effectively explore high-dimensional continuous space to realize robot motion planning and solve the real-time path planning problem of non-holonomic constraint systems (such as drones and vehicles) in high-dimensional space. RRT can quickly expand the search tree in dynamic or unknown environments through random sampling to meet the needs of real-time planning. The basic principle is to start from the starting point, take the starting point x_{init} as the root node of the tree, randomly generate a sample point $x$rand in the entire state space, and find the node x_{near} closest to x_{rand} in the search tree:

$$x_{near} = \arg\min_{x \in T} \|x - x_{rand}\| \tag{5}$$

Starting from the node x_{near}, move a certain step length ϵ along the direction of $x$rand to generate a new node x_{new}:

$$x_{new} = x_{near} + \epsilon \frac{x_{rand} - x_{near}}{\|x_{rand} - x_{near}\|} \tag{6}$$

Where ϵ is the set expansion step length. If there is no collision on the path from x_{near} to x_{new}, x_{new} is added to the search tree. Then repeat the above steps until the search tree is expanded enough to connect the starting point and the target or the preset number of iterations is reached [7].

RRT can adapt to complex environments, including those with non-convex obstacles, dynamically changing environments, and high-dimensional degrees of freedom systems, but, for example, in the narrow passage problem, the search tree expansion of RRT may be hindered, resulting in reduced search efficiency. In terms of learning efficiency, the learning efficiency of RRT is usually determined by its sample complexity, that is, the number of samples required to find a feasible path. Since RRT is based on random sampling, the sample complexity is high, which makes the convergence speed slow. RRT has high search efficiency in high-dimensional space, but in low-dimensional space, the search efficiency may be lower than that of traditional methods (such as A* or Dijkstra) due to randomness. In terms of safety, in a dynamic environment, due to its incremental search characteristics, RRT may not be able to avoid new obstacles in time, increasing the probability of collision. In terms of real-time evaluation, RRT has extremely fast decision-making speed and is

suitable for real-time obstacle avoidance, but path optimization is still a problem, which requires a balance between speed and quality. In terms of robustness, RRT has a certain tolerance for environmental disturbances, but its robustness in a dynamic environment is weak, especially when the environment changes rapidly. In general, RRT performs well in environmental adaptability and real-time performance, and can adapt to high-dimensional complex environments and quickly search for feasible paths. However, there is still room for improvement in learning efficiency, safety, and robustness, especially in a dynamic environment. The anti-interference ability is weak. Improved versions such as RRT*, DRRT, and Informed-RRT* have improved the performance of RRT at different levels. In an autonomous driving system, RRT can plan a path from the current position to the target position in an environment with known obstacles. RRT can also be used for local path planning, real-time obstacle avoidance, and route adjustment. For example, the improved Bi-RRT algorithm satisfies the vehicle's non-complete constraints through high-level path planning and generates a path that is more in line with the vehicle's motion characteristics.

2.5. Hybrid A* Algorithm

The hybrid A* algorithm was proposed by Dmitri Dolgov et al. in the 2010s for use in the field of autonomous driving. It is an algorithm that improves the actual physical constraints of robot motion planning. Its core goal is to use heuristic search to find a path in the continuous state space that is both in line with dynamic constraints and smooth and feasible. The motion planning layer is responsible for calculating a safe, comfortable and dynamically feasible trajectory from the current configuration of the vehicle to the target configuration provided by the behaviour layer in the decision hierarchy, thereby solving the shortcomings of traditional discrete grid planning methods in dealing with continuous constraints such as vehicle steering and minimum turning radius [8]. In practical applications, this algorithm can generate a path that is more in line with physical kinematic requirements. Its basic principle extends the traditional A* algorithm and uses it.

In terms of environmental perception, this algorithm can handle both narrow indoor environments and complex urban roads. Its discrete and continuous hybrid ideas give it additional flexibility. In terms of learning efficiency, this method does not rely on offline learning but is based on state space search. Its sample complexity is mainly affected by the state space partitioning. In high-dimensional continuous control problems, the choice of partitioning structure directly affects the search efficiency and path quality. In safety assessment, Hybrid A* achieves safety in complex dynamic environments by strictly following vehicle kinematic constraints and collision detection; however, multiple paths may be generated during the algorithm search process. If there is a problem with the collision detection efficiency, detection delays may occur, bringing collision risks. However, since this algorithm requires discrete search and continuous smoothing based on the traditional A* algorithm, its decision time is slightly longer than A*. In terms of robustness, the Hybrid A* algorithm is more robust to external interference. The main advantage of hybrid state A* is reflected in manoeuvres in confined spaces, and it can better adjust the path in combination with kinematic constraints [9]. While considering the actual physical constraints, this algorithm also considers the smoothness of the path. It is a relatively mature path planning method in autonomous driving and robot navigation. The Hybrid A* algorithm is often used in narrow spaces such as parking lots. Vehicles need to complete parking operations in a limited space to ensure that the radius of curvature of the path is not less than the minimum turning radius of the vehicle to avoid situations where turning is impossible. The path needs to avoid surrounding obstacles to ensure driving safety. The path's smoothness must also be ensured. The Hybrid A* algorithm generates a path that meets the above requirements by searching in the continuous state space and combining the vehicle kinematic model.

2.6. Covariant Hamiltonian Optimization for Motion Planning

The Covariant Hamiltonian Optimization for Motion Planning (CHOMP) algorithm was proposed around 2009. Its original design was to solve the robot motion trajectory problem through optimization methods. Its goal is to generate a trajectory that is both smooth and can avoid obstacles.

It is suitable for path planning problems in high-dimensional continuous space. The CHOMP algorithm uses the gradient descent method to optimize the objective function in the continuous trajectory space, and the function gradient technology is used to iteratively improve the quality of the initial trajectory, optimizing a function that balances smoothness and obstacle avoidance [10]. The basic principle of the CHOMP algorithm is to represent the entire trajectory ξ as a set of discrete states and construct a cost function. The function is usually divided into smoothing cost # and obstacle cost #. The specific formula is:

$$U(\xi) = f_{prior}(\xi) + f_{obs}(\xi) \tag{7}$$

The smoothing cost $f_{prior}(\xi)$ is used to ensure the continuity and smoothness of the path, which can be expressed as:

$$f_{prior}(\xi) = \frac{1}{2}\sum_{d=1}^{D} w_d \|K_d\xi + e_d\|^2 \tag{8}$$

e_d are constant vectors that encapsulate the contributions from the fixed end points [11]. The obstacle cost $f_{obs}(\xi)$ calculates an additional cost for each node through the predefined obstacle function $V(x)$. The gradient descent method is used to update the trajectory during optimization:

$$\xi_{new} = \xi - \alpha\nabla J(\xi) \tag{9}$$

$Alpha$ is the step size and $\nabla J(\xi)$ is the gradient of the cost function concerning the trajectory; $V(x)$ is a predefined potential function used to measure the proximity of the robot's current position x to obstacles in the environment, namely:

$$V(x) = -dist(x) \tag{10}$$

CHOMP relies on the predefined obstacle potential function $V(x)$, to quantify the "interaction" between the robot's current position and obstacles, and optimizes the trajectory based on gradient descent. Unlike traditional high-dimensional motion planning methods, CHOMP does not divide trajectory generation into different planning and optimization stages, ensuring that the generated trajectory avoids obstacles as much as possible while being smooth. The algorithm starts with an infeasible initial trajectory, and by responding to the surrounding environment, it quickly pulls the trajectory out of collision, while optimizing dynamic quantities such as joint velocity and acceleration, and finally converges to a smooth, collision-free trajectory [10]. In terms of learning efficiency, the CHOMP algorithm does not require offline training data and directly optimizes based on environmental information. However, when the cost function of the problem has multiple local extreme values, the gradient method may fall into local optimality, but in most practical applications, the sample complexity of CHOMP is relatively low, and the convergence speed is fast. In terms of safety, when the obstacle potential function $V(x)$ is properly matched with the external environmental information, the trajectory can effectively avoid the obstacle area and reduce the collision probability. However, since this method relies on gradient descent, the update may not be enough to completely escape from the dangerous area, thereby increasing the risk of collision. Since CHOMP is based on analytical gradient calculation, the iterative process usually has a faster convergence speed, especially when the optimization problem is of moderate scale and the obstacle information is relatively continuous, which can generate updated trajectories quickly. In terms of robustness, if the environmental information is noisy or dynamically changing, the gradient direction may be disturbed, resulting in unstable trajectory updates. In summary, CHOMP performs well in relatively smooth and continuous environments but has relatively low interference tolerance. During the robot's pick-and-place tasks, the robot needs to avoid obstacles and maintain smooth movement. The CHOMP algorithm generates an initial trajectory from the starting point to the target point according to the task requirements, then optimizes the initial trajectory and uses the total cost function of the minimized trajectory to comprehensively consider factors such as the smoothness of the

trajectory and the distance to obstacles. Finally, the optimized trajectory is sent to the robot controller, ensuring the smoothness and safety of the robot when performing tasks, reducing unnecessary vibration and loss, and improving execution efficiency.

2.7. Stochastic Trajectory Optimization for Motion Planning

The Stochastic Trajectory Optimization for Motion Planning (STOMP) algorithm was proposed by Anna Kuindersma around 2013. The core idea is to combine random sampling with optimization methods, reduce computational complexity through gradient-free optimization strategies, improve planning efficiency in high-dimensional space, and find a safe and secure path map through iterative updates. The iterative update process of the STOMP algorithm is to calculate the finite difference matrix.

$$A = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ -2 & 1 & 0 & \cdots & 0 \\ 1 & -2 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix} \tag{11}$$

In the pre-calculation stage, using A to calculate trajectory acceleration $\ddot{\theta} = A\theta$, regenerate the noise trajectory, and generate K strip noise trajectory.

$$\widetilde{\theta_k} = \theta + \epsilon_k \tag{12}$$

Based on the current trajectory θ , where ϵ_k is sampled from a zero-mean normal distribution with a covariance of R^{-1}. For each discrete time step i , calculate the state cost:

$$S\left(\widetilde{\theta_{k,\iota}}\right) = q\left(\widetilde{\theta_{k,\iota}}\right) \tag{13}$$

The noise trajectory, and use the exponential form to calculate the probability weight:

$$P\left(\widetilde{\theta_{k,\iota}}\right) = \frac{e^{-\frac{1}{\lambda}S\left(\widetilde{\theta_{k,\iota}}\right)}}{\sum_{l=1}^{K} e^{-\frac{1}{\lambda}S\left(\widetilde{\theta_{l,\iota}}\right)}} \tag{14}$$

λ is the sensitivity of the adjustment cost. For each time step i, calculate the weighted noise update:

$$\left[\delta\tilde{\theta}\right]_i = \sum_{k=1}^{K} P\left(\widetilde{\theta_{k,\iota}}\right)[\epsilon_k]_i \tag{15}$$

Smooth the update through the matrix M: $\delta\theta = M\delta\tilde{\theta}$, and finally update the trajectory parameters: $\theta \longleftarrow \theta + \delta\theta$ [12]. STOMP uses random sampling to optimize motion trajectories. Its main advantage is that it does not rely on gradient information and supports black box cost functions. By generating many random trajectory samples, STOMP can escape the local optimal trap and find a relatively safe and smooth path in scenes with complex obstacles or frequent changes. In terms of learning efficiency, STOMP has a high sample complexity. In a high-dimensional environment, to ensure full exploration of the trajectory space, more sampling times are often required, which increases the amount of calculation and learning time. In terms of safety, STOMP evaluates the costs of different trajectories, including collision penalty terms, through random sampling, to select safer trajectories from multiple samples. However, due to the randomness of the sampling process, a single update may select some samples with lower costs but higher risks. Therefore, collision detection is often combined with post-processing smoothing measures in practical applications. In terms of real-time performance, since STOMP does not rely on precise gradients, each update can be efficiently executed on a parallel computing platform, which is suitable for applications that require rapid decision-making in complex environments. In terms of robustness, STOMP uses random sampling, which naturally has a certain degree of robustness. Especially in dynamic environments, STOMP can adapt to interference by increasing the number of samples or adjusting the sampling strategy. The STOMP algorithm provides greater flexibility and robustness for robot paths and is particularly

suitable for application scenarios facing nonlinear dynamic environments. It is a practical method in the current field of motion planning.

3. Comparison analysis

In terms of environmental adaptability, the traditional Dijkstra algorithm can have good path planning capabilities in regular environments because it is based on static graph search, but when encountering a dynamic environment, its pre-discretized map limits its ability to respond to environmental changes; the A* algorithm inherits the advantages of Dijkstra while narrowing the search space through heuristic functions, making it more adaptable to environmental changes to a certain extent, but it also relies on discrete maps; the sampling-based PRM method uses random sampling to construct a road map, which has strong adaptability in high-dimensional and complex environments, but its dependence on environmental updates requires the reconstruction of the path map in dynamic scenarios; the RRT algorithm can quickly expand the search tree, better explore complex spaces, and has strong adaptability, but because the generated path is usually not smooth, its adaptability is insufficient in later optimization; Hybrid A* can more accurately adapt to the actual vehicle motion environment by combining A* discrete heuristic search with vehicle dynamics constraints in continuous state space; and the optimization-based CHOMP and STOMP The algorithm optimizes continuous trajectories through gradient descent and random sampling respectively. The former has higher adaptability in environments with smoother obstacle distribution but is prone to fall into local optimality. The latter can better cope with complex or dynamic environments through many samplings, but requires a lot of computing resources to ensure sufficient coverage. In terms of learning efficiency, the Dijkstra algorithm has a high inverse sample complexity due to its characteristic of traversing the entire graph. Although A* uses heuristic information to reduce the number of search nodes, it still has a high sample requirement in a large-scale environment. The efficiency of the PRM method depends on the quantity and quality of pre-sampling. Usually, the sample complexity is large in the offline construction phase, but it performs quickly in online queries. RRT uses a random expansion strategy, so a feasible solution can be found with a small number of samples in the initial stage, but if asymptotic optimality is required, the number of samples needs to be continuously increased. Hybrid A* is more efficient in sample utilization by combining heuristic search with continuous trajectory optimization. CHOMP uses gradient information to converge quickly, and the sample complexity is relatively low, but the gradient calculation may be disturbed by local minima in complex environments. STOMP achieves global search through a gradient-free random sampling method. Although the sample complexity is high, the improvement in parallel computing capabilities can partially make up for this shortcoming. In terms of safety, Dijkstra and A* algorithms rely on the accuracy of discrete maps. If the map accuracy is insufficient, there may be a collision risk, but their path planning process usually guarantees the global optimal solution, thereby reducing the probability of collision. The safety of the PRM method depends on the distribution and connectivity of the sampling points. Insufficient sampling or uneven distribution of obstacles may lead to an increase in the risk of collision. Although the RRT algorithm can find a path quickly, the generated path is indeed smooth, and the smoothness needs to be checked during actual execution to ensure safety. Hybrid A* generates a smooth trajectory through continuous state optimization while considering vehicle dynamics constraints, and its safety is significantly improved compared to traditional discrete search methods. CHOMP uses continuous potential functions to penalize obstacles and can guide the path away from obstacles. STOMP effectively smooths the update process through multiple random sampling and weighted average path updates, reducing the probability of collision caused by single update deviations, so its overall safety is relatively high. In terms of real-time performance, the Dijkstra algorithm needs to traverse the entire graph, which takes a long time to calculate and is not suitable for real-time applications. The A* algorithm greatly reduces the search space with the help of heuristic information, which significantly improves the decision-making speed, but there are still delays in large-scale and complex environments. The PRM

method is usually divided into two stages: offline construction and online query. The online query stage has better real-time performance, but the offline construction time is longer. The RRT algorithm can quickly expand the search tree, and the initial planning is faster, but the path smoothing optimization requires additional time. Hybrid A* combines discrete search with continuous optimization, which can usually control the calculation time better while ensuring accuracy, thereby meeting the real-time requirements. CHOMP relies on the gradient descent algorithm in the trajectory optimization process. It has better real-time performance in lower-dimensional problems but may cause delays in high-dimensional systems due to the increase in the number of iterations. Although STOMP uses parallel sampling and random optimization strategies, the amount of calculation for a single update is large, but modern multi-core processors can achieve a faster update frequency, and the overall real-time performance can reach a high level with sufficient hardware support. In terms of robustness, the Dijkstra and A* algorithms rely heavily on discrete maps and predefined cost functions, are not naturally robust to environmental changes and noise, and are easily affected by map update delays and errors; the PRM method has a certain degree of robustness to noise and uncertainty due to its reliance on random sampling, but its robustness will decrease in the case of insufficient sampling; the RRT algorithm can tolerate interference in the environment to a certain extent through random expansion, but the path quality may fluctuate greatly; Hybrid A* has good robustness and can adapt to sensor noise and model errors within a certain range; although CHOMP performs well in smooth trajectory generation, it relies heavily on gradient information, and may lead to a decrease in robustness once it encounters flat gradients or high noise; STOMP adopts a gradient-free random sampling optimization strategy, which effectively filters out abnormal fluctuations in a single sampling through a large number of sample average updates, thereby showing high interference tolerance and robustness. In general, various algorithms have their advantages and disadvantages in different dimensions. Traditional graph search algorithms (Dijkstra, A*) are suitable for static and structured scenarios, but are easily limited in complex, high-dimensional, and dynamic environments; sampling methods (PRM, RRT) can explore complex spaces quickly, but often require subsequent optimization to improve path smoothness and safety; while hybrid and optimization algorithms (Hybrid A*, CHOMP, STOMP) pay more attention to the comprehensive processing of continuous states and dynamic constraints, and have achieved a good balance in safety, real-time and robustness through optimization and random sampling techniques. The specific choice should be based on the specific needs of the application scenario for environmental adaptability, sample complexity, collision risk, decision delay, and interference tolerance.

4. Conclusion

Based on the five-dimensional evaluation system of environmental adaptability, learning efficiency, safety, real-time, and robustness, this paper systematically reviews robot algorithms such as the Dijkstra algorithm, A* algorithm, PRM, RRT, Hybrid A, CHOMP, and STOMP. By combining many domestic and foreign literature and cases, this paper summarizes the basic principles, performance indicators, and advantages and disadvantages of various algorithms in path planning applications in obstacle environments. Among the traditional shortest path search algorithms, Dijkstra and A* algorithms can provide accurate paths in static and low-complexity environments with accuracy and theoretical optimality, but in large-scale or dynamic environments, due to high computational complexity, there are certain limitations in real-time performance; in contrast, the sampling-based PRM and RRT algorithms can construct a feasible path in a short time by randomly expanding the search space, and are well adapted to complex and high-dimensional spaces, but their path smoothness and path optimality still have room for improvement. The Hybrid A* algorithm has achieved a good balance between heuristic search and vehicle kinematic constraints and is suitable for autonomous driving and path planning in narrow environments. However, its implementation complexity and computational cost are high, and it is highly sensitive to algorithm parameters. The optimized CHOMP and STOMP algorithms significantly improve the smoothness and safety of path

planning, reduce the risk of collision, and improve robustness to a certain extent by continuously improving the initial path. However, they are highly dependent on the initial path and need further optimization in terms of real-time response. Overall, each algorithm has its advantages in different evaluation indicators, and each has its applicable scenarios and limitations. How to reasonably select or integrate multiple algorithms in practical applications has become an important direction for future research. For real-time path planning in large-scale dynamic environments, compatibility with complex kinematic constraints, and robustness improvement under external interference, etc., it is still necessary to explore and improve in theory and engineering applications. This article provides a reference for research in related fields and provides a direction for the design and integration of innovative new algorithms.

References

[1] Mehlhorn, Kurt; Sanders, Peter. Algorithms and Data Structures: The Basic Toolbox. Springer. 2008.

[2] Deng Y, Chen Y, Zhang Y, et al. Fuzzy Dijkstra algorithm for the shortest path problem under an uncertain environment[J]. Applied Soft Computing, 2012, 12 (3): 1231-1237.

[3] Patel, Amit. Heuristics. Amit's Thoughts on Pathfinding, 1997–2025. Web. 16 Apr. 2025.

[4] Dechter R, Pearl J. Generalized best-first search strategies and the optimality of A[J]. Journal of the ACM (JACM), 1985, 32(3): 505-536.

[5] Khanmirza E, Haghbeigi M, Nazarahari M, et al. A comparative study of deterministic and probabilistic mobile robot path planning algorithms. 2017 5th RSI international conference on robotics and mechatronics (ICRoM). IEEE, 2017: 534-539.

[6] Gang L, Wang J. PRM path planning optimization algorithm research. Wseas Transactions on Systems and Control, 2016, 11(81-86): 7.

[7] Karaman S, Frazzoli E. Sampling-based algorithms for optimal motion planning. The International Journal of Robotics Research. 2011;30(7):846-894.

[8] B. Paden, M. Čáp, S. Z. Yong, D. Yershov and E. Frazzoli. A Survey of Motion Planning and Control Techniques for Self-Driving Urban Vehicles. IEEE Transactions on Intelligent Vehicles, 2016, vol. 1, no. 1, pp. 33-55.

[9] Dolgov D, Thrun S, Montemerlo M, et al. Practical search techniques in path planning for autonomous driving. Ann Arbor, 2008, 1001(48105): 18-80.

[10] Zucker M, Ratliff N, Dragan AD, et al. CHOMP: Covariant Hamiltonian optimization for motion planning. The International Journal of Robotics Research. 2013;32(9-10):1164-1193.

[11] N. Ratliff, M. Zucker, J. A. Bagnell and S. Srinivasa. CHOMP: Gradient optimization techniques for efficient motion planning. 2009 IEEE International Conference on Robotics and Automation, Kobe, Japan, 2009, pp. 489-494.

[12] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor and S. Schaal. STOMP: Stochastic trajectory optimization for motion planning. 2011 IEEE International Conference on Robotics and Automation, Shanghai, China, 2011, pp. 4569-4574.