Journal of Intelligent Manufacturing (2022) 33:387–424
https://doi.org/10.1007/s10845-021-01867-z

A review of motion planning algorithms for intelligent robots

Chengmin Zhou[1] · **Bingding Huang**[2] · **Pasi Fränti**[1]

Received: 11 February 2021 / Accepted: 12 October 2021 / Published online: 25 November 2021

Abstract
Principles of typical motion planning algorithms are investigated and analyzed in this paper. These algorithms include traditional planning algorithms, classical machine learning algorithms, optimal value reinforcement learning, and policy gradient reinforcement learning. Traditional planning algorithms investigated include *graph search algorithms*, *sampling-based algorithms*, *interpolating curve algorithms*, and *reaction-based algorithms*. Classical machine learning algorithms include *multiclass support vector machine*, *long short-term memory*, *Monte-Carlo tree search* and *convolutional neural network*. Optimal value reinforcement learning algorithms include *Q learning*, *deep Q-learning network*, *double deep Q-learning network*, *dueling deep Q-learning network*. Policy gradient algorithms include *policy gradient method*, *actor-critic algorithm*, *asynchronous advantage actor-critic*, *advantage actor-critic*, *deterministic policy gradient*, *deep deterministic policy gradient*, *trust region policy optimization* and *proximal policy optimization*. New general criteria are also introduced to evaluate the performance and application of motion planning algorithms by analytical comparisons. The convergence speed and stability of optimal value and policy gradient algorithms are specially analyzed. Future directions are presented analytically according to principles and analytical comparisons of motion planning algorithms. This paper provides researchers with a clear and comprehensive understanding about advantages, disadvantages, relationships, and future of motion planning algorithms in robots, and paves ways for better motion planning algorithms in academia, engineering, and manufacturing.

Keywords Motion planning · Path planning · Intelligent robots · Reinforcement learning · Deep learning

Introduction

Intelligent robot, nowadays, is serving people from different backgrounds in dense and dynamic shopping malls, train stations and airports (Bai et al., 2015) like Daxin in Beijing and Changi in Singapore. Intelligent robots guide pedestrians to find coffee house, departure gates and exits via accurate motion planning, and assist pedestrians in luggage delivery. Another example of intelligent robot is the parcel delivery robot from e-commercial tech giants like JD in China and Amazon in US. Researchers in tech giants make it possible for robots to autonomously navigate themselves and avoid dynamic and uncertain obstacles via applying motion planning algorithms to accomplish parcel delivery tasks. In short, intelligent robot gradually plays a significant role in service industry, agricultural production, manufacture industry and dangerous scenarios like nuclear radiation environment to replace human manipulation, therefore the risk of injury is reduced, and efficiency is improved.

Research of motion planning is going through a flourishing period, due to development and popularity of *deep learning* (DL) and *reinforcement learning* (RL) that have better performance in coping with non-linear problems with *complexity*. The complexity of these problems generally refers to the uncertainty, ambiguity, and incompleteness (Cai et al., 2017), especially the uncertainty that is the most challenging issue in robotic motion planning. Many universities, tech giants, and research groups all over the world therefore attach much importance, time, and energy on developing new motion planning techniques by applying DL algorithms or integrating traditional motion planning algorithms with advanced *machine learning* (ML) algorithms. *Autonomous vehicle* is an example. Among tech giants, Google initiated their self-driving project named Waymo in 2016 (Samuel., 2017). In 2017, Tesla pledged a fully self-driving capable vehicle (Bilbeisi & Kesse, 2017). Autonomous car from

✉ Pasi Fränti
franti@cs.uef.fi

[1] Machine Learning Group, School of Computing, University of Eastern Finland, Joensuu, Finland

[2] College of Big Data and Internet, Shenzhen Technology University, Shenzhen, China

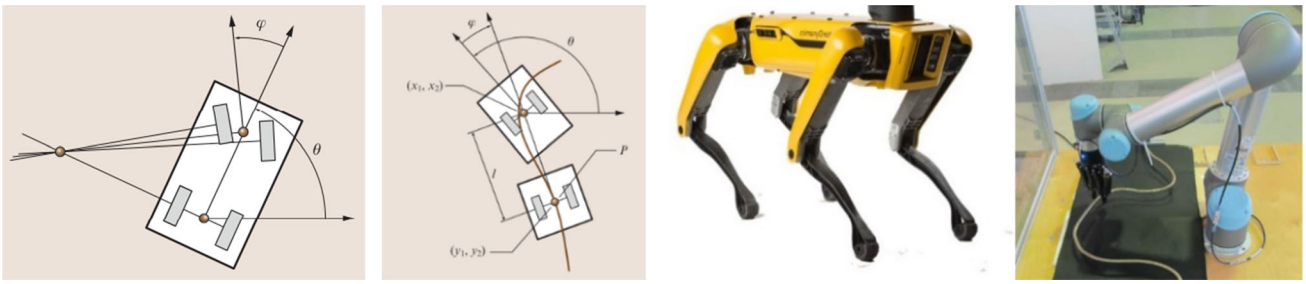


Fig. 1 Three types of robotic platform. The first and second figures represent wheel-based chassis (Minguez et al., 2008). The first figure represents an Ackerman-type (car-like) chassis, while the second figure represents a differential-wheel chassis. The third and fourth figures represent four-leg dog "SpotMini" from Boston Dynamic and the robotic arm (Meyes et al., 2017)

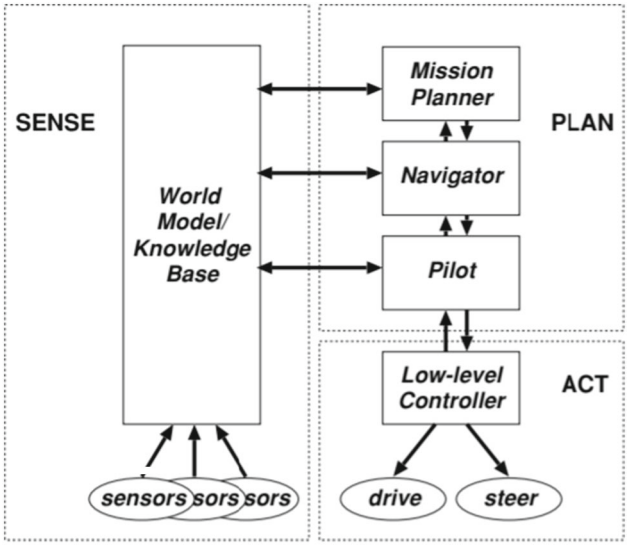
Baidu had been tested successfully in highways of Beijing in 2017 (Fan et al., 2018), and man-manipulated buses had already been replaced by autonomous buses from Huawei in some specific areas of Shenzhen. Other companies in traditional vehicle manufacturing, like Audi and Toyota, also have their own experimental autonomous vehicles. Among research institutes and universities, Navlab (navigation lab) of Carnegie Mellon, Oxford University and MIT are leading research institutes. Up to 2020, European countries like Belgium, France, Italy, and UK are planning to operate transport systems for autonomous vehicles. Twenty-nine US states had passed laws in permitting autonomous vehicles. Autonomous vehicle is therefore expected to widely spread in near future with improvement of traffic laws.

Motion planning and robotic platform Robots use motion planning algorithms to plan their trajectories both at global and local level. Human-like and dog-like robots from Boston Dynamic and autonomous robotic car from MIT (Everett et al., 2018) are good examples. All of them leverage motion planning algorithms to enable robots to freely walk in dense and dynamic scenarios both indoor and outdoor. *Chassis of robots* has two types of wheels, including *Ackerman-type wheel* and *differential wheel* (Fig. 1).

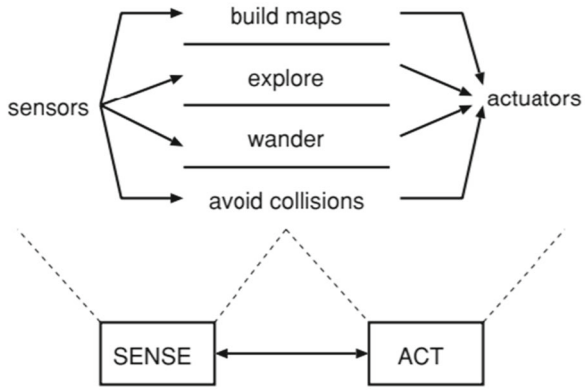
In Ackerman-type robots, two front wheels steer the robot, while two rear wheels drive the robot. The Ackerman-type chassis has two servos. Two front wheels share a same servo, and it means these two wheels can steer with a same steering angle or range φ (Fig. 1). Two rear wheels share another servo to control the speed of robots. The robot using differential wheel, however, is completely different with Ackerman-type robot in functions of servo. The chassis with differential wheels generally has two servos, and each wheel is controlled by one servo for forwarding. Steering is realized by giving different speeds to each wheel. Steering range in Ackerman-type robots is limited because two front wheels steer with a same angle φ. The Ackerman-type wheel is therefore suitable to be used in high-speed outdoor scenarios because of stability. Robots with differential wheels, however, can steer in an $angle \in (0, 2\pi]$, and it means robots can change their *yaw* solely without changing their position (x, y). Robots with differential wheels are also sensitive to the speed difference of two front wheels. The sensitivity depends on the rate of the gearing steer mechanism that yields the speed reduction and angular moment rotation. It means it is flexible to move in low-speed indoor scenarios but very dangerous to move in high-speed situations if something wrong in the speed control of two front wheels, because little speed changes of two front wheels in differential chassis can be exaggerated and accident follows.

It is popular to use *legs* in the chassis of robots in recent years. Typical examples are human-like and animal-like (dog-like, Fig. 1) robots from Boston Dynamic. The robotic *arm* (Fig. 1) is also a popular platform to deploy motion planning algorithms. In summary, wheels, arms, and legs are choices of chassis to implement motion planning algorithms which are widely used in academic and industrial scenarios including commercial autonomous driving, service robot, surgery robot and industrial arms.

Architecture of robots Classical *hierarchical robotic architecture* (Meystel, 1990) in Fig. 2a is composed by three stages: *sense*, *plan* and *act* (Murphy, 2000). Robots with this architecture can be successfully used in simple applications. It can generate long-term action plans, however, researchers are unsatisfied with the slow speed of this architecture in the update of world model and navigation plan, when coping with the environment with uncertainty. *Reactive architecture* (Brooks, 1986) in Fig. 2b, therefore, is introduced to cope with uncertain scenarios. Reactive architecture is designed to output instant response by the *sense-act structure* (Murphy, 2000). Reactive strategies or algorithms (e.g., potential fields) originate from the intuitive response of animals, and they are computationally inexpensive. However, the robot based on reactive architecture is short-sighted. It cannot generate long-term plans to fulfill challenging tasks. *Hybrid deliberative/reactive architecture* in Fig. 2c fuses advantages of hierarchical and reactive architectures, and it is also successfully used in autonomous robots (Arkin et al., 1987; Murphy, 2000). Hybrid deliberative/reactive



(a) Hierarchical architecture (Meystel, 1990; Murphy, 2000).



(b) Reactive architecture (Brooks, 1986; Murphy, 2000).

Fig. 2 Architectures of autonomous robots. (**a**–**c**) denote the classical autonomous robotic architecture, while (**d**) denotes recent trend of autonomous robotic architecture that features the DL and RL

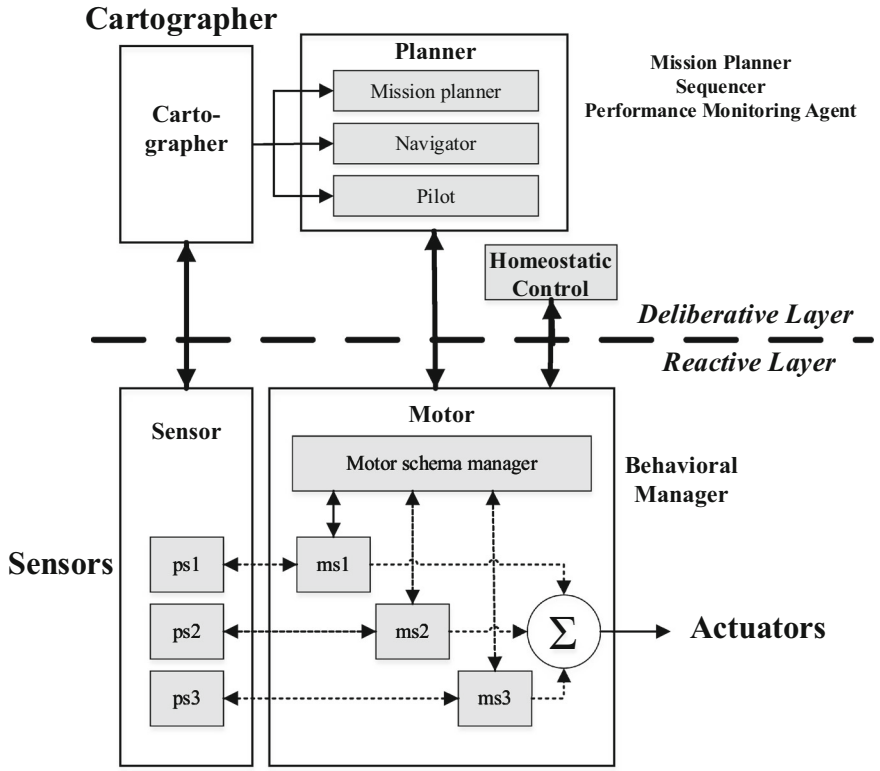
architecture uses the *deliberative layer* to realize high-level long-term planning, while the *reactive layer* is used to realize local reactive planning. Hybrid deliberative/reactive architecture is still widely used in robots nowadays. A typical example is that: (1) in deliberative layer, world maps of the environment are constructed using information from sensors like the *light detection and ranging* (liDAR). Planning algorithms (e.g., A*) then plan high-level paths; (2) in reactive layer, reactive strategies, like *dynamic window approach* (DWA) for local planning and PID for speed control, are used to make local planning or instant reactions to cope with dynamic and uncertain scenarios. Finally, high-level planning, local planning or instant reactions are evaluated in the *behavior manager* to generate a better combined planning.

However, autonomous robotic architecture is evolving towards a simple architecture in Fig. 2d with the development of DL and RL algorithms in recent years. For example, in recent works (Chen et al., 2016, 2019, 2020; Everett et al., 2018; Long et al., 2018): (1) The goal's information (e.g., position of goals), sensor's information (e.g. distances to other robots) and attributes of robots (e.g., radius) are combined to form the features of robots; (2) More features of robots are obtained by interacting with the environment, and feedbacks (rewards) are obtained accordingly; (3) These features are recorded by the networks as the world model that will be updated according to feedbacks, and it is followed by obtaining a converged model; (4) Previous procedures are defined as the *trainer* to obtain the world model, and then time-sequential actions are generated in the *navigator* by performing the trained world model to navigate robots to destinations; (5) However, these time-sequential actions cannot be recognized by the *actuators* of robots (e.g., motor), therefore it is necessary to use the *parser* to parse them to proper formats that can be executed by actuators. This architecture of current autonomous robots can be simply described as five functional modules: *feature extraction, environment perception, environment understanding, time-sequential navigation,* and *decision execution*. It can be also simply divided into three stages: *sense and train*, *plan*, and *act*.

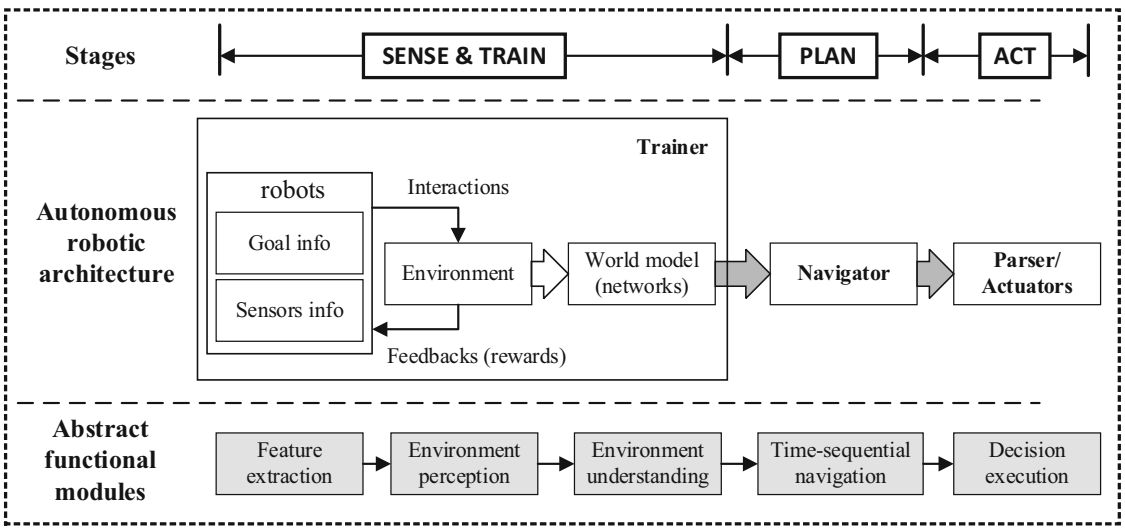
The advantages of recent autonomous robotic architecture are the *simplicity*, *safety*, and *efficiency*. Unlike Fig. 2c, there is no clear *boundary* between deliberative and reactive layers in Fig. 2d. Time-sequential planning can realize long-term planning goals, local planning goals or quick response with safety (e.g., safe distances to other objects) and efficiency (e.g., shortest path, shortest time) at the same time. Disadvantages of recent autonomous robotic architecture are *expensive computation cost* and *poor network convergence* especially when networks are trained with data from large outdoor scenarios.

Motion planning and path planning *Motion planning* is the extension of *path planning*. They are almost the same term, but few differences exist. For example, path planning aims at finding the path between the origin and destination in *workspace* by strategies like shortest distance or shortest time (Fig. 3), therefore path is planned from the global metric or topological level. Motion planning, however, aims at generating interactive trajectories in workspace when robots interact with dynamic environment, therefore motion planning needs to consider kinetics features, velocities and poses of robots and dynamic objects nearby (Fig. 3) when robots move towards the goal. Note that workspace here is an area where an algorithm works, or the task exists.

To conclude, on one hand, motion planning must consider short-term optimal or suboptimal reactive strategies to make instant or reactive response. This is achieved by rotary or linear control in hardware (e.g., motor, servo) from the perspective of robotic and control engineering. On the



(c) Hybrid deliberative/reactive architecture for autonomous robots (Arkin et al., 1987; Murphy, 2000)



(d) Recent trend of autonomous robotic architecture.

Fig. 2 continued

other hand, motion planning should achieve long-term optimal planning goals as path planning when robots interact with the environment.

Classification of planning algorithms Robotic planning algorithms can be divided into two categories: *traditional algorithms* and *ML-based* algorithms according to their principles and the era they were invented. Traditional algorithms are composed by four groups including *graph search algo-*

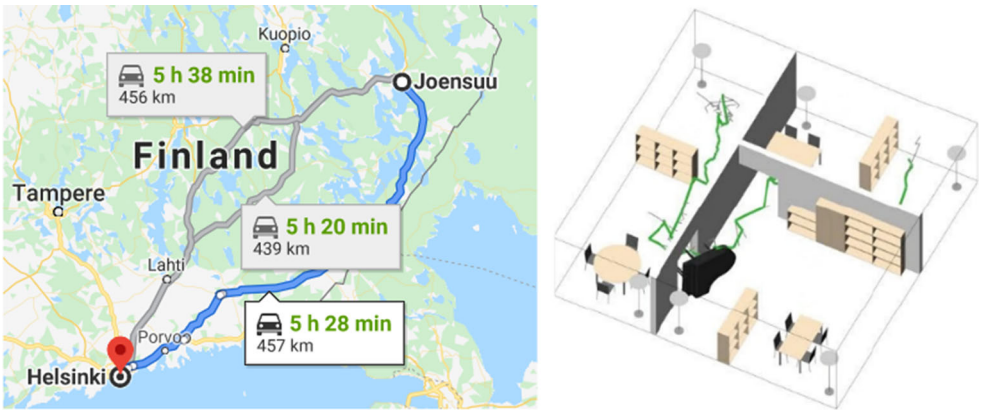


Fig. 3 Path planning and motion planning. The left figure denotes a planned path based on shortest distance and time, and path is generated from high or global level. The right figure denotes famous piano mover's problem that not only consider planning a path from global level, but also consider kinetics features, speeds and poses of the piano

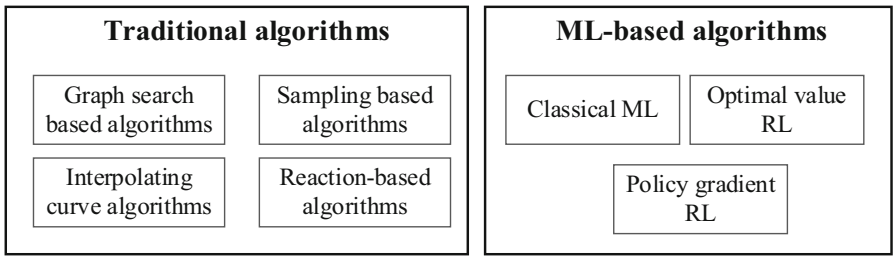


Fig. 4 Two categories of robotic planning algorithm

rithms (e.g., A*), *sampling-based algorithms* like *rapidly-exploring random tree* (RRT), *interpolating curve algorithms* (e.g., *line and circle*), and *reaction-based algorithms* (e.g., DWA). ML based planning algorithms include *classical ML algorithms* like *support vector machine* (SVM), *optimal value RL* like *deep Q-learning network* (DQN) and *policy gradient RL* (e.g., *actor-critic algorithm*). Categories of planning algorithms are summarized in Fig. 4.

Development of ML-based algorithms *Classical ML*, like SVM, are used to implement simple motion planning at an earlier stage, but its performance is poor because SVM is short-sighted for its one-step prediction. It requires well-prepared vector as inputs that cannot fully represent features of image-based dataset. Significant improvement to extract high-level features from images were made after the invention of *convolutional neural network* (CNN) (Lecun et al., 1998). CNN is widely used in many image-related tasks including motion planning, but it cannot cope with complex time-sequential motion planning problems. These better suit *Markov chain* (Chan et al., 2012) and *long short-term memory* (LSTM) (Inoue et al., 2019). Neural networks are then combined with LSTM or algorithms that are based on Markov chain (e.g., *Q learning* (Smart & Kaelbling, 2002)) to implement time-sequential motion planning. However, the efficiency is limited (e.g., poor performance in network convergence). A breakthrough was made when Google DeepMind introduced nature DQN (Mnih et al., 2013, 2015), in which *reply buffer* is to reuse old data to improve the efficiency. Performance in robustness, however, is limited because of noise that impacts the estimation of state-action value (Q value). *Double DQN* (Hasselt et al., 2016; Sui et al., 2018) and *dueling DQN* (Wang et al., 2015) are therefore invented to cope with problems caused by noise. Double DQN utilizes another network to evaluate the estimation of Q value in DQN to reduce noise, while advantage value (A value) is utilized in dueling DQN to obtain better Q value, and noise is mostly reduced. The Q learning, DQN, double DQN and dueling DQN are all based on optimal values (Q value and A value) to select optimal time-sequential actions. These algorithms are therefore called *optimal value algorithms*. Implementation of optimal value algorithms, however, is computationally expensive.

Optimal value algorithms are latter replaced by *policy gradient method* (Sutton et al., 1999), in which *gradient approach* (Zhang, 2019) is directly utilized to upgrade *policy* that is used to generate optimal actions. Policy gradient method is more stable in network convergence, but it lacks efficiency in speed of network convergence. *Actor-critic algorithm* ((Cormen et al., 2009; Konda & Tsitsiklis, 2001)) improves the speed of convergence by the actor-critic architecture. However, improvement in convergence speed is

achieved by sacrificing the stability of convergence, therefore the network of actor-critic algorithm is hard to converge in earlier-stage training. *Asynchronous advantage actor-critic* (A3C) (Gilhyun, 2018; Mnih et al., 2016), *advantage actor-critic* (A2C)[1] (Babaeizadeh et al., 2016), *trust region policy optimization* (TRPO) (Schulman et al., 2017a) and *proximal policy optimization* (PPO) (Schulman et al., 2017b) algorithms are then invented to cope with this shortcoming. *Multi-thread technique* (Mnih et al., 2016) is utilized in A3C and A2C to accelerate the speed of convergence, while TRPO and PPO improve the policy of actor-critic algorithm by introducing *trust region constraint* in TRPO, and "surrogate" and adaptive penalty in PPO to improve the speed and stability of convergence. Data, however, is dropped after training, and new data must therefore be collected to train the network until convergence of network.

Off-policy gradient algorithms including *deterministic policy gradient* (DPG) (Silver et al., 2014) and *deep DPG* (DDPG) ((Lillicrap et al., 2019; Munos et al., 2016)) are invented to reuse data by replay buffer. DDPG fuses the actor-critic architecture and *deterministic policy* to enhance the convergence speed. In summary, classical ML, optimal value RL, and policy gradient RL are typical ML algorithms in robotic motion planning, and the development of these ML-based motion planning algorithms is shown in Fig. 5.

In this paper, state-of-art ML-based algorithms are investigated and analyzed to provide researchers with a comprehensive and clear understanding about functions, structures, advantages, and disadvantages of planning algorithms. We also summarize new criteria to evaluate the performance of planning algorithms. Potential directions for making practical optimization in motion planning algorithms are discussed simultaneously. Contributions of this paper include: (1) Survey of traditional planning algorithms. (2) Detailed investigations of classical ML, optimal value RL and policy gradient RL for robotic motion planning. (3) Analytical comparisons of these algorithms according to new evaluation criteria; (4) Analysis of future directions.

This paper is organized as follows: Sects. "Traditional planning algorithms", "Classical ML", "Optimal value RL" and "Policy gradient RL" present principles and applications of traditional planning algorithms, classical ML, optimal value RL and policy gradient RL in robotic motion planning; section VI presents analytical comparisons of these algorithms, and criteria for performance evaluation; section VII analyzes future directions of robotic motion planning.

[1] OpenAI Baselines: ACKTR and A2C. Web. August 18, **2017**. https://openai.com/blog/baselines-acktr-a2c.

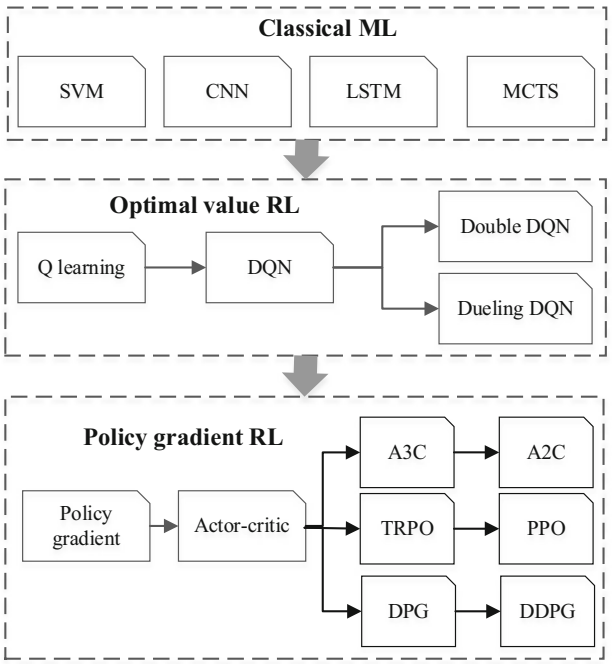


Fig. 5 Development of ML-based robotic motion planning algorithms. These algorithms evolve from classical ML to optimal value RL and policy gradient RL. Classical ML cannot address time-sequential planning problem but RL copes with it well. Optimal value RL suffers slow and unstable convergence speed but policy gradient RL performs better in network convergence

Traditional planning algorithms

Traditional planning algorithms can be divided into four groups: *graph-search*, *sampling-based, interpolating curve,* and *reaction-based* algorithms. They will be described in detail in the following sections.

Graph-search algorithms

Graph-search algorithms can be divided into *depth-first search*, *breadth-first search*, and *best-first search* (Dijkstra, 1959). The depth-first search algorithm builds a search tree as deep and fast as possible from the origin to destination until a proper path is found. The breadth-first search algorithm shares similarities with the depth-first search algorithm by building a search tree. The search tree in the breadth-first search algorithm, however, is accomplished by extending the tree as broad and quick as possible until a proper path is found. The best-first search algorithm adds a numerical criterion (value or cost) to each node and edge in the search tree. According to that, the search process is guided by calculation of values in the search tree to decide: (1) whether the search tree should be expanded; (2) which branch in the search tree should be extended. The process of building search trees repeats until a proper path is found. Graph

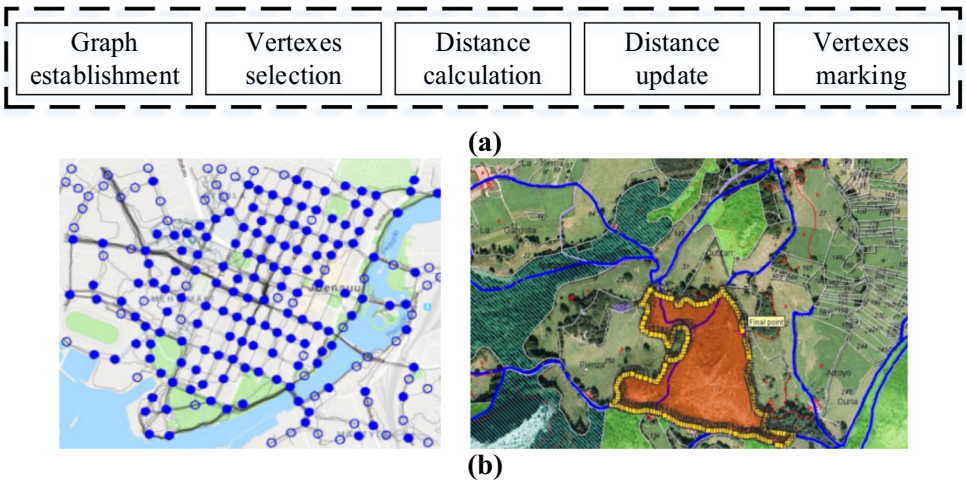


Fig. 6 Steps of the Dijkstra algorithm (**a**) and road networks in web maps (**b**) (Indrajaya et al., 2015; Mariescu & Franti., 2018). Web maps are based on GPS data. Road network is mapped into the graph that is composed by nodes and edges, therefore graph search algorithms like A* and Dijkstra's algorithms can be used in these graphs

search algorithms are composed by many algorithms. The most popular are *Dijkstra's algorithm* (Dijkstra, 1959) and *A* algorithm* (Hart et al., 1968).

Dijkstra's algorithm is one of earliest optimal algorithms based on best-first search technique to find the shortest paths among nodes in a graph. Finding the shortest paths in a road network is a typical example. Steps of the Dijkstra algorithm (Fig. 6) include: (1) converting the road network to a graph, and distances between nodes in the graph are expected to be found by exploration; (2) picking the unvisited node with the lowest distance from the source node; (3) calculating the distance from the picked node to each unvisited neighbor and update the distance of all neighbor nodes if the distance to the picked node is smaller than the previous distance; (4) marking the visited node when the calculation of distance to all neighbors is done. Previous steps repeat until the shortest distance between origin and destination is found. Dijkstra's algorithm can be divided into two versions: *forward version* and *backward version*. Calculation of overall cost in the backward version, called *cost-to-come*, is accomplished by estimating the minimum distance from selected node to destination, while estimation of overall cost in the forward version, called *cost-to-go*, is realized by estimating the minimum distance from selected node to the initial node. In most cases, nodes are expanded according to the *cost-to-go*.

A* algorithm is based on the best-first search, and it utilizes heuristic function to find the shortest path by estimating the overall cost. The algorithm is different from the Dijkstra's algorithm in the estimation of the path cost. The cost estimation of a node i in a graph by A* is as follows: (1) estimate the distance between the initial node and node i; (2) find the nearest neighbor j of the node I; and estimate the distance of nodes$j$ and i; (3) estimate the distance between the node j and the goal node. The overall estimated cost is the sum of these three factors:

$$C_i = c_{start,i} + min_j\left(d_{i,j} + d_{j,goal}\right) \tag{1}$$

where C_i represents overall estimated cost of node i, $c_{start,i}$ the estimated cost from the origin to the node i, $d_{i,j}$ the estimated distance from the node i to its nearest node j, and $d_{j,goal}$ the estimated distance from the node j to the node of goal. A* algorithm has a long history in path planning in robots. A common application of the A* algorithm is mobile rovers planning via an occupancy grid map (Fig. 7) using the Euclidean distance (Wang, 2005). There are many variants of A* algorithm, like *dynamic A** and *dynamic D** (Stentz, 1994), *Field D** (Ferguson & Stentz, 2006), *Theta** (Daniel et al., 2014), *Anytime Repairing A** (ARA*) and *Anytime D** (Likhachev et al., 2008), *hybrid A** (Montemerlo et al., 2008), and *AD** (Ferguson et al., 2008). Other graph search algorithms have a difference with common robotic grid map. For example, the *state lattice algorithm* (Ziegler & Stiller, 2009) uses one type of grid map with a specific shape (Fig. 7), while the grid in normal robotic map is in a square-grid shape (Fig. 7).

Sampling-based algorithms

Sampling-based algorithms randomly sample a fixed workspace to generate sub-optimal paths. The RRT and the *probabilistic roadmap method* (PRM) are two algorithms that are commonly utilized in motion planning. The RRT algorithm is more popular and widely used for commercial

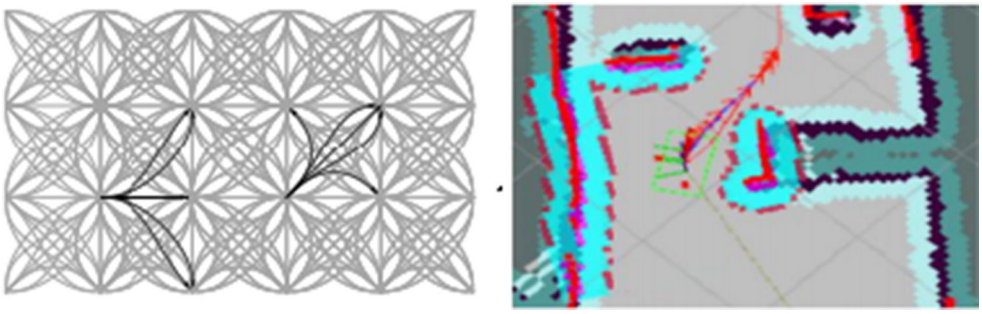


Fig. 7 The left figure represents a specific grid map in the State Lattice algorithm (Ziegler & Stiller, 2009), while the right figure represents a normal square-grid (occupancy grid) map in the *robot operating system* (ROS)

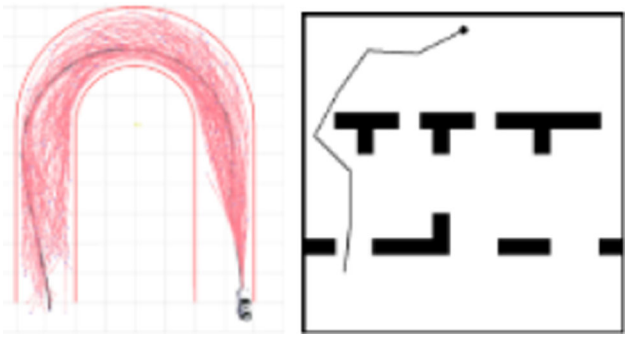


Fig. 8 Trajectories planned by the RRT and PRM. The left figure represents trajectories planned by RRT algorithm (Jeon et al., 2013), and the right figure represents the trajectory planned by PRM algorithm (Kavraki et al., 2002)

and industrial purposes. It constructs a tree that attempts to explore the workspace rapidly and uniformly via a random search (LaValle & Kuffner, 1999). The RRT algorithm can consider non-holonomic constraints, such as the maximum turning radius and momentum of the vehicle (Bautista et al., 2015). The example of trajectories generated by RRT is shown in Fig. 8. The PRM algorithm (Kavraki et al., 2002) is normally used in a static scenario. It is divided into two phases: *learning phase* and *query phase*. In the learning phase, a collision-free probabilistic roadmap is constructed and stored as a graph. In query phase, a path that connects original and targeted nodes is searched from the probabilistic roadmap. An example of trajectory generated by PRM is shown in Fig. 8.

Interpolating curve algorithms

Interpolating curve algorithm is defined as a process that constructs or inserts a set of mathematical rules to draw trajectories. The interpolating curve algorithm is based on techniques, e.g., *computer aided geometric design* (CAGD), to draw a smooth path. Mathematical rules are used for path smoothing and curve generation. Typical path smoothing and curve generation rules include *line and circle* (Reeds & Shepp, 1990), *clothoid curves* (Funke et al., 2012), *polynomial curves* (Xu et al., 2012), *Bezier curves* (Bautista et al., 2014) and *spline curves* (Farouki & Sakkalis, 1994). Examples of trajectories are shown in Fig. 9.

Reaction-based algorithms

Unlike graph-search algorithms that cost longer time to plan high-level or global-level paths, reaction-based algorithms are about making reactions or doing local path planning quickly and intuitively, as the description of algorithms in reactive architecture (Murphy, 2000). Here three reaction-based algorithms that are widely used in engineering and manufacturing are presented, and they are *potential field method* (PFM), *velocity obstacle method* (VOM), and DWA.

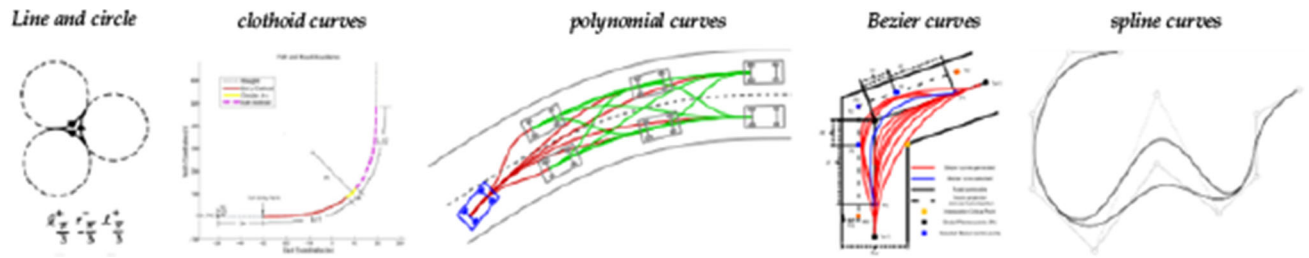


Fig. 9 Trajectories generated by mathematical rules (Bautista et al., 2014; Farouki & Sakkalis, 1994; Funke et al., 2012; Reeds & Shepp, 1990; Xu et al., 2012)

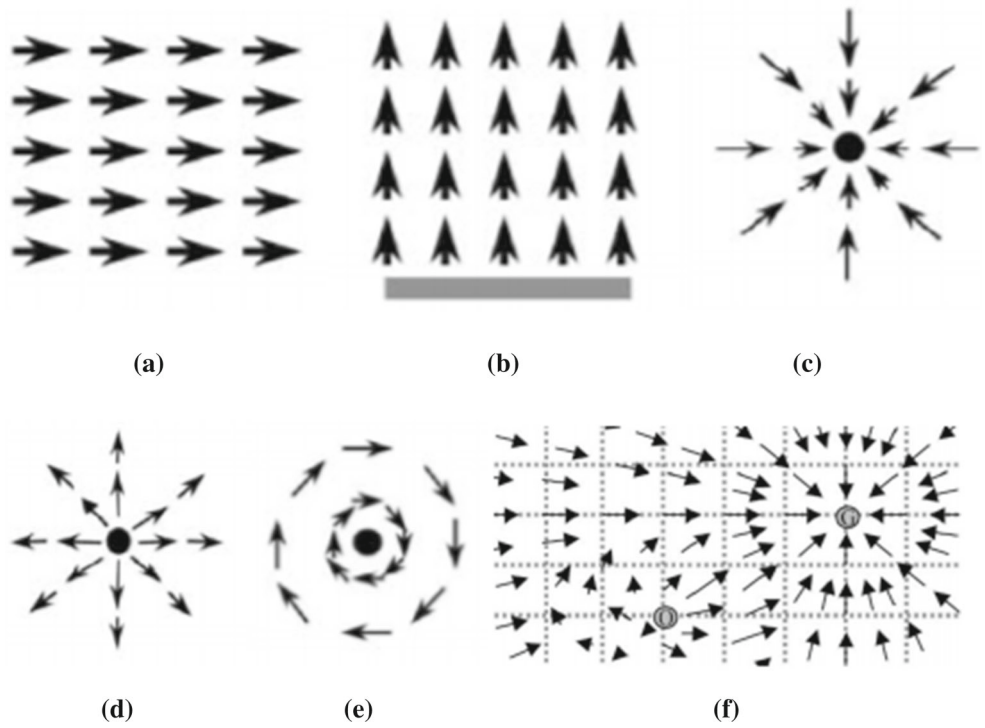


Fig. 10 Different types of potential filed. (**a**–**e**) denote five primitive potential fields: *uniform*, *perpendicular*, *attraction*, *repulsion*, and *tangential*. (**f**) denotes a potential field combined by *attraction (goal)* and *repulsion (obstacle)* (Murphy, 2000)

PFM (Khatib, 1986) is about using *vectors* to represent behaviors and using *vector summation* to combine vectors from different behaviors to produce an emergent behavior (Murphy, 2000). Potential field is a differentiable real-valued function U whose value can be seen as energy, and its gradient can be seen as a *force*. If potential field function U is defined artificially, it is called *artificial potential field* (APF). Its gradient $\nabla U(x)$, where x denotes a robot configuration (e.g., positions of robots), is a vector which points at a local direction that maximally increases U (Tobaruela, 2012). Hence, robots in potential field or combined potential field (Fig. 10) will be forced to move along the gradient of potential field to maximize U.

Shortcomings of PFM include: (1) *local minima* if potential field converges to a minimum that is not global minimum. (2) *oscillation of motion* when robots navigate among very close obstacles at high speed. (3) *impossibility to go through small openings*. These shortcomings can be solved or partially solved by potential field variants (e.g., *generalized potential fields method* (GPFM) (Krogh, 1984), *virtual force field* (VFF) (Borenstein & Koren, 1989), *vector field histogram* (VFH) (Borenstein & Koren, 1991) and *harmonic potential field* (HPF) (Masoud, 2007)) in real-world engineering and manufacturing.

VOM (Fiorini & Shiller, 1998) relies on *current positions* and *velocities* of robots and obstacles to compute a *reachable avoidance velocity space* (RAV), and then a proper avoidance maneuver (velocity) is selected from RAV to avoid static and moving obstacles (Fiorini & Shiller, 1998). To compute a RAV (Fig. 11): (1) *Velocity obstacle* (VO) must be obtained. VO is a velocity set or space, and the selection of velocity from VO will lead to collision. (2) A set of *reachable velocities* (RV) should be obtained. This is achieved by mapping the actuator constraints to acceleration constraints (Fiorini & Shiller, 1998). (3) RAV is obtained by computing the difference between RV and VO.

To select a proper avoidance maneuver (Fig. 11), *exhaustive global search* method and *heuristic search* method in RAV are suitable for *off-line* and *on-line* cases, respectively: (1) A search tree can be obtained by expanding the tree on RAV. A proper avoidance maneuver can be selected from the search tree according to assigned *cost* on the branch of search tree. Cost is relevant with some objective functions (e.g., distance traveled, motion time and energy). The search tree is expanded off-line, therefore near-optimal trajectories that lead to shortest time or distance can be obtained (Fiorini & Shiller, 1998). (2) The heuristic search costs less time on search process, and it is designed to select specific velocities that can realize special goals (e.g., the highest avoidance velocity towards goals, the maximum avoidance velocity, and velocities that ensure desired trajectory structures).

However, collisions with obstacles still exist when using velocity obstacle method in complex scenarios like dense and dynamic cases. Hence, some optimized velocity obstacle

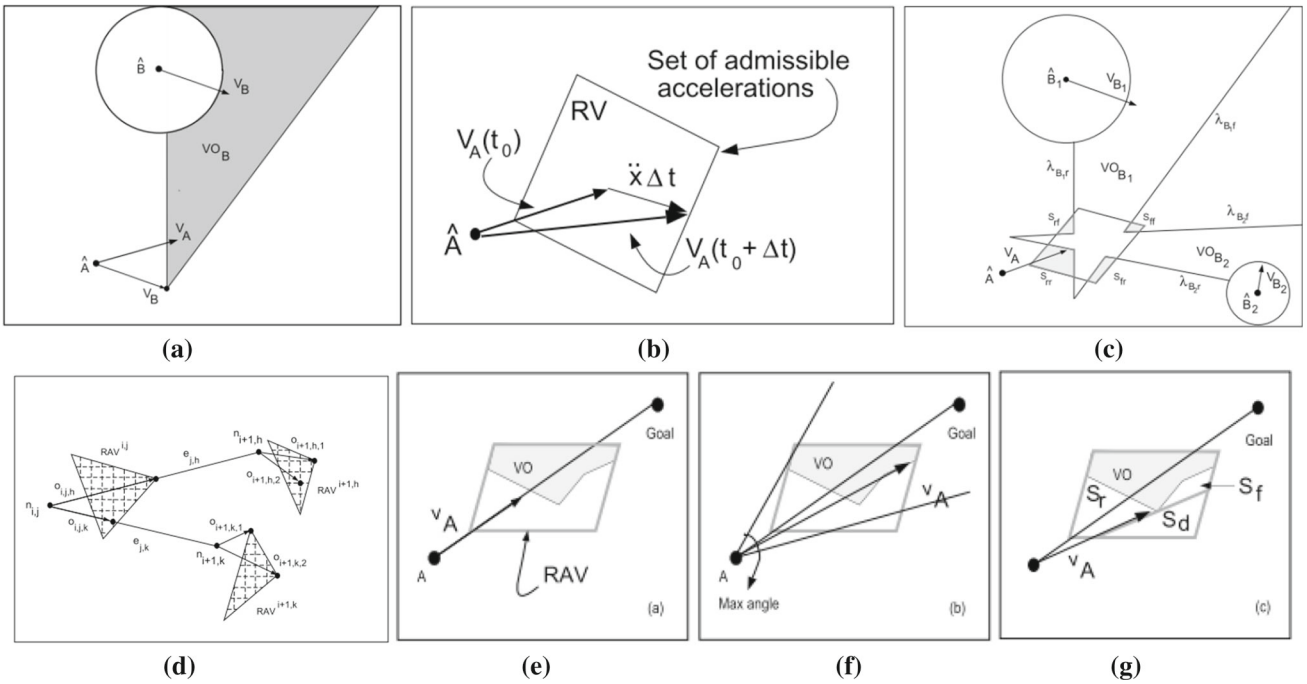


Fig. 11 The principle of VOM. (**a**)–(**c**) denote the VO, RV, and RAV. (**d**) denotes the exhaustive search in the search tree. (**e**)–(**g**) denote heuristic search method to select the highest avoidance velocity towards goals, the maximum avoidance velocity, and velocities that ensure desired trajectory structures (Fiorini & Shiller, 1998)

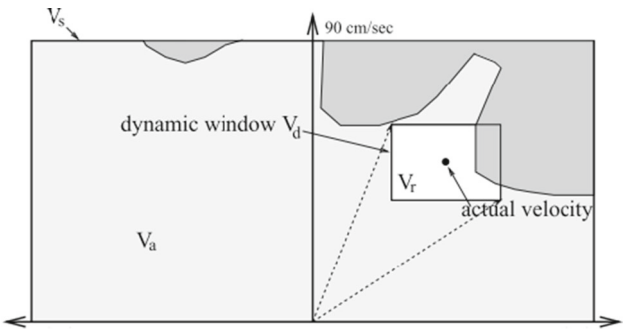


Fig. 12 The relationship of possible velocity search space V_s, admissible velocities V_a, dynamic window velocity V_d, and resulting velocity V_r (Stentz, 1994)

methods, like *reciprocal velocity obstacle* (RVO) (Berg et al., 2008, 2011; Guy et al., 2009), are introduced to better avoid collisions.

DWA (Fox et al., 1997) is about choosing a proper translational and rotational velocity $(\boldsymbol{v}, \boldsymbol{w})$ that will maximize an *objective function* within *dynamic window*. Objective function includes a *measure of progress towards a goal location*, the *forward velocity of the robot*, and the *distance to the next obstacle on the trajectory*. Proper velocity $(\boldsymbol{v}, \boldsymbol{w})$ is selected within the dynamic window (a search space of velocity) which consists of the velocities reachable within a short time interval. This is achieved by: (1) computing a two-dimensional *possible velocity search space* V_s that is related to circular trajectories (curvatures) uniquely determined by $(\boldsymbol{v}, \boldsymbol{w})$. (2) computing *admissible velocities* V_a that ensures the stop before the robot reaches the closest obstacle on the corresponding curvature. (3) computing *dynamic window velocity* V_d. All curvatures outside the dynamic window cannot be reached within the next time interval. (4) selecting a proper velocity from *resulting search space* which consists of *resulting velocity* V_r defined by $V_r = V_s \cap V_a \cap V_d$ (Fox et al., 1997). The relationship of these velocity search spaces is shown in Fig. 12 where the resulting search space is the white area in the dynamic window.

Classical ML

Here basic principles of four classical but pervasive ML algorithms for motion planning are presented. These algorithms include three supervised learning algorithms (SVM, LSTM and CNN) and one RL that is the *Monte-Carlo tree search* (MCTS).

SVM (Evgeniou & Pontil, 1999) is a well-known supervised learning algorithm for classification. The basic principle of SVM is about drawing an optimal separating hyperplane between inputted data by training a maximum margin classifier (Evgeniou & Pontil, 1999). Inputted data is in the form of vector that is mapped into high-dimensional

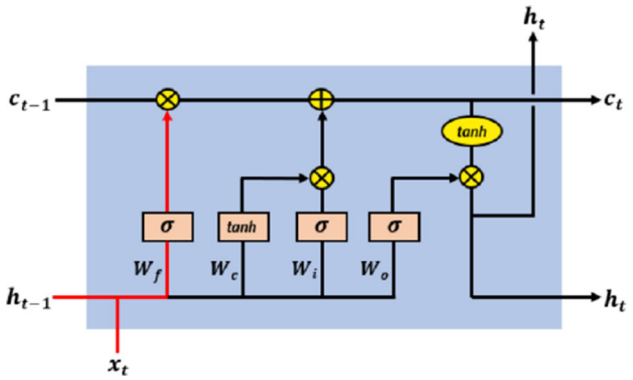


Fig. 13 Cells of LSTM that are implemented using neural network (N. Arbel. How LSTM networks solve the problem of vanishing gradients. Web. Dec 21, 2018. https://medium.com/datadriveninvestor/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients). c_t denotes cell's state in time step t. h_t denotes the output that will be transferred to the next state as its input, therefore format of input is the vector $[h_{t-1}, x_t]$. Cell states are controlled and updated by three gates (*forget gate*, *input gate* and *output gate*) that are implemented using neural networks with weights W_f, $W_c + W_i$, and W_o respectively

space where classified vectors are obtained by performing trained classifier. SVM is used in 2-class classification that cannot suit real-world task, but its variant *multiclass SVM* (MSVM) (Weston & Watkins, 1998) works.

LSTM ((Hochreiter & Schmidhuber, 1997; Inoue et al., 2019)) is a variant of *recurrent neural network* (RNN). LSTM can remember inputted data (vectors) in its cells. Because of limited capacity of cell in storage, a part of data will be dropped when cells are updated with past and new data, and then a part of data will be remembered and transferred to next time step. These functions in cells are achieved by neural network as the description in Fig. 13. In robotic motion planning, robots' features and labels in each time step are fed into neural networks in cells for training, therefore decisions for motion planning are made by performing trained network.

MCTS is a classical RL algorithm, and it is the combination of Monte-carlo method (Kalos & Whitlock, 2008) and the search tree (Coulom, 2006). MCTS is widely used in games (e.g., Go and chess) for motion prediction ((Paxton et al., 2017; Silver et al., 2016)). Mechanism of MCTS is composed by four processes that include *selection*, *expansion*, *simulation*, and *backpropagation* as Fig. 14. In robotic motion planning, node of MCTS represents possible state of robot, and stores state value of robot in each step. First, selection is made to choose some possible nodes in the tree based on known state value. Second, tree expands to unknown state by tree policy (e.g., random search). Third, simulation of expansion is made on new-expanded node by default policy (e.g., random search) until terminal state of robot and reward R is obtained. Finally, backpropagation is made from new-expanded node to root node, and state values in these nodes are updated according to received reward. These four processes are repeated until the convergence of state values in the tree. The robot can therefore plan its motion according to state values in the tree. MCTS fits discrete-action tasks (e.g., AlphaGo (Silver et al., 2016)), and it also fits time-sequential tasks like autonomous driving.

CNN (Lecun et al., 1998) has become a research focus of ML after *LeNet5* (Lecun et al., 1998) was introduced and successfully applied into handwritten digits recognition. CNN is one of the essential types of neural network because it is good at extracting high-level features from high-dimensional high-resolution images by convolutional layers. CNN makes the robot avoid obstacles and plans motions of robot according to human experience by models trained in *forward propagation* and *back propagation* processes, especially the back propagation. In the back propagation, a model with a weight matrix/vector θ is updated to record features of obstacles. Note that $\theta = \{w_i, b_i\}_i^L$ where w and b represent weight and bias, and i represents the serial number of w-b pairs. L represents the length of weight.

Training steps of CNN are shown as Fig. 15. Images of objects (obstacles) are used as inputs of CNN. Outputs are *probability distributions* obtained by *softmax function* (Bridle, 1990). *Loss value* $Loss_{CE}$ is *cross-entropy* (CE) and that is obtained by

$$Loss_{CE} = -\sum_i p_i \cdot \log q_i \tag{2}$$

where p denotes probability distributions of output (observed real value), q represents probability distributions of expectation ($p, q \epsilon (0, 1)$), and i represents the serial number of each batch of images in training. The loss function measures the difference (distance) of observed real value p and expected value q. *Mean-square error* (MSE) is an alternative of CE and MSE is defined by $Loss_{MSE} = \sum_i (p_i - q_i)^2$ where p_i represents observed values while q_i represents predicted values or expectation. The weight is updated in *optimizer* by minimizing the loss value using *gradient descent approach* (Zhang, 2019) therefore new weight w_i^{new} is obtained by

$$w_i^{new} = w_i - \eta \cdot \frac{\partial Loss}{\partial w_i} \tag{3}$$

where w represents the weight, η represents a learning rate ($\eta \epsilon (0, 1)$) and$i$ represents the serial number of each batch of images in training. Improved variants of CNN are also widely used in motion planning, e.g., *residue networks* (Gao et al., 2017; He et al., 2016).

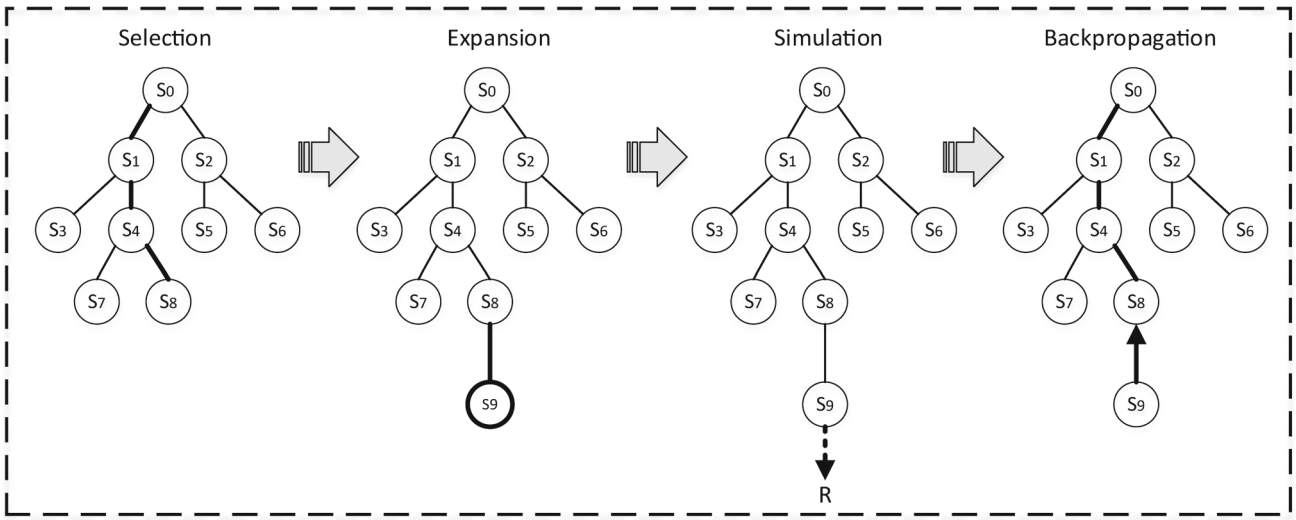


Fig. 14 Four processes of MCTS. These processes repeat until the convergence of state values in the tree

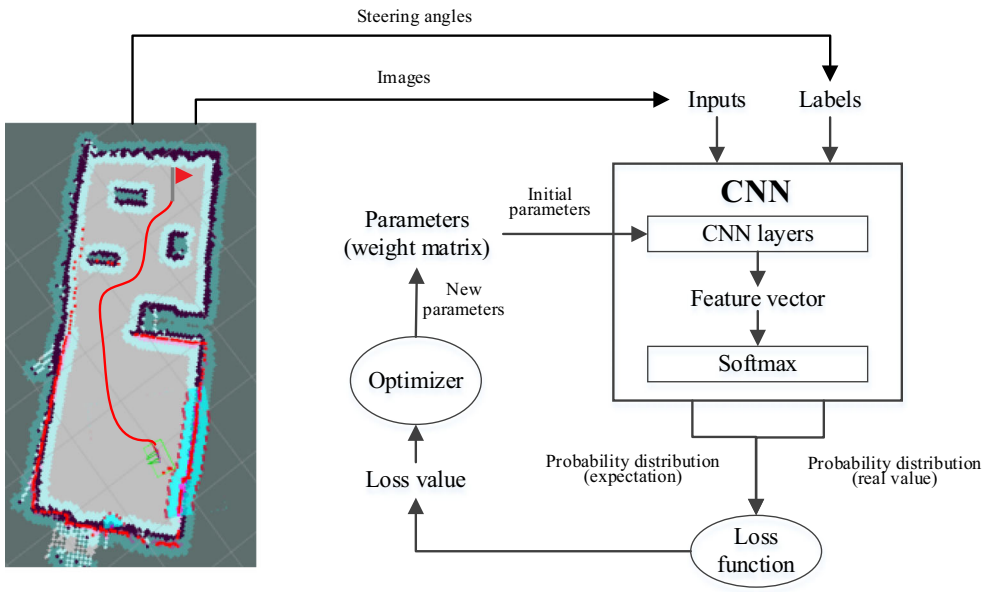


Fig. 15 Training steps of CNN. The trajectory is planned by human in data collection in which steering angles of robots are recorded as labels of data. Robots learn behavior strategies in training and move along the planned trajectory in the test. The softmax function maps values of feature to probabilities $p \in (0, 1)$. The optimizer represents gradient descent approach, e.g., *stochastic gradient descent* (SGD) (Zhang, 2019)

Optimal value RL

Here basic concepts of RL are recalled firstly, and then the principles of Q learning, nature DQN, double DQN and dueling DQN are given.

Classical ML algorithm like CNN is competent only in static obstacle avoidance by one-step prediction, therefore it cannot cope with time-sequential obstacle avoidance. RL algorithms, e.g., optimal value RL, fit time-sequential tasks. Typical examples of these algorithms include Q learning, nature DQN, double DQN and dueling DQN. Motion planning is realized by attaching destination and safe paths with big *reward* (numerical value), while obstacles are attached with *penalties* (negative reward). Optimal path is found according to total rewards from initial place to destination. To better understand optimal value RL, it is necessary to recall several fundamental concepts: *Markov chain*, *Markov decision process* (MDP), *model-based dynamic programming*, *model-free RL*, *Monte-Carlo method* (MC), *temporal difference method* (TD), and *State-action-reward-state-action*

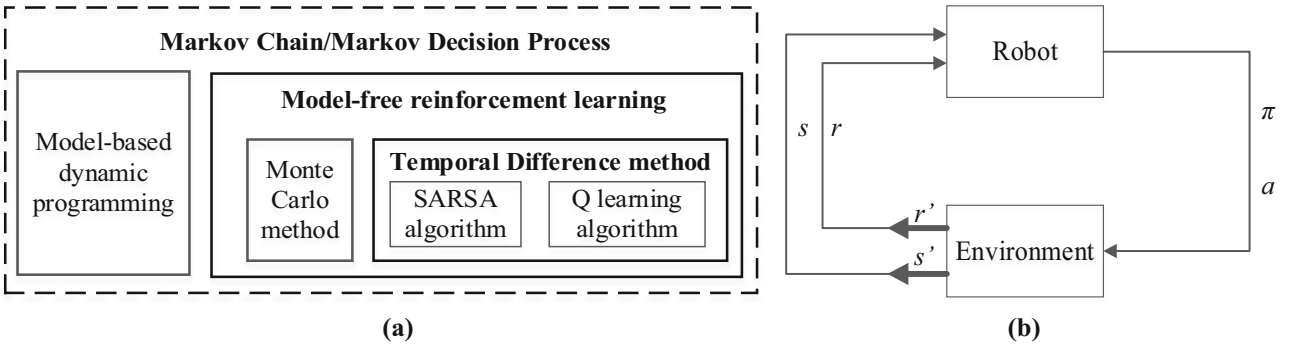


Fig. 16 **a** represents the relationship of basic concepts of RL. **b** represents the principle of MDP

(SARSA). MDP is based on Markov chain (Chan et al., 2012), and it can be divided into two categories: model-based dynamic programming and model-free RL. Mode-free RL can be divided into MC and TD that includes SARSA and Q learning algorithms. Relationship of these concepts is shown in Fig. 16.

Markov chain Variable set $\boldsymbol{X} = \{X_n n > 0\}$ is called Markov chain (Chan et al., 2012) if $\boldsymbol{X}$ meets

$$p(X_{t+1}|X_t, \ldots, X_1) = p(X_{t+1}|X_t) \tag{4}$$

This means the occurrence of event X_{t+1} depends only on event X_t and has no correlation to any earlier events.

Markov decision process MDP (Chan et al., 2012) is a sequential decision process based on Markov Chain. This means the state and action of the next step depend only on the state and action of the current step. MDP is described as a tuple $< S, A, P, R >$. S represents the state and here it refers to the state of robot and obstacles. A represents an action taken by robot. State S transits into another state under a state-transition probability P and a reward R from the environment is obtained. Principle of MDP is shown in Fig. 16. First, the robot in state s interacts with the environment and generate an action based on policy $\pi(\mathrm{s})s \rightarrow a$. Robot then obtains the reward r from the environment, and state transits into the next state s'. The reach of next state s' marks the end of one loop and the start of the next loop.

Model-free RL and model-based dynamic programming Problems in MDP can be solved using *model-based dynamic programming* and *model-free RL* methods. The model-based dynamic programming is used in a known environment, while the model-free RL is utilized to solve problems in an unknown environment.

Temporal difference and Monte Carlo methods The model-free RL includes MC and TD. A sequence of actions is called an *episode*. Given an episode$<S_1, A_1, R_2, S_2, A_2, R_3, \ldots, S_t, A_t, R_{t+1}, \ldots, S_T >$, the state value $V(s)$ in the time step t is defined as the expectation of accumulative rewards G_t by

$$V(s) = \mathbb{E}\Big[G_t = R_{t+1} + \gamma R_{t+1} + \ldots + \gamma^{T-1} R_T | S_t = s\Big] \tag{5}$$

where γ represent a discount factor ($\gamma \epsilon [0, 1]$). MC uses $G_t - V(s)$ to update its state value $V_{MC}(s)$ by

$$V_{MC}(s) \leftarrow V(s) + \alpha(G_t - V(s)) \tag{6}$$

where "$\leftarrow$" represents the update process in which new value will replace previous value. α is a discount factor. TD uses $R_{t+1} + \gamma V(s_{t+1}) - V(s)$ to update its state value $V_{TD}(s)$ by

$$V_{TD}(s) \leftarrow V(s) + \alpha\Big[R_{t+1} + \gamma V(s_{t+1}) - V(s)\Big] \tag{7}$$

where α is a learning rate, $R_{t+1} + \gamma V(s_{t+1})$ is the *TD target* in which the estimated state value $V(s_{t+1})$ is obtained by *bootstrapping* method (Tsitsiklis, 2003). This means MC updates its state value after the termination of an episode, while TD update its state value in every steps. TD method is therefore efficient than MC in state value update.

Q learning

TD includes SARSA (Rummery & Niranjan, 1994) and Q learning ((Smart & Kaelbling, 2002; Sutton & Barto, 1998)). Given an episode$<S_1, A_1, R_2, S_2, A_2, R_3, \ldots, S_t, A_t, R_{t+1}, \ldots, S_T >$, SARSA and Q learning use the ε-*greedy* method (Santos Mignon., 2017) to select an action A_t at time step t. There are two differences between SARSA and Q learning: (1) SARSA uses ε-*greedy* again to select an estimated action value $Q(S_{t+1}, A_{t+1})$ at time step $t + 1$ to update its action value by

$$Q_{SARSA}(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right) \tag{8}$$

while$Q$ learning directly uses maximum estimated action value $\max_Q Q(S_{t+1}, A_{t+1})$ at time step $t + 1$ to update its action value by

$$Q_{QL}(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left(R_{t+1} + \gamma \max_{A_{t+1}} Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right) \quad (9)$$

(2) SARSA adopts selected action A_{t+1} directly to update its next action value, but Q learning algorithm use *ε-greedy* to select a new action to update its next action value.

SARSA uses *ε-greedy* method to sample all potential action values of next step and selects a "safe" action eventually, while Q learning pays attention to the maximum estimated action value of the next step and selects optimal actions eventually. Steps of SARSA is shown in **Algorithm 1** (Sutton & Barto, 1998), while Q learning algorithm as **Algorithm 2** (Sutton & Barto, 1998) and Fig. 17. Implementations of robotic motion planning by Q learning are as (Panov et al., 2018; Qureshi et al., 2018; Smart & Kaelbling, 2002).

Nature deep Q-learning network

DQN (Mnih et al., 2013) is a combination of Q leaning and deep neural network (e.g., CNN). DQN uses CNN to approximate Q values by its weight θ. Hence, Q table in Q learning changes to Q value network that can be converged in a faster speed in complex motion planning. DQN became a research focus when it was invented by Google DeepMind (Mnih et al., 2013, 2015), and performance of DQN approximates or even surpasses the performance of human being in Atari games (e.g., Pac-man and Enduro in Fig. 18) and real-world motion planning tasks (Bae et al., 2019; Isele et al., 2017). DQN utilizes CNN to approximate Q values (Fig. 19) by

$$Q^*(s, a) \approx Q(s, a; \theta) \quad (10)$$

Algorithm 1: SARSA

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Choose a from s using policy derived from Q (e.g., *ε-greedy*)
  Repeat (for each step of episode):
    Take action a, observe r, s'
    Choose a' from s' using policy derived from Q (e.g., *ε-greedy*)
    $Q(s,a) \leftarrow Q(s,a)+\alpha[r+\gamma Q(s',a')-Q(s,a)]$
    $s \leftarrow s'$; $a \leftarrow a'$.
  until s is terminal

Algorithm 2: Q-learning

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    Choose a from s using policy derived from Q (e.g., *ε-greedy*)
    Take action a, observe r, s'
    $Q(s,a) \leftarrow Q(s,a)+\alpha[r+\gamma \max_{a'} Q(s',a')-Q(s,a)]$
    $s \leftarrow s'$;
  until s is terminal

Note that s' and a' denote S_{t+1} and A_{t+1} respectively.

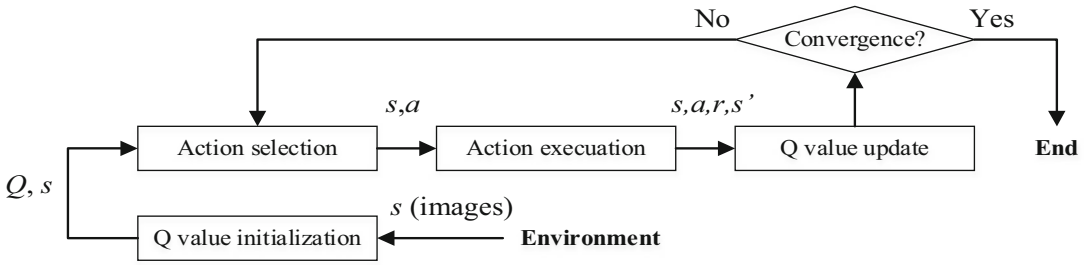


Fig. 17 Steps of Q learning algorithm. Input of Q learning is in the vector format normally. Q value is obtained via Q value table or network as approximator. Extra preprocessing is needed to extract features from image if input is in image format

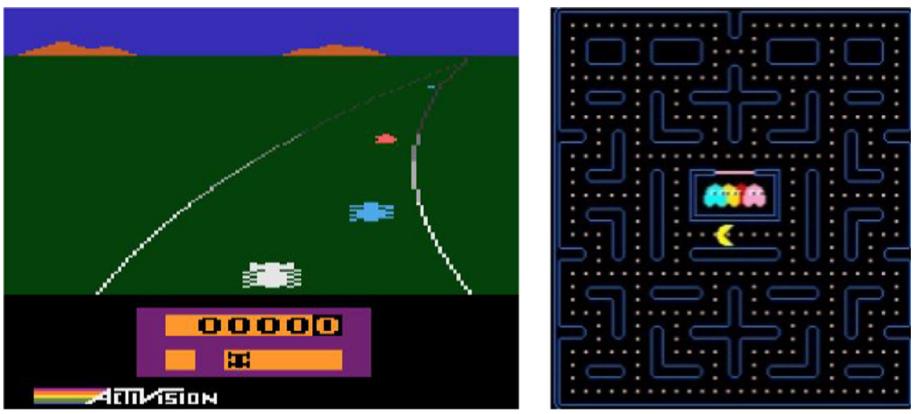


Fig. 18 Two examples of motion planning in early-stage arcade games: Enduro (left) and Pac-man (right)

In contrast with the Q learning, DQN features three components: CNN, *replay buffer* (Schaul et al., 2016) and *targeted network*. CNN extracts feature from images that are inputs. Outputs can be Q value of current state $Q(s,a)$ and Q value of next state $Q(s',a')$, therefore experiences $<s,a,r,s'>$ are obtained and temporarily stored in replay buffer. It is followed by training DQN using experiences in the replay buffer. In this process, a targeted network θ' is leveraged to minimize the loss value by

$$Loss = \left(r + \gamma \max_{a'} Q(s', a'; \theta') - Q(s, a; \theta)\right)^2 \tag{11}$$

Loss value measures the distance between expected value and real value. In DQN, expected value is $(r + \gamma \max Q(s',a';\theta'))$ that is similar to labels in supervised learning, while $Q(s,a;\theta)$ is the observed real value. weights of targeted network and Q value network share a same weight θ. The difference is that weight of Q value network θ is updated

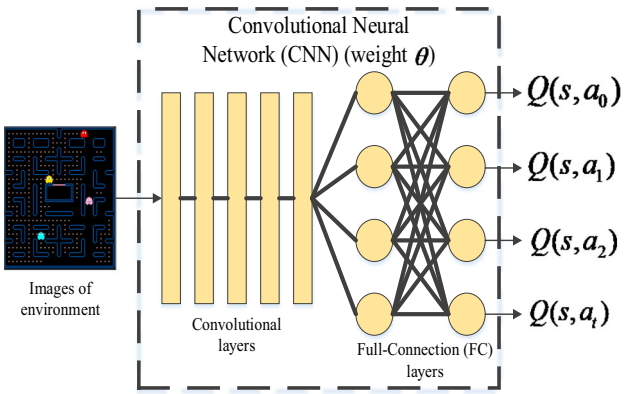


Fig. 19 $Q(s,a_0)$, $Q(s,a_1)$, $Q(s,a_2)$ and $Q(s,a_t)$ denote Q values of all potential actions

in each step, while weight of targeted network θ' is updated in a long period of time. Hence, θ is updated frequently while θ' is more stable. It is necessary to keep targeted network stable, otherwise Q value network will be hard to converge. Detailed steps of DQN are shown as **Algorithm 3** (Mnih et al., 2013) and Fig. 20.

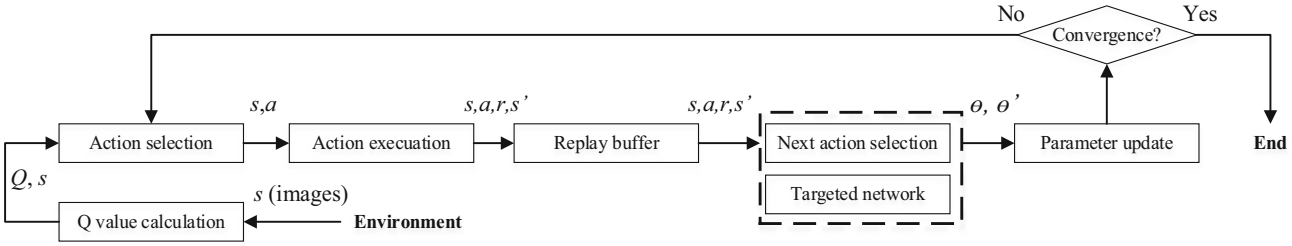


Fig. 20 Steps of DQN algorithm

Algorithm 3: Deep Q-learning with experience replay

Initialize replay memory $\mathcal{D}$ to capacity $\mathcal{N}$
Initialize action-value function Q with random weights
for episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\emptyset_1 = \emptyset(s_1)$
  for t = 1, T **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\emptyset(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t,\ a_t,\ x_{t+1}$ and preprocess $\emptyset_{t+1} = \emptyset(s_{t+1})$
    Store transition $(\emptyset_t, a_t, r_t, \emptyset_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\emptyset_j, a_j, r_j, \emptyset_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for ternimal } \emptyset_{j+1} \\ r_j + \gamma max_{a'} Q(\emptyset_{j+1}, a'; \theta) & \text{for nonternimal } \emptyset_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\emptyset_j, a_j; \theta))^2$
  end for
end for

Double deep Q-learning network

Noise in DQN leads to *bias* and false selection of next action a' follows, therefore leading to over-estimation of next action value $Q(s', a'; \theta')$. To reduce the over-estimation caused by noise, researchers invented the *double DQN* (Hasselt et al., 2016) in which another independent targeted network with weight θ^- is introduced to evaluate the selected action a'. Hence, equation of targeted network therefore changes from $y^{DQN} = r + \gamma \max Q(s', a'; \theta')$ to

$$y^{doubleDQN} = r + \gamma\, Q\left(s', \arg max_{a'} Q(s', a'; \theta'); \theta^-\right) \quad (12)$$

Steps of double DQN are the same with DQN. Examples of application are (Chao et al., 2020; Lei et al., 2018; Sui et al., 2018) in which double DQN is used in games and physical robots based on ROS.

Dueling deep Q-learning network

The state value $V^\pi(s)$ measures "how good the robot is" in the state s where π denotes policy $\pi : s \rightarrow a$, while the action value $Q^\pi(s, a)$ denotes "how good the robot is" **after** robot takes action a in state s using policy π. *Advantage value* (A value) denotes the difference of $Q^\pi(s, a)$ and $V^\pi(s)$ by

$$A(s, a) = Q(s, a) - V(s, a) \quad (13)$$

therefore A value measures "how good the action a is" in state s **if** robot takes the action a. In neural network case (Fig. 21), weights α, β, θ are added, therefore

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + A(s, a; \theta, \alpha) \quad (14)$$

where θ is the weight of neural network and it is the shared weight of Q, V and A values. Here α denotes the weight of A value, and β the weight of V value. $V(s; \theta, \beta)$ is a scalar, and $A(s, a; \theta, \alpha)$ is a vector. There are however too many V-A value pairs if Q value is simply divided into two components

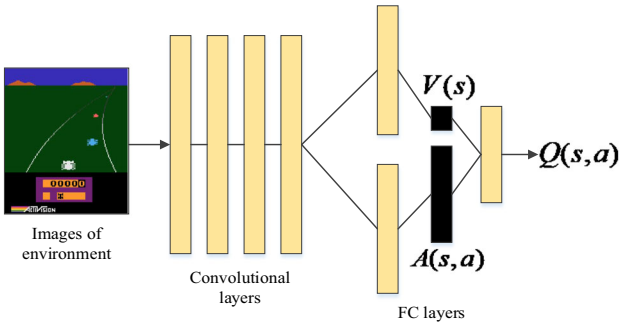


Fig. 21 The architecture of dueling DQN, in which Q value $Q(s,a)$ is decoupled into two parts, including V value $V(s)$ and A value $A(s,a)$

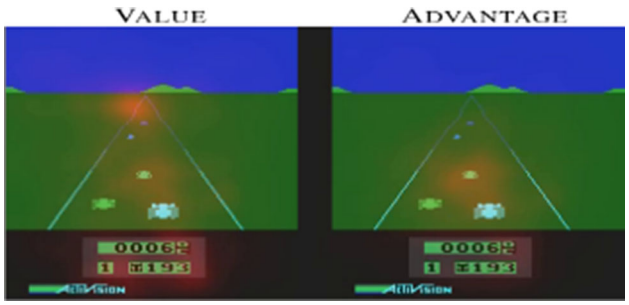


Fig. 22 $Q(s,a)$ and $A(s,a)$ saliency maps (red-tinted overlay) on the Atari game (Enduro). $Q(s,a)$ learns to pay attention to the road, but pay less attention to obstacles in the front. $A(s,a)$ learns to pay much attention to dynamic obstacles in the front (Wang et al., 2015)

without constraints, and only one V-A pairs are qualified. Thus, it is necessary to constrain the V value or A value to obtain a fixed V-A pair. According to relationship of $Q^{\pi}(s,a)$ and $V^{\pi}(s)$ where $V^{\pi}(s) = \mathbb{E}_{a\sim\pi(s)}[Q^{\pi}(s,a)]$, the expectation of A value is

$$\mathbb{E}_{a\sim\pi(s)}[A(s,a)] = 0 \tag{15}$$

Equation 15 can be used as a rule to constrain A value for obtaining a stable V-A pair. Expectation of A value is obtained by using $A(s_t,a_t)$ to subtract mean A value that is obtained from all possible actions, therefore

$$\mathbb{E}[A(s_t,a_t)] = A(s_t,a_t) - \frac{1}{|\mathcal{A}|}\sum_{a_t^-\epsilon\mathcal{A}} A\left(s_t,a_t^-\right) \tag{16}$$

where $\mathcal{A}$ represents *action space* in time step t, $|\mathcal{A}|$ the number of actions, and a_t^- one of possible actions in $\mathcal{A}$ at time step t. Expectation of A value keeps zero for $t\epsilon[0,T]$, although the fluctuation of $A(s_t,a_t)$ in different action choices. Researchers use the expectation of A value to replace the current A value by

$$\begin{aligned} &Q(s,a;\theta,\alpha,\beta) \\ &\quad = V(s;\theta,\beta) \\ &\qquad + \left\{A(s,a;\theta,\alpha) - \frac{1}{|\mathcal{A}|}\sum_{a_t^-\epsilon\mathcal{A}} A\left(s_t,a_t^-;\theta,\alpha\right)\right\} \end{aligned} \tag{17}$$

Thus, a stable V-A pair is obtained although original semantic definition of A value (Eq. 13) is changed (Wang et al., 2015). In other words: (1) advantage constraint $\left\{A(s,a;\theta,\alpha) - \frac{1}{|\mathcal{A}|}\sum_{a_t^-\epsilon\mathcal{A}} A\left(s_t,a_t^-;\theta,\alpha\right)\right\} = 0$ is used to constrain the update of A value network α; (2) Q value network θ is therefore obtained by $Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta)$. Q value network θ is updated according to V value that is more accurate and it is easy to obtain via accumulative rewards defined by Eq. 5. Hence, a better estimation of action value is obtained by performing a better Q value network θ.

DQN obtained action value $Q(s,a)$ directly by using network to approximate action value. This process introduces over-estimation of action value. Dueling DQN obtains better action value $Q(s,a)$ by constraining advantage value $A(s,a)$. Finally, three weights (θ,α,β) are obtained after training, and Q value network θ is with less bias but A value is better than action value to represent "how good the action is" (Fig. 22).

Further optimizations are distributional DQN (Bellemare et al., 2017), noise network (Fortunato et al., 2017), dueling double DQN[2] and rainbow model (Hessel et al., 2017). Distributional DQN is like the dueling DQN, as noise is reduced by optimizing the architecture of DQN. Noise network is about improving the ability in exploration by a more exquisite and smooth approach. Dueling double DQN and rainbow model are hybrid algorithms. Rainbow model fuses several suitable components: double networks, replay buffer, dueling network, multi-step learning, distributional network, and noise network.

Policy gradient RL

Here the principles of policy gradient method and actor-critic algorithm are given firstly. It is followed by recalling the principles of their optimized variants: (1) A3C and A2C; (2) DPG and DDPG; (3) TROP and PPO.

Optimal value RL uses neural network to approximate optimal values to indirectly select actions. This process is simplified as $a \leftarrow argmax_a R(s,a) + Q(s,a;\theta)$. Noise leads to over-estimation of $Q(s,a;\theta)$, therefore the selected actions are suboptimal, and network θ is hard to converge. Policy gradient algorithm uses neural network θ as policy

[2] A. Suran. Dueling Double Deep Q Learning using Tensorflow 2.x. Web. Jul 10, **2020**. https://towardsdatascience.com/dueling-double-deep-q-learning-using-tensorflow-2-x-7bbbcec06a2a.

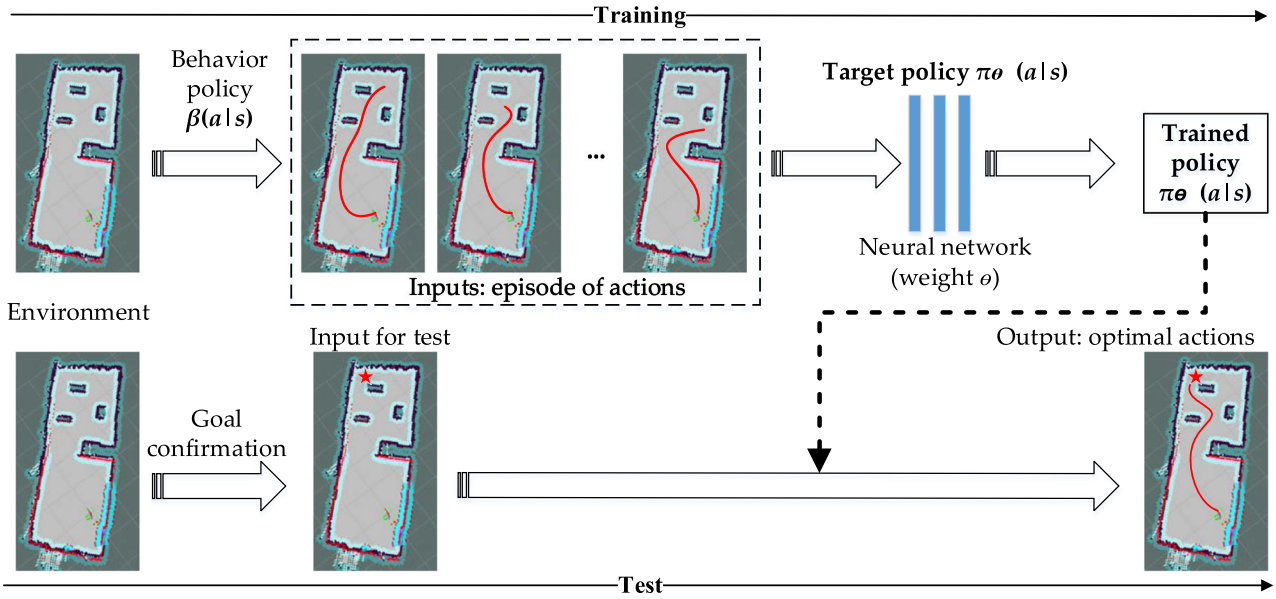


Fig. 23 Training and test steps of policy gradient algorithms. In the training, time-sequential actions are generated by the behavior policy. Note that policy is divided to *behavior policy* and *target policy*. Behavior policy is about selecting actions for training and behavior policy will not be updated, while target policy is also used to select actions but it will be updated in training. Policy refers to target policy normally. Robots learn trajectories via target policy (neural network as approximator) and trained policy is obtained. In the test, optimal time-sequential actions are generated directly by trained policy $\pi_\theta : s \rightarrow a$ until destination is reached

$\pi_\theta : s \rightarrow a$ to directly select actions to avoid this problem. Brief steps of policy gradient algorithm are shown in Fig. 23.

Policy gradient method

Policy is a *probability distribution* $P\{a|s,\theta\} = \pi_\theta(a|s) = \pi(a|s;\theta)$ that is used to select action a in state s, where weight θ is the parameter matrix that is used as an approximation of policy $\pi(a|s)$. *Policy gradient method* (PG) (Sutton et al., 1999) seeks an optimal policy and uses it to find optimal actions. how to find this optimal policy? Given a episode $\tau = (s_1,a_1,\ldots,s_T,a_T)$, the probability to output actions in τ is $\pi_\theta(\tau) = p(s_1)\prod_{t=2}^{T}\pi_\theta(a_t|s_t)p(s_t|s_{t-1},a_{t-1})$. The aim of the PG is to find optimal parameter $\theta^* = \arg max_\theta \mathbb{E}_{\tau\sim\pi_\theta(\tau)}[R(\tau)]$ where episode reward $R(\tau) = \sum_{t=1}^{T} r(s_t,a_t))$ is the accumulative rewards in episode τ. Objective of PG is defined as the expectation of rewards in episode τ by

$$J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta(\tau)}[R(\tau)] = \int \pi_\theta(\tau)R(\tau)d\tau \tag{18}$$

To find higher expectation of rewards, *gradient operation* is used on objective to find the increment of network that may lead to a better policy. Increment of network is the gradient value of objective, and that is

$$\begin{aligned}\nabla_\theta J(\theta) &= \int \nabla_\theta \pi_\theta(\tau) R(\tau) d\tau \\ &= \int \pi_\theta(\tau) \nabla_\theta \log \pi_\theta(\tau) R(\tau) d\tau \\ &= \mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[\nabla_\theta \log \pi_\theta(\tau) R(\tau)\right]\end{aligned} \tag{19}$$

An example of PG is *Monte-carlo reinforce* (Williams, 1992). Data τ for training are generated from simulation by *stochastic policy*. Previous objective and its gradient (Eq. 18–19) are replaced by

$$J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} r\left(s_t^i, a_t^i\right) \tag{20}$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left[\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta\left(a_t^i, s_t^i\right)\right]\left[\sum_{t=1}^{T} r\left(s_t^i, a_t^i\right)\right] \tag{21}$$

where N is the number of episodes, T the length of trajectory. A target policy π_θ is used to generate episodes for training. For example, *Gaussian distribution function* is used as behavior policy to select actions by $a \sim \mathcal{N}\left(\mu(s), \sigma^2\right)$. Network $f(s;\theta)$ is then used to approximate expectation of Gaussian distribution by $\mu(s) = f(s;\theta)$. It means $a \sim \mathcal{N}\left(mean = f\left[s;\theta = \{w_i, b_i\}_i^L\right], stdev = \sigma^2\right)$ and $\mu(s;\theta) =$

$[mean, stdev]$ where w and b represent weight and bias of network, L the number of w-b pairs. Its objective is defined as $J(\theta) = f(s_t; w) - a^2_{t\Sigma}$, therefore the objective gradient is

$$\nabla_\theta J(\theta) = -\frac{1}{2}\Sigma^{-1}(f(s_t) - a_t)\frac{df}{d\theta} \tag{22}$$

where $\frac{df}{d\theta}$ is obtained by backward-propagation. According to Eq. 21–22, its objective gradient is

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left[\sum_{t=1}^{T} -\frac{1}{2}\Sigma^{-1}\left(f\left(s_t^i\right) - a_t^i\right)\frac{df}{d\theta}\right] \times \left[\sum_{t=1}^{T} r\left(s_t^i, a_t^i\right)\right] \tag{23}$$

Once objective gradient is obtained, network is updated by *gradient ascent method*. That is

$$\theta \leftarrow \theta + \nabla_\theta J(\theta) \tag{24}$$

Actor-critic algorithm

The update of policy in PG is based on expectation of accumulative rewards in episode τ $\mathbb{E}_{\tau\sim\pi_\theta(\tau)}[R(\tau)]$. This leads to high *variance* that causes low speed in network convergence, but convergence stability is improved. *Actor-critic algorithm* (AC) (Cormen et al., 2009; Kim et al., 2019; Konda & Tsitsiklis, 2001) reduces the variance by one-step reward in TD-error e for network update. TD-error is defined by

$$e = r_t + V(s_{t+1}) - V(s_t) \tag{25}$$

To enhance convergence speed, AC uses actor-critic architecture that includes *actor network (policy network)* and *critic network*. Critic network is used in TD-error therefore TD-error changes into

$$e = r_t + V(s_{t+1}; w) - V(s_t; w) \tag{26}$$

Objective of critic network is defined by

$$J(w) = e^2 \tag{27}$$

Objective gradient is therefore obtained by minimizing the mean-square error

$$\nabla_w J(w) = \nabla_w e^2 \tag{28}$$

Critic network is updated by *gradient ascent method* (Zhang, 2019). That is

$$w \leftarrow w + \beta\nabla_w J(w) \tag{29}$$

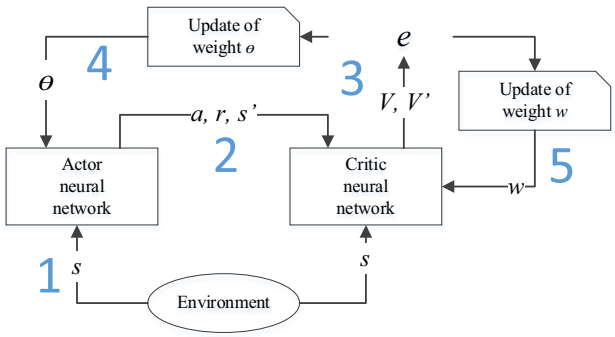


Fig. 24 Training steps of AC

where β represents learning rate. Objective of policy network is defined by

$$J(\theta) = \pi_\theta(a_t|s_t) \cdot e \tag{30}$$

Hence, objective gradient of policy network is obtained by

$$\nabla_\theta J(\theta) = \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot e \tag{31}$$

and policy network is updated by

$$\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta) \tag{32}$$

where α is a learning rate of actor network. Detailed steps of the AC are as Fig. 24: (1) action a_t at time step t is selected by policy network θ; (2) selected action is executed and reward is obtained. State transits into the next state s_t; (3) state value is obtained by critic network and TD error is obtained; (4) policy network is updated by minimizing objective of critic network; (5) critic network is updated according to objective gradient of critic network. This process repeats until the convergence of policy and critic networks.

A3C and A2C

A3C In contrast to AC, the A3C (Everett et al., 2018) has three features (1) multi-thread computing; (2) multi-step rewards; (3) policy entropy. Multi-thread computing means multiple interactions with the environment to collect data and update networks. Multi-step rewards are used in critic network, therefore the TD-error$e$ of A3C is obtained by

$$e = \sum_{i=t}^{T} \gamma^{i-t} r_i + V(s_{t+n}) - V(s_t) \tag{33}$$

Hence, the speed of convergence is improved. Here γ is a discount factor, and n is the number of steps. Data collection by policy $\pi(s_t; \theta)$ will cause *over-concentration*, because

initial policy is with poor performance therefore actions are selected from small area of workspace. This causes poor quality of the input, therefore convergence speed of network is poor. Policy entropy increases the ability of policy in action exploitation to reduce over-concentration. Objective gradient of A3C therefore changes to

$$\nabla_\theta J(\theta)_{A3C} = \nabla_\theta J(\theta)_{AC} + \beta \nabla_\theta H(\pi(s_t;\theta)) \tag{34}$$

where β is a discount factor and $H(\pi(s_t;\theta)$ is the policy entropy.

A2C A2C (Everett et al., 2018) is the alternative of A3C algorithm. Each thread in A3C algorithm can be utilized to collect data, train critic and policy networks, and send updated weights to global model. Each thread in A2C however can only be used to collect data. Weights in A2C are updated synchronously compared with the asynchronous update of A3C, and experiments demonstrate that synchronous update of weights is better than asynchronous way in weights update ((Babaeizadeh et al., 2016; Mnih et al., 2016)). Their mechanisms in weight update are shown in Fig. 25.

DPG and DDPG

Here some prerequisites are recalled *on-policy algorithm*, *off-policy algorithm*, *important sampling ratio, stochastic policy gradient algorithm*, and then the principles of DPG and DDPG are given.

Prerequisites In data generation and training processes, if behavior policy and target policy of an algorithm are the same policy π_θ, this algorithm is called ***on-policy algorithm***. On-policy algorithm however may lead to low-quality data in data generation and a slow speed in network convergence. This problem can be solved by using one policy (behavior policy) β_θ for data generation and another policy (target policy) π_θ for learning and making decision. Algorithms using different policies on data generation and learning are therefore called ***off-policy algorithms***. Although policies in off-policy algorithm are different, their relationship can still be measured by *transition probability* $\rho^\beta(s)$ that is the ***importance-sampling ratio*** and defined by

$$\rho^\beta(s) = \frac{\pi_\theta(a|s)}{\beta_\theta(a|s)} = \frac{\prod_{k=t}^{T} \pi(a_k|s_k;\theta)}{\prod_{k=t}^{T} \beta(a_k|s_k;\theta)} \tag{35}$$

Importance-sampling ratio measures the similarity of two policies. These policies must be with large similarity in the definition of important sampling. Particularly, behavior policy β_θ is the same as policy π_θ in on-policy algorithms. This means $\pi_\theta = \beta_\theta$ and $\rho^\beta(s) = \rho^\pi(s) = 1$.

In on-policy policy gradient algorithm (e.g., PG), its objective is defined as

$$\begin{aligned} J(\theta) &= \mathbb{E}_{\tau\sim\pi_\theta(\tau)}[R(\tau)] = \int_{\tau\sim\pi_\theta(\tau)} \pi_\theta(\tau) R(\tau) d\tau \\ &= \int_{s\sim\mathcal{S}} \rho^\pi(s) \int_{a\sim\mathcal{A}} \pi_\theta(a|s) R(s,a) da ds \\ &= \mathbb{E}_{s\sim\rho^\pi, a\sim\pi_\theta}[R(s,a)] \end{aligned} \tag{36}$$

where ρ^π is the distribution of state transition. The objective gradient of PG $\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[\nabla_\theta \log \pi_\theta(\tau) R(\tau)\right]$ includes a vector $C = \nabla_\theta \log \pi_\theta(\tau)$ and a scalar $R = R(\tau)$. Vector C is the *trend* of policy update, while scalar R is the *range* of this trend. Hence, the scalar R acts as a critic that decides how policy is updated. Action value $Q^\pi(s,a)$ is defined as the expectation of discounted rewards by

$$Q^\pi(s,a) = \mathbb{E}\left[r_1^\gamma = \sum_{k=t}^{\infty} \gamma^{k-t} r(s_k,a_k) | S_1 = s, A_1 = a; \pi\right] \tag{37}$$

$Q^\pi(s,a)$ is an alternative of scalar R, and it is better than R as the critic. Hence, objective gradient of PG changes to

$$\begin{aligned} \nabla_\theta J(\theta) &= \nabla_\theta \int_{s\sim\mathcal{S}} \rho^\pi(s) \int_{a\sim\mathcal{A}} \pi_\theta(a|s) Q^\pi(s,a) da ds \\ &= \mathbb{E}_{s\sim\rho^\pi, a\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s,a)\right] \end{aligned} \tag{38}$$

and policy is updated using objective gradient with action value $Q^\pi(s,a)$. Hence, algorithms are called ***stochastic policy gradient algorithm*** if action value $Q^\pi(s,a)$ is used as critic.

DPG DPG are algorithms in which a deterministic policy $\mu_\theta(s)$ is trained to select actions, instead of policy $\pi_\theta(a|s)$ in AC. A policy is *deterministic policy* $\mu_\theta(s)$ if it directly maps the state to the action $\leftarrow \mu_\theta(s)$, while stochastic policy $\pi_\theta(a|s)$ maps state and action to a probability $P(a|s)$ (Silver et al., 2014). The update of deterministic policy is defined as

$$\mu^{k+1}(s) = \arg max_a Q^{\mu^k}(s,a) \tag{39}$$

If network θ is used as approximator of deterministic policy, update of network changes to

$$\begin{aligned} \theta^{k+1} &= \theta^k + \alpha \mathbb{E}_{s\sim\rho^{\mu^k}}\left[\nabla_\theta Q^{\mu^k}(s, \mu_\theta(s))\right] \\ &= \theta^k + \alpha \mathbb{E}_{s\sim\rho^{\mu^k}}\left[\nabla_\theta \mu_\theta(s) \nabla_a Q^{\mu^k}(s,a)|_{a=\mu_\theta(s)}\right] \end{aligned} \tag{40}$$

There are small changes in state distribution ρ^μ of deterministic policy during the update of network θ, but this change will not impact the update of network. Hence, network of deterministic policy is updated by

$$\theta \leftarrow \theta + \alpha \mathbb{E}_{s\sim\rho^\mu}\left[\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)}\right] \tag{41}$$

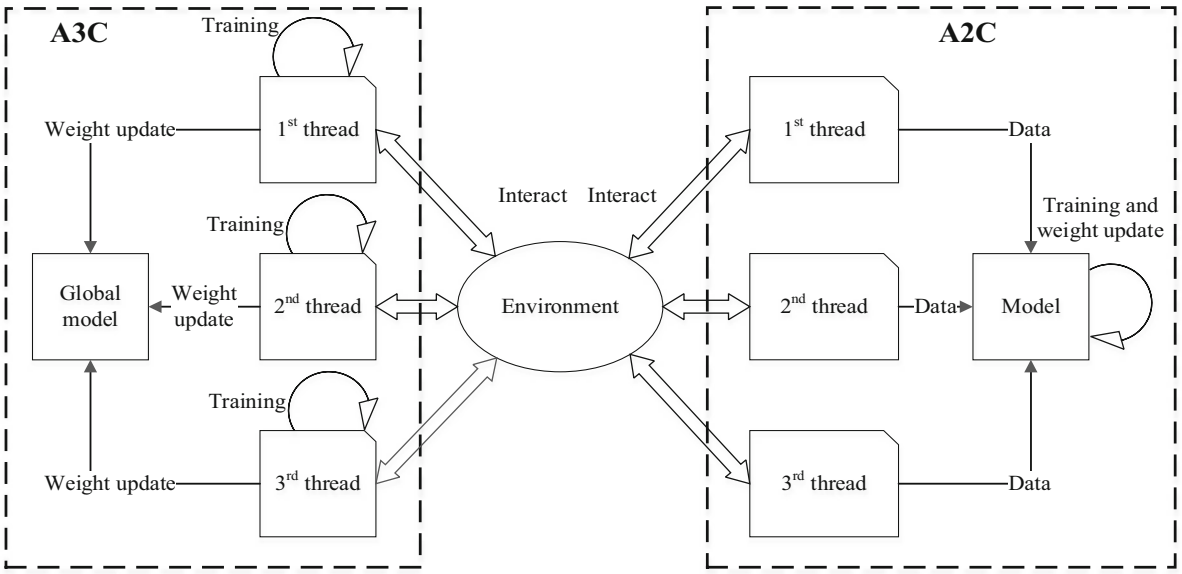


Fig. 25 The weight update processes of the A3C and A2C

because

$$\begin{aligned}\nabla_\theta J(\mu_\theta) &= \nabla_\theta \int_{\mathcal{S}} \rho^\mu(s) R(s, \mu_\theta(s))\, ds \\ &= \nabla_\theta \mathbb{E}_{s\sim\rho^\mu}[R(s, \mu_\theta(s))] \\ &= \int_{\mathcal{S}} \rho^\mu(s) \nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)} \\ &= \mathbb{E}_{s\sim\rho^\mu}\left[\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)}\right] \end{aligned} \tag{42}$$

Once $Q^\mu(s,a)$ is obtained, θ can be updated after obtaining objective gradient.

How to find $Q^\mu(s,a)$? Note that discounted reward $Q^\mu(s,a)$ is a critic in stochastic policy gradient mentioned before. If network w is used as approximator, $Q^\mu(s,a)$ is obtained by

$$Q^\mu(s,a) \approx Q^w(s,a) \tag{43}$$

stochastic policy gradient algorithm includes two networks, in which w is the *critic* that approximates Q value and θ is used as *actor* to select actions in test (actions are selected by behavior policy β in training). Stochastic policy gradient in this case is called *off-policy deterministic actor-critic* (OPDAC) or *OPDAC-Q*. Objective gradient of OPDAC therefore changes from the Eq. 42 to

$$\nabla_\theta J_\beta(\mu_\theta) \approx \mathbb{E}_{s\sim\rho^\beta}\left[\nabla_\theta \mu_\theta(s) \nabla_a Q^w(s,a)|_{a=\mu_\theta(s)}\right] \tag{44}$$

where β represents the behavior policy. Two networks are updated by

$$\delta_t = r_t + \gamma Q^w(s_{t+1}, \mu_\theta(s_{t+1})) - Q^w(s_t, a_t) \tag{45}$$

$$w_t \leftarrow w_t + \alpha_w \delta_t \nabla_w Q^w(s_t, a_t) \tag{46}$$

$$\theta_t \leftarrow \theta_t + \alpha_\theta \nabla_\theta \mu_\theta(s) \nabla_a Q^w(s,a)|_{a=\mu_\theta(s)} \tag{47}$$

However, no constrains is used on network w in the approximation of Q value and this will lead to a large bias.

How to obtain a $Q^w(s,a)$ without bias? *Compatible function approximation* (CFA) can eliminate the bias by adding two constraints on w (proof is given in Silver et al. (2014)): (1) $\nabla_a Q^w(s,a)|_{a=\mu_\theta(s)} = \nabla_\theta \mu_\theta(s)^\intercal w$; (2) $MSE(\theta, w) = \mathbb{E}[\epsilon(s;\theta,w)]\left[\epsilon(s;\theta,w)^\intercal \epsilon(s;\theta,w)\right] \rightarrow 0$ where $\in(s;\theta,w) = \nabla_a Q^w(s,a)|_{a=\mu_\theta(s)} - \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)}$. In other words, $Q^w(s,a)$ should meet

$$Q^w(s,a) = (a - \mu_\theta(s))^\intercal \nabla_\theta \mu_\theta(s)^\intercal w + V^v(s) \tag{48}$$

where state value $V^v(s)$ may be any *differentiable baseline function* (Silver et al., 2014). Here v and $\varnothing$ are feature and parameter of state value ($V^v(s) = v^\intercal \varnothing(s)$). Parameter $\varnothing$ is also the feature of advantage function ($A^w(s,a) = \varnothing(s,a)^\intercal w$), and $\varnothing(s,a)$ is defined as $\varnothing(s,a) \overset{\text{def}}{=} \nabla_\theta \mu_\theta(s)(a - \mu_\theta(s))$. Hence, a low-bias $Q^w(s,a)$ is obtained using OPDAC-Q and CFA. This new algorithm with less bias is called *Compatible OPDAC-Q* (COPDAC-Q) (Silver et al., 2014), in which weights are updated as Eq. 49–51

$$v_t \leftarrow v_t + a_v \delta_t \varnothing(s_t) \tag{49}$$

$$w_t \leftarrow w_t + \alpha_w \delta_t \varnothing(s_t, a_t) = w_t + \alpha_w \delta_t \nabla_w A^w(s_t, a_t) \tag{50}$$

$$\theta_t \leftarrow \theta_t + \alpha_\theta \nabla_\theta \mu_\theta(s_t)(\nabla_\theta \mu_\theta(s_t)^\intercal w_t) \tag{51}$$

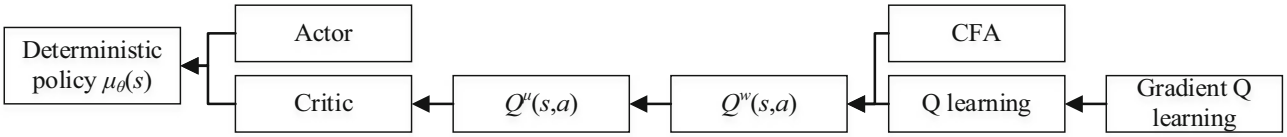


Fig. 26 Brief steps of DPG algorithm

where δ_t is the same as the Eq. 45. Here a_v, α_w and α_θ are learning rates. Note that *linear function approximation method* (Silver et al., 2014) is used to obtain advantage function $A^w(s,a)$ that is used to replace the value function $Q^w(s,a)$ because $A^w(s,a)$ is efficient than $Q^w(s,a)$ in weight update. Linear function approximation however may lead to divergence of $Q^w(s,a)$ in critic δ. Critic δ can be replaced by the *gradient Q-learning critic* (Sutton et al., 2009) to reduce the divergence. Algorithm that combines COPDAC-Q and gradient Q-learning critic is called *COPDAC Gradient Q-learning* (COPDAC-GQ). Details of gradient Q-learning critic and COPDAC-GQ algorithm can be found in ((Silver et al., 2014; Sutton et al., 2009)).

By analytical illustration above, 2 examples (COPDAC-Q and COPDAC-GQ) of DPG algorithm are obtained. In short, key points of DPG are to: (1) find a no-biased $Q^w(s,a)$ as critic; (2) train a deterministic policy $\mu_\theta(s)$ to select actions. networks of DPG are updated as AC via the policy ascent approach. Brief steps of DPG are shown in Fig. 26.

DDPG (Lillicrap et al., 2019) is the combination of replay buffer, deterministic policy $\mu(s)$ and actor-critic architecture. θ^Q is used as critic network to approximate action value $Q\left(s_i,a_i;\theta^Q\right)$. θ^μ is used as policy network to approximate deterministic policy $\mu(s;\theta^\mu)$. TD target y of DDPG is defined by

$$y_i = r_i + \gamma Q'\left(s_{i+1}, \mu'\left(s_{i+1};\theta^{\mu'}\right);\theta^{Q'}\right) \tag{52}$$

where $\theta^{Q'}$ and $\theta^{\mu'}$ are copies of θ^Q and θ^μ as target networks that are updated with low frequency. The objective of critic network is defined by

$$J\left(\theta^Q\right) = y_i - Q\left(s_i,a_i;\theta^Q\right) \tag{53}$$

Critic network θ^Q is updated by minimizing the loss value (MSE loss)

$$Loss = \frac{1}{N}\sum_i J\left(\theta^Q\right)^2 \tag{54}$$

where N is the number of tuples $<s,a,r,s'>$ sampled from replay buffer. Target function of policy network is defined by

$$J\left(\theta^\mu\right) = \frac{1}{N}\sum_i Q\left(s_i,a_i;\theta^\mu\right) \tag{55}$$

and objective gradient is obtained by

$$\nabla_{\theta^\mu} J\left(\theta^\mu\right) \cong \frac{1}{N}\sum_i \nabla_{\theta^\mu}\mu\left(s_i;\theta^\mu\right)\nabla_a Q\left(s_i,a;\theta^Q\right)|_{a=\mu(s_i)} \tag{56}$$

Hence, policy network θ^μ is updated according to gradient ascent method by

$$\theta^\mu \leftarrow \theta^\mu + \alpha\nabla_{\theta^\mu} J\left(\theta^\mu\right) \tag{57}$$

where α is a learning rate. New target networks

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'} \tag{58}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'} \tag{59}$$

where τ is a learning rate, are obtained by "soft" update method that improves the stability of network convergence. Detailed steps of DDPG are shown in **Algorithm 4** (Lillicrap et al., 2019) and Fig. 27. Examples can be found in ((Jorgensen & Tamar, 2019; Munos et al., 2016)) in which DDPG is used in robotic arms.

Algorithm 4: DDPG

Randomly initialize critic network $Q(s,a;\theta^Q)$ and actor $\mu(s;\theta^\mu)$ with weight θ^Q and θ^μ
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer $\mathcal{R}$
for episode =1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t=1, T **do**
    Selection action $a_t = \mu(s_t;\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{R}$
    Sample a random minibatch of $\mathcal{N}$ transitions (s_i, a_i, r_i, s_{i+1}) from $\mathcal{R}$
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1};\theta^{\mu'});\theta^{Q'})$
    Update critic by minimization the loss: $Loss = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i;\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J(\theta^\mu) \approx \frac{1}{N}\sum_i \nabla_{\theta^\mu}\mu(s_i;\theta^\mu)\nabla_a Q(s_i, a;\theta^Q)|_{a=\mu(s_i)}$$

    Update the target network:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

TRPO and PPO

PPO (Long et al., 2018; Schulman et al., 2017b) is the optimized version of TRPO (Schulman et al., 2017a). Hence, here the principle of TRPO is given before recalling that of PPO.

TRPO Previous policy gradient algorithms update their policies by $\theta \leftarrow \theta + \nabla_\theta J(\theta)$. However, new policy is improved unstably with fluctuation. The goal of TRPO is to improve its policy monotonously, therefore stability of convergence is improved by finding a new policy with the objective that is defined by

$$J(\theta) = L_{\theta_{old}}(\theta), s.t. D_{KL}^{max}(\theta_{old}, \theta) \leq \delta \tag{60}$$

where $L_{\theta_{old}}(\theta)$ is the approximation of new policy's expectation, $D_{KL}^{max}(\theta_{old}, \theta)$ the KL divergence between old policy θ_{old} and new policy θ, and δ a *trust region constraint* of KL divergence. The objective gradient $\nabla_\theta J(\theta)$ is obtained by maximizing the objective $J(\theta)$.

$\eta(\theta)$ and $\eta(\theta_{old})$ denote expectations of new and old policies, respectively. Their relationship is defined by $\eta(\theta) = \eta(\theta_{old}) + \mathbb{E}_{s_0,a_0,s_1,a_1\ldots}\left[\sum_{t=0}^{\infty}\gamma^t A_{\theta_{old}}(s_t, a_t)\right]$ where γ is a discount factor, and $A_{\theta_{old}}(s_t, a_t)$ is the advantage value that is defined by $A_\theta(s,a) = Q_\theta(s,a) - V_\theta(s)$. Thus, $\eta(\theta) = \eta(\theta_{old}) + \sum_s \rho_\theta(s)\sum_a \theta(a|s)A_{\theta_{old}}(s,a)$ where $\rho_\theta(s)$ is the probability distribution of new policy, but $\rho_\theta(s)$ is unknown therefore it is impossible to obtain new policy $\eta(\theta)$. Approximation of new policy's expectation $L_{\theta_{old}}(\theta)$ is defined by

$$\begin{aligned} L_{\theta_{old}}(\theta) &= \eta(\theta_{old}) + \sum_s \rho_{\theta_{old}}(s)\sum_a \theta(a|s)A_{\theta_{old}}(s,a) \\ &= \eta(\theta_{old}) + \sum_s \rho_{\theta_{old}}(s)\sum_a \frac{\theta(a|s)}{\theta_{old}(a|s)} \\ &\quad \cdot \theta_{old}(a|s)\cdot A_{\theta_{old}}(s,a) \\ &= \eta(\theta_{old}) + \mathbb{E}\left[\frac{\theta(a|s)}{\theta_{old}(a|s)}A_{\theta_{old}}(s,a)\right] \end{aligned} \tag{61}$$

where $\rho_{\theta_{old}}(s)$ is known. The relationship of $L_{\theta_{old}}(\theta)$ and $\eta(\theta)$ (Kakade & Langford, 2002) is proved to be

$$\eta(\theta) \geq L_{\theta_{old}}(\theta) - C\cdot D_{KL}^{max}(\theta_{old}, \theta) \tag{62}$$

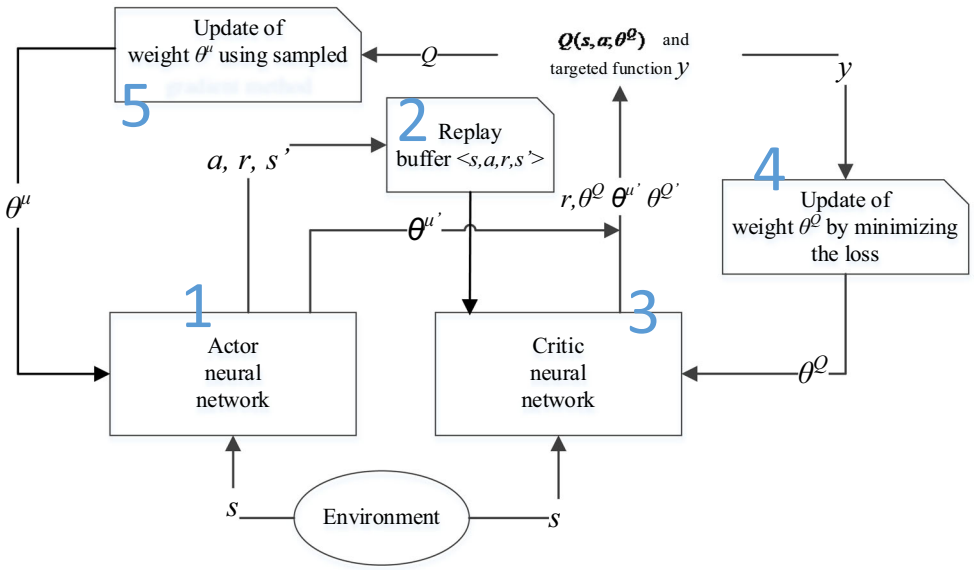


Fig. 27 Steps of DDPG. DDPG combines the replay buffer, actor-critic architecture, and deterministic policy. First, action is selected by policy network and reward is obtained. State transits to next state. Second, experience tuple is saved in replay buffer. Third, experiences are sampled from replay buffer for training. Fourth, critic network is updated. Finally, policy network is updated

where penalty coefficient $C = \frac{2\epsilon\gamma}{(1-\gamma)^2}$, $\gamma\epsilon[0,1]$ and ϵ the maximum advantage. Hence, it is possible to obtain $\eta(\theta)$ by $maximize_{\theta}\left[L_{\theta_{old}}(\theta) - C \cdot D_{KL}^{max}(\theta_{old},\theta)\right]$ or

$$maximize_{\theta}\mathbb{E}\left[\frac{\theta(a|s)}{\theta_{old}(a|s)}A_{\theta_{old}}(s,a) - C \cdot D_{KL}^{max}(\theta_{old},\theta)\right] \tag{63}$$

However, penalty coefficient C (constrain of KL divergence) will lead to small step size in policy update. Hence, a trust region constraint δ is used to constrain KL divergence by

$$maximize_{\theta}\mathbb{E}\left[\frac{\theta(a|s)}{\theta_{old}(a|s)}A_{\theta_{old}}(s,a)\right], s.t. D_{KL}^{max}(\theta_{old},\theta) \leq \delta \tag{64}$$

therefore step size in policy update is enlarged robustly. New improved policy is obtained in trust region by maximizing objective $L_{\theta_{old}}(\theta)$, s.t. $D_{KL}^{max}(\theta_{old},\theta) \leq \delta$. This objective can be simplified further (Schulman et al., 2017a) and new policy θ is obtained by

$$maximize_{\theta}\left[\nabla_{\theta}L_{\theta_{old}}(\theta)|_{\theta=\theta_{old}} \bullet (\theta - \theta_{old})\right]$$

$$\text{s.t.}\frac{1}{2}\|\theta - \theta_{old}\|^2 \leq \delta \tag{65}$$

PPO The objective of TRPO is $maximize_{\theta}\mathbb{E}\left[\frac{\theta(a|s)}{\theta_{old}(a|s)}A_{\theta_{old}}(s,a)\right], s.t. D_{KL}^{max}(\theta_{old},\theta) \leq \delta$, in which a fixed trust region constraint δ is used to constrain KL divergence instead of penalty coefficient$C$. Fixed trust region constraint δ leads to a reasonable step size in policy update therefore stability in convergence is improved and convergence speed is acceptable. However, objective of TRPO is obtained in implementation by conjugate gradient method (Schulman et al., 2017b) that is computationally expensive.

PPO optimizes objective $maximize_{\theta}\mathbb{E}\left[\frac{\theta(a|s)}{\theta_{old}(a|s)}A_{\theta_{old}}(s,a) - C \cdot D_{KL}^{max}(\theta_{old},\theta)\right]$ from two aspects: (1) *probability ratio* $r(\theta) = \frac{\theta(a|s)}{\theta_{old}(a|s)}$ in objective is constrained in interval $[1-\epsilon, 1+\epsilon]$ by introducing *"surrogate" objective*

$$L^{CLIP}(\theta) = \mathbb{E}\{\min[r(\theta)A, clip(r(\theta), 1-\epsilon, 1+\epsilon)]\} \tag{66}$$

where ϵ is a hyperparameter, to penalize changes of policy that move $r(\theta)$ away from 1 (Schulman et al., 2017b); (2) penalty coefficient C is replaced by *adaptive penalty coefficient* β that increases or decreases according to the expectation of KL divergence in new update. To be exact,

$$if d\left\langle\frac{d_{targ}}{1.5}, \beta \leftarrow \frac{\beta}{2}; if d\right\rangle d_{targ} \times 1.5, \beta \leftarrow \beta \times 2 \tag{67}$$

where $d = \mathbb{E}[D_{KL}^{max}(\theta_{old},\theta)]$ and d_{targ} denotes target value of KL divergence in each policy update, therefore *KL-penalized objective* is obtained by

$$L^{KLPEN}(\theta) = \mathbb{E}\left[\frac{\theta(a|s)}{\theta_{old}(a|s)} A_{\theta_{old}}(s,a) - \beta \cdot D_{KL}^{max}(\theta_{old}, \theta)\right] \quad (68)$$

In the implementation with neural network, loss function is required to combine the policy surrogate and value function error (Schulman et al., 2017b), and entropy are also used in objective to encourage exploration. Hence, *combined surrogate objective* is obtained by

$$L^{CLIP+VF+S}(\theta) = \mathbb{E}\Big[L^{CLIP}(\theta) + c_1 L^{VF}(\theta) + c_2 S(\pi_\theta|s)\Big] \quad (69)$$

where c_1, c_2, S and $L^{VF}(\theta)$ denote two coefficients, entropy bonus and square-error loss respectively. Objectives of PPO ($L^{CLIP+VF+S}(\theta)$ and $L^{KLPEN}(\theta)$) is optimized by SGD that costs less computing resource than conjugate gradient method. PPO is implemented with actor-critic architecture, therefore it converges faster than TRPO.

Analytical comparisons

To provide a clear understanding about advantages and disadvantages of different motion planning algorithms, we divide these algorithms into four groups: traditional algorithms, classical ML algorithms, optimal value RL and policy gradient RL. Comparisons are made according to principles of the algorithm mentioned in section II, III, IV and V. First, *direct comparisons* of the algorithm in each group are made to provide a clear understanding about the *input*, *output*, and *key features* of these algorithms. Second, *analytical comparisons* of all motion planning algorithms are made to provide a comprehensive understanding about performances and applications of the algorithm, according to criteria summarized. Third, *analytical comparisons about the convergence* of RL-based motion planning algorithms are specially made, because RL-based algorithms are the research focus recently.

Direct comparisons of motion planning algorithm

Traditional algorithms This group includes graph search algorithms, sampling-based algorithms, interpolating curve algorithms and reaction-based algorithms. Table 1 lists their input, output, and key features. According to Table 1: (1) Graph search algorithms, sampling-based algorithms, and interpolating curve algorithms use graph or map of workspace as the input and global trajectories are generated directly, while reaction-based algorithms use different types of information as the input. (2) Graph search algorithms find the shortest and collision-free trajectories by the search methods (e.g., best-first search). For example, Dijkstra's algorithm is based on best-first search. However, the search process is computationally expensive because search space is large, therefore heuristic function is used to reduce the search space and the shortest path is found by estimating the overall cost (e.g., A*). (3) Sampling-based algorithms randomly sample collision-free trajectories in search space (e.g., PRM), and constraints (e.g., non-holonomic constraint) are needed for some algorithms (e.g., RRT) in the sampling process. (4) Interpolating curve algorithms plan their path by mathematical rules, and then planned path is smoothed by CAGD. (5) In reaction-based algorithms, the moving directions of robots in PFM (APF) are the gradients of the converged and combined potential field function U. The velocity of robots in VOM is selected from RAV that is related to VO and RV. Exhaustive search and heuristic search can be used in the velocity selection process of VOM, and selected velocity must maximize the objective function U (e.g., distance traveled and motion time). Before velocity selection process of DWA, it is necessary to reduce the search space of velocity to obtain the resulting search space $V_r = V_s \cap V_a \cap V_d$. Proper velocity of robots in DWA is selected from V_r, and selected velocity must maximize the objective function U. Note that U includes a *measure of progress towards a goal location*, the *forward velocity of the robot*, and the *distance to the next obstacle on the trajectory*. (6) Outputs of graph search algorithms, sampling-based algorithms, and interpolating curve algorithms are trajectories. The outputs of PFM (APF) are directions of robots according to the gradient of the converged and combined potential field function U, while outputs of VOM and DWA are selected velocity among possible velocities.

Classical ML algorithms This group includes MSVM, LSTM, MCTS and CNN. These algorithms are listed in Table 2. According to that: (1) MSVM, LSTM and MCTS use well-prepared vector as the input, while CNN can directly use image as its input. (2) LSTM and MCTS can output time-sequential actions, because of their structures (e.g., tree) that can store and learn time-sequential features. MSVM and CNN cannot output time-sequential actions because they output one-step prediction by performing trained classifier. (3) MSVM plans the motion of the robot by training a maximum margin classifier. LSTM stores and processes inputs in its cell that is a stack structure, and then actions are outputted by performing trained LSTM model. MCTS is the combination of Monte-Carlo method and search tree. Environmental states and values are stored and updated in its node of tree, therefore actions are outputted by performing trained MCTS model. CNN converts high-dimensional images to low-dimensional features by convolutional layers. These low-dimensional features are used to train a CNN model, therefore actions are outputted by performing trained CNN model.

Table 1 Comparisons of traditional planning algorithm

Classification	Example	Input	Key features	Output
Graph search algorithm	Dijkstra's [1]	Graph or map	1. Best-first search (large search space) 2. Heuristic function in cost estimation	Trajectory
	A* [1,2]			
Sampling based algorithm	PRM [1]		1. Random search (suboptimal path) 2. Non-holonomic constraint	
	RRT [1,2]			
Interpolating Curve algorithm	Line and circle Clothoid curves Polynomial curves Bezier curves Spline curves		1. Mathematical rules 2. Path smoothing using CAGD	
Reaction based algorithm	PFM (APF)	Robot configurations (e.g., position)	1. Different potential field functions U for different targets (e.g., goal, obstacle) 2. Combined U and gradient of U	Moving directions
	VOM	Positions and velocities (robot and obstacles)	1. VO, RV and RAV 2. Exhaustive/global search, and heuristic search according to objective function U	Selected velocity among possible velocities
	DWA	Robot's position, distances to goal/obstacles, (v, w), and kinematics of robot	1. V_s, V_a, V_d, V_r 2. Velocity selection according to objective function U	

Table 2 Comparison of classical ML algorithms

Algorithm	Input	Key features	Output
MSVM	Vector	Maximum margin classifier	None-sequential actions
LSTM	Vector	Cell (stack structure)	Time-sequential actions
MCTS	Vector	Monte-carlo method/Tree structure	Time-sequential actions
CNN	Image	Convolutional layers/Weight matrix	None-sequential actions

Optimal value RL This group here includes Q learning, DQN, double DQN, and dueling DQN. Features of algorithms here include the replay buffer, objectives of algorithm, and method of weight update. Comparisons of these algorithms are listed in Table 3. According to that: (1) Q learning normally uses well-prepared vector as the input, while DQN, double DQN and dueling DQN use images as their input because these algorithms use convolutional layer to process high-dimensional images. (2) Outputs of these algorithms are time-sequential actions by performing trained model. (3) DQN, double DQN and dueling DQN use replay buffer to reuse the experience, while Q learning collects experiences and learns from then in an online way. (4) DQN, double DQN and dueling DQN use MSE e^2 as their objectives. Their differences are: first, DQN obtains action values by neural network $Q(s, a; \theta)$, while Q learning obtains action values by querying the Q-table; second, double DQN uses another neural network θ^- to evaluate selected actions to obtain better action values by $Q\left(s', \text{arg}max_a Q\left(s', a'; \theta'\right); \theta^-\right)$; third, dueling DQN obtains action values by dividing them to advantage values and state values. The constraint $\mathbb{E}_{a\sim\pi(s)}[A(s, a)] = 0$ is used on the advantage value, therefore a better action value is obtained by $Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \left\{A(s, a; \theta, \alpha) - \frac{1}{|\mathcal{A}|}\sum_{a_t^- \in \mathcal{A}} A(s_t, a_t^-; \theta, \alpha)\right\}$. Networks that approximate action value in these algorithms are updated by minimizing MSE with gradient descent approach.

Table 3 Comparison of optimal-value RL

Algorithm	Input	Output	Replay buffer	Objective	Method of weight update
Q learning	Vector	Time-sequential actions	No	e^2 where $e = r + \gamma \max_a Q(s\prime, a\prime) - Q(s, a)$	Gradient descent
DQN	Image	Time-sequential actions	Yes	e^2 where $e = r + \gamma \max_a Q(s\prime, a\prime; \theta\prime) - Q(s, a; \theta)$	Gradient descent
Double DQN	Image	Time-sequential actions	Yes	e^2 where $e = r + \gamma Q(s^{'}, \arg max_a Q\left(s^{'}, a^{'}; \theta^{'}\right); \theta^{-}) - Q(s, a; \theta)$	Gradient descent
Dueling DQN	Image	Time-sequential actions	Yes	e^2 where $e = r + \gamma max_a Q(s^{'}, a^{'}; \theta) - Q(s, a; \theta)$ and $Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \left\{A(s, a; \theta, \alpha) - \frac{1}{\lvert\mathcal{A}\rvert} \sum_{a_t^- \in \mathcal{A}} A(s_t, a_t^-; \theta, \alpha)\right\}$	Gradient descent

Table 4 Comparison of policy gradient RL

Algorithm	Input	Output	Actor-critic architecture	Multi-thread method	Replay buffer	Objective	Method of weight update
PG	Image/vector	Time-sequential actions	—*	—	—	1	Gradient ascent
AC	Image/vector	Time-sequential actions	Yes	—	—	Critic: 2 Actor: 3	Gradient ascent
A3C	Image/vector	Time-sequential actions	Yes	Yes	—	Critic: 4 Actor: 5	Gradient ascent Asynchronous update
A2C	Image/vector	Time-sequential actions	Yes	Yes	—	Critic: 6 Actor: 7	Gradient ascent Synchronous update
DPG	Image/vector	Time-sequential actions	Yes	—	Yes	Critic: 8 Actor: 9	Gradient ascent
DDGP	Image/vector	Time-sequential actions	Yes	—	Yes	Critic: 10 Actor: 11	Gradient ascent Soft update
TRPO	Image/vector	Time-sequential actions	Yes	—	—	Critic: 12 Actor: 13	Gradient ascent
PPO	Image/vector	Time-sequential actions	Yes	—	—	Critic: 14 Actor: 15	Gradient ascent

*Here the mark "—" denotes "No"
1. $\mathbb{E}_{\tau \sim \pi_\theta(\tau)}[R(\tau)]$
2. $e^2, e = r_t + V(s_{t+1}; w) - V(s_t; w)$
3. $\pi_\theta(a_t|s_t) \bullet e$
4. $e^2, e = \sum_{i=t}^{T} \gamma^{i-t} r_i + V(s_{t+n}; w) - V(s_t; w)$
5. $\pi_\theta(a_t|s_t) \bullet e + \beta H(\pi(s_t; \theta)$
6. $e^2, e = \sum_{i=t}^{T} \gamma^{i-t} r_i + V(s_{t+n}; w) - V(s_t; w)$
7. $\pi_\theta(a_t|s_t) \bullet e + \beta H(\pi(s_t; \theta)$
8. $e^2, e = r_t + \gamma Q^w(s_{t+1}, \mu_\theta(s_{t+1})) - Q^w(s_t, a_t)$
9. $Q^w(s_t, a_t)$, objective gradient: $\nabla_\theta \mu_\theta(s) \nabla_a Q^w(s, a)|_{a=\mu_\theta(s)}$
10. $e^2, e = r_i + \gamma Q^{'}\left(s_{i+1}, \mu^{'}\left(s_{i+1}; \theta^{\mu^{'}}\right); \theta^{Q^{'}}\right) - Q(s_i, a_i; \theta^Q)$
11. $Q(s_i, a_i; \theta^\mu)$, objective gradient: $\nabla_{\theta^\mu} \mu(s_i; \theta^\mu) \nabla_a Q(s_i, a; \theta^Q)|_{a=\mu(s_i)}$
12. $A_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + \gamma\lambda^{T-t}\delta(s_T), \delta_t = r_t + \gamma V(s_{t+1}; w) - V(s_t; w)$
13. $\mathbb{E}[\frac{\theta(a|s)}{\theta_{old}(a|s)} A_{\theta_{old}}(s, a)], s.t. D_{KL}^{max}(\theta_{old}, \theta) \leq \delta$
14. $A_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + \gamma\lambda^{T-t}\delta(s_T), \delta_t = r_t + \gamma V(s_{t+1}; w) - V(s_t; w)$
15. (1) $L^{KLPEN}(\theta) = \mathbb{E}[\frac{\theta(a|s)}{\theta_{old}(a|s)} A_{\theta_{old}}(s, a) - \beta \bullet D_{KL}^{max}(\theta_{old}, \theta)]$
(2) $L^{CLIP+VF+S}(\theta) = \mathbb{E}[L^{CLIP}(\theta) + c_1 L^{VF}(\theta) + c_2 S(\pi_\theta|s)]$
where $L^{CLIP}(\theta) = \mathbb{E}\{\min[r(\theta)A, clip(r(\theta), 1-\epsilon, 1+\epsilon)]\}$

Policy gradient RL Algorithms in this group include PG, AC, A3C, A2C, DPG, DDPG, TRPO, and PPO. Features of them include actor-critic architecture, multi-thread method, replay buffer, objective of algorithm, and method of weight update. Comparisons of these algorithms are listed in Table 4. According to that: (1) Inputs of policy gradient RL can be image or vector, and image is used as inputs under the condition that convolutional layer is used as preprocessing component to convert high-dimensional image to low-dimensional feature. (2) Outputs of policy gradient RL are time-sequential actions by performing trained policy $\pi(\mathrm{s}) : s \rightarrow a$. (3) Actor-critic architecture is not used in PG, while other policy gradient RL are implemented with actor-critic architecture. (4) A3C and A2C use multi-thread method to collect data and update their network, while other policy gradient RL are based on single thread in data collection and network update. (5) DPG and DDPG use replay buffer to reuse data in an offline way, while other policy gradient RL learn online. (6) The objective of PG is defined as the expectation of accumulative rewards in the episode by $\mathbb{E}_{\tau\sim\pi_\theta(\tau)}[R(\tau)]$. Critic objectives of AC, A3C, A2C, DPG and DDPG are defined as MSE e^2, and their critic networks are updated by minimizing the MSE. However, their actor objectives are different because: first, actor objective of AC is defined as $\pi_\theta(a_t|s_t)\cdot e$; second, policy entropy is added on $\pi_\theta(a_t|s_t)\cdot e$ to encourage the exploration, therefore actor objectives of A3C and A2C are defined by $\pi_\theta(a_t|s_t)\cdot e+\beta H(\pi(s_t;\theta)$; third, DPG and DDPG use action value as their actor objectives by $Q(s_i, a_i;\theta^\mu)$ that are approximated by neural network, and their policy networks (actor networks) are updated by obtaining objective gradient $\nabla_{\theta^\mu}\mu(s_i;\theta^\mu)\nabla_a Q(s_i, a;\theta^Q)|_{a=\mu(s_i)}$. Critic objectives of TRPO and PPO are defined as the advantage value by $A_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \ldots + \gamma\lambda^{T-t}\delta(s_T)$ where $\delta_t = r_t + \gamma V(s_{t+1}; w) - V(s_t; w)$, and their critic networks w are updated by minimizing δ_t^2. Actor objectives of TRPO and PPO are different: objective of TRPO is defined as $\mathbb{E}[\frac{\theta(a|s)}{\theta_{old}(a|s)}A_{\theta_{old}}(s, a)], s.t. D_{KL}^{max}(\theta_{old}, \theta) \leq \delta$ in which a fixed trust region constraint δ is used to ensure the monotonous update of policy network θ, while PPO uses "surrogate" $[1-\epsilon, 1+\epsilon]$ and adaptive penalty β to ensure a better monotonous update of policy network, therefore PPO has two objectives that are defined as $L^{KLPEN}(\theta) = \mathbb{E}[\frac{\theta(a|s)}{\theta_{old}(a|s)}A_{\theta_{old}}(s, a) - \beta\cdot D_{KL}^{max}(\theta_{old}, \theta)]$ (KL-penalized objective) and $L^{CLIP+VF+S}(\theta) = \mathbb{E}[L^{CLIP}(\theta) + c_1 L^{VF}(\theta) + c_2 S(\pi_\theta|s)]$ (surrogate objective) where $L^{CLIP}(\theta) = \mathbb{E}\{\min[r(\theta)A, clip(r(\theta), 1-\epsilon, 1+\epsilon)]\}$. (7) Policy network of policy gradient RL are all updated by gradient ascent method $\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta)$. Policy networks of A3C and A2C are updated in asynchronous and synchronous ways, respectively. Networks of DDPG are updated in a "soft" way by $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$ and $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$.

Analytical comparisons of motion planning algorithms

Here analytical comparisons of motion planning algorithms are made according to *general criteria* we summarized. These criteria consist of six aspects: *local or global planning*, *path length*, *optimal velocity*, *reaction speed*, *safe distance* and *time-sequential path*. The speed and stability of network convergence for optimal value RL and policy gradient RL are then compared analytically because convergence speed and stability of RL in robotic motion planning are recent research focus.

Comparisons according to general criteria

Local (reactive) or global planning This criterion denotes the area where the algorithm is used in most case. Table 5 lists planning algorithms and the criteria they fit. According to Table 5: (1) Graph search algorithms plan their path globally by search methods (e.g., depth-first search, best-first search) to obtain a collision-free trajectory on the graph or map. (2) Sampling-based algorithms samples local or global workspace by sampling methods (e.g., random tree) to find collision-free trajectories. (3) Interpolating curve algorithms draw fixed and short trajectories by mathematical rules to avoid local obstacles. (4) Reaction based algorithms plan local paths or reactive actions according to their objective functions. (5) MSVM and CNN make one-step prediction by trained classifiers to decides their local motions. (6) LSTM, MCTS, optimal value RL and policy gradient RL can make time-sequential motion planning from the start to destination by performing their trained models. These models include the stack structure model of LSTM, tree model of MCTS and matrix weight model of RL. These algorithms fit global motion planning tasks theoretically if size of workspace is not large, because it is hard to train a converged model in large workspace. In most case, models of these algorithms are trained in local workspace to make time-sequential predictions by performing their trained model or policy $\pi(\mathrm{s}) : s \rightarrow a$.

Path length This criterion denotes the length of planned path that is described as *optimal path (the shortest path)*, *suboptimal path*, and *fixed path*. Path length of algorithms are listed in Table 5. According to that: (1) Graph search algorithms can find a shortest path by performing search methods (e.g., best-first search) in the graph or map. (2) Sampling-based algorithms plan a suboptimal path. Their sampling method (e.g., random tree) leads to the insufficient sampling that only covers a part of cases and suboptimal

Table 5 Analytical comparisons according to general criteria

Algorithm	Local/global	Path length	Optimal velocity	Reaction speed	Safe distance	Time-sequential path
Graph search alg	Global	Optimal path (shortest path)	–*	Slow	Fixed distance/High Collison rate	No
Sampling-based alg	Local/Global	Suboptimal path	–	Slow	Fixed distance/High Collison rate	No
Interpolating curve alg	Local	Fixed path	–	Medium	Fixed distance	No
Reaction based alg	Local	Optimal path	Optimal velocity	Medium	Suboptimal distance	No
MSVM	Local	Suboptimal path	Suboptimal velocity	Fast	Suboptimal distance	No
LSTM	Local/Global	Suboptimal path	Suboptimal velocity	Fast	Suboptimal distance	Yes
MCTS	Local/Global	Optimal path	–	Fast	Optimal distance	Yes
CNN	Local	Suboptimal path	Suboptimal velocity	Fast	Suboptimal distance	No
Q learning	Local/Global	Optimal path	Optimal velocity	Fast	Optimal distance	Yes
DQN	Local/Global	Optimal path	Optimal velocity	Fast	Optimal distance	Yes
Double DQN	Local/Global	Optimal path	Optimal velocity	Fast	Optimal distance	Yes
Dueling DQN	Local/Global	Optimal path	Optimal velocity	Fast	Optimal distance	Yes
PG	Local/Global	Optimal path	Optimal velocity	Fast	Optimal distance	Yes
AC	Local/Global	Optimal path	Optimal velocity	Fast	Optimal distance	Yes
A3C	Local/Global	Optimal path	Optimal velocity	Fast	Optimal distance	Yes
A2C	Local/Global	Optimal path	Optimal velocity	Fast	Optimal distance	Yes
DPG	Local/Global	Optimal path	Optimal velocity	Fast	Optimal distance	Yes
DDGP	Local/Global	Optimal path	Optimal velocity	Fast	Optimal distance	Yes
TRPO	Local/Global	Optimal path	Optimal velocity	Fast	Optimal distance	Yes
PPO	Local/Global	Optimal path	Optimal velocity	Fast	Optimal distance	Yes

*The mark "–" denotes the performance that cannot be evaluated

path is obtained. (3) Interpolating curve algorithms plan their path according to mathematical rules that lead to a fixed length of path. (4) Reaction-based algorithms can find the shortest path if their objective functions are related to shortest distance travelled, and then robots will move towards directions that maximize objective functions. (5) MSVM, LSTM and CNN plan their path by performing models that are trained with human-labeled dataset, therefore suboptimal path is obtained. MCTS can receive the feedback (reward) from the environment to update its model (tree), therefore it is possible to find a shortest path like other RL algorithms. (6) RL algorithms (optimal value RL and policy gradient RL) can generate optimal path under the condition that reasonable penalty is used to punish moved steps in the training, therefore optimal path is obtained by performing the trained RL policy.

Optimal velocity This criterion denotes the ability to *tune the velocity* when algorithms plan their path, therefore robot can reach the destination with minimum time in travelled path. This criterion is described as *optimal velocity* and *suboptimal velocity*. Table 5 lists performance of algorithms in the optimal velocity. According to that: (1) Performance of graph search algorithms, sampling-based algorithms and interpolating algorithms in the velocity tuning cannot be evaluated, because these algorithms are only designed for the path planning to find a collision-free trajectory in the graph or map. (2) Among reaction-based algorithms, the velocity of robots in PFM can be tuned according to obtained potential field, while velocities of robots in VOM and DWA are dynamically selected among their possible velocities according to their objective functions. Hence, reaction-based algorithms can realize optimal velocity. (3) MSVM, LSTM, and CNN can output actions that are in the format $\boldsymbol{v} = [v_x, v_y]$ where v_x and v_y are velocity in x and y axis, if algorithms are trained with these vector labels. However, these velocity-related labels are all hard-coded artificially. Time to reach

destination heavily relies on the artificial factor, therefore these algorithms cannot realize optimal velocity. MCTS can realize optimal velocity theoretically if the size of action space is small. However, the case where MCTS is used in velocity-related tasks has not been found. Large action space will lead to a large search tree, and it is hard to train such a large tree model. In normal case, MCTS is used in one-step prediction via its trained tree model, therefore the state can transit into next state but velocity in this state transition process will not be considered. Hence, its ability in velocity tunning cannot be evaluated. (4) optimal value RL and policy gradient RL can realize optimal velocity by attaching the penalty to consumed time in the training. These algorithms can automatically learn how to choose the best velocity in the action space for training to cost time as less as possible, therefore robots can realize optimal velocity by performing trained policy. Note that in this case, actions in optimal value RL and policy gradient RL must be in format of $[v_x, v_y]$ and action space that contains many action choices must be pre-defined artificially.

Reaction speed This criterion denotes the *computational cost* or *time cost* of the algorithm to react dynamic obstacles. Computational cost of some algorithms (e.g., traditional non-AI algorithms) would be easy to obtain by counting the number of lines and functions in the source code (Kinnunen et al., 2011). However, the problems of this approach are that all source codes should be found first, and the result would still depend on the chosen programming language and the quality of the implementation. It is hard to obtain clear computational costs, thus here reaction speed is briefly described analytically using three levels: *slow*, *medium* and *fast*. Table 5 lists reaction speed of the algorithms. According to that: (1) Graph search algorithms and sampling-based algorithms rely on planned trajectories in the graph or map to avoid obstacles. However, the graph or map is updated in a slow frequency normally, therefore reaction speed of these algorithms is slow. (2) Interpolating curve algorithms plan their path according to mathematical rules with limited and predictable time in computation, therefore reaction speed of these algorithms is medium. (3) Reaction-based algorithms should first update their potential field (for PFM) and search space (for VOM and DWA) in small local areas to select moving directions or proper velocities. These update processes take lesser time than that of graph search algorithms. Hence, reaction speed of reaction-based algorithms is medium. (4) classical ML algorithms, optimal value RL and policy gradient RL react to obstacles by performing trained model or policy $\pi(\mathrm{s}) : s \rightarrow a$ that maps state of environment to a probability distribution $P(\mathrm{a}|\mathrm{s})$ or to actions directly (e.g., DDPG). This process is fast and time cost can be ignored, therefore reaction speed of these algorithms is fast.

Safe distance This criterion denotes the *ability to keep a safe distance to obstacles*. Safe distance is described as three level that includes f*ixed distance*, *suboptimal distance* and *optimal distance*. Table 5 lists the performance of algorithms. According to that: (1) Graph search algorithms and sampling-based algorithms keep a fixed distance to static obstacles by hard-coded setting in robotic application. However, high collision rate is inevitable in dynamic environment because of slow update frequency of graph or map. (2) Interpolating algorithms keep a fixed distance to static and dynamic obstacles according to mathematical rules. (3) Reaction-based algorithms can keep safe and dynamic distances to obstacles theoretically according to their potential field functions (for PFM) or objective functions (for VOM and DWA). However, their updates in potential fields or search spaces cost a short period of time that cannot be ignored, therefore increasing the rate of collision with obstacles especially in high-speed, dense, and dynamic scenarios. Hence, they keep a suboptimal distance to obstacles. (4) MSVM, LSTM and CNN keep a suboptimal distance to static and dynamic obstacles. Suboptimal distance is obtained by performing a model that is trained with human-labeled dataset. MCTS can output a collision-free time-sequential path and keep a safe distance to obstacles theoretically as other RL algorithms. (5) optimal value RL and policy gradient RL keep a safe and dynamic distance to static and dynamic obstacles with the computational cost that can be ignored, by performing a trained policy $\pi(\mathrm{s}) : s \rightarrow a$. This policy is trained under the condition that the penalty is used to punish the close distances between robot and obstacles in the training, therefore algorithms will automatically learn how to keep an optimal distance to obstacles when robot moves towards destination.

Time-sequential path This criterion denotes *whether an algorithm fits time-sequential task or not*. Table 5 lists algorithms that fit time-sequential planning. According to that: (1) Graph search algorithms, sampling-based algorithms, interpolating curve algorithms, and reaction-based algorithms plan their path according to the graph or map, mathematical rules, obtained potential field or search spaces, regardless of environmental state in each time step. Hence, these algorithms cannot fit time-sequential task. (2) MSVM and CNN output actions by one-step prediction that has no relation with environmental state in each time step. (3) LSTM and MCTS store environmental state in each time step in their cells and nodes respectively, and their models are updated by learning from these time-related experience. Time-sequential actions are outputted by performing trained models, therefore these algorithms fit time-sequential task. (4) Optimal value RL and policy gradient RL train their policy network by learning from environmental state in each time step. Time-sequential

actions are outputted by performing trained policy, therefore these algorithms fit time-sequential task.

Comparisons of convergence speed and stability

Convergence speed Here *poor*, *reasonable*, *good*, and *excellent* are used to describe the performance of convergence speed. Table 6 lists the performance of optimal value RL and policy gradient RL. According to that: (1) Q learning only fits simple motion planning with small-size Q table. It is hard to converge for Q learning with large-size Q table in complex environment. Over-estimation of Q value also leads to poor performance of Q learning if neural network is used to approximate Q value. (2) DQN suffers the over-estimation of Q value when CNN is used to approximate Q values, but DQN learns from the experience in replay buffer that make network reuse high-quality experience efficiently. Hence, convergence speed of DQN is reasonable. (3) Double DQN uses another network θ^- to evaluate actions that are selected by θ'. New Q value with less over-estimation is obtained by $Q\left(s', \arg\max_a Q\left(s', a'; \theta'\right); \theta^-\right)$, therefore the convergence speed is improved. (4) Dueling DQN finds better Q value by: first, dividing action value to state value and advantage value $Q(s,a) = V(s,a) + A(s,a)$; second, constraining advantage value $A(s,a)$ by $\mathbb{E}_{a\sim\pi(s)}[A(s,a)] = 0$, therefore new action value is obtained by $Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left\{A(s,a;\theta,\alpha) - \frac{1}{|\mathcal{A}|}\sum_{a_t^-\epsilon\mathcal{A}} A\left(s_t, a_t^-;\theta,\alpha\right)\right\}$. Hence, performance of dueling DQN in convergence speed is good. (5) PG updates its policy according to the episode rewards by $\mathbb{E}_{\tau\sim\pi_\theta(\tau)}[R(\tau)]$, therefore poor performance in convergence speed is inevitable. (5) AC uses the critic network to evaluate actions selected by the actor network, therefore speeding up the convergence. (6) A3C and A2C use multi-thread method to improve convergence speed directly, and policy entropy is also used to encourage exploration. These methods indirectly enhance the convergence speed. (7) Performance of DPG and DDPG in convergence speed is good because: first, their critics are unbiased critic networks obtained by CFA and gradient Q learning; second, their policies are deterministic policy $\mu_\theta(s)$ that is faster than the stochastic policy $\pi_\theta(a_t|s_t)$ in convergence speed; third, their networks are updated offline with replay buffer; fourth, noise is used in DDPG to encourage the exploration; (8) TRPO makes a great improvement in convergence speed by adding trust region constraint to policies by $D_{KL}^{max}(\theta_{old},\theta) \le \delta$, therefore its networks are updated monotonously by maximizing its objective $\mathbb{E}\left[\frac{\theta(a|s)}{\theta_{old}(a|s)} A_{\theta_{old}}(s,a)\right], s.t. D_{KL}^{max}(\theta_{old},\theta) \le \delta$. (9) PPO moves further in improving convergence speed by introducing "surrogate" objective $L^{CLIP+VF+S}(\theta) = \mathbb{E}\left[L^{CLIP}(\theta) + c_1 L^{VF}(\theta) + c_2 S(\pi_\theta|s)\right]$ and KL-penalized objective $L^{KLPEN}(\theta) = \mathbb{E}[\frac{\theta(a|s)}{\theta_{old}(a|s)} A_{\theta_{old}}(s,a) - \beta \cdot D_{KL}^{max}(\theta_{old},\theta)]$.

Convergence stability Table 7 lists convergence stability of optimal value RL and policy gradient RL. According to that: (1) Q learning update its action value every step, therefore the bias is introduced. Over-estimation of Q value leads to suboptimal update direction of Q value, if neural network is used as approximator. Hence, convergence stability of Q learning is poor. (2) DQN improves the convergence stability by replay buffer in which a batch of experiences are sampled for training and its network is update according to the batch loss. (3) Double DQN and dueling DQN can find a better action value than that of DQN by the evaluation network and advantage network respectively, therefore networks of these algorithms are updated towards a better direction. (4) PG updates its network according to the accumulative episode reward. This reduces bias caused by one-step rewards, but it introduces high variance. Hence, network of PG is updated with stability, but it is still hard to converge. (5) The performance of actor and critic network of AC is poor in early-stage training. This leads to a fluctuated update of networks in the beginning, although network is updated by gradient ascent method $\theta \leftarrow \theta + \nabla_\theta J(\theta)$. (6) A3C and A2C update their networks by multi-step rewards $\sum_{i=t}^{T} \gamma^{i-t} r_i$ that reduces the bias and improves convergence stability, although it will introduce some variance. Gradient ascent method also helps in convergence stability, therefore performance in their convergence stability is reasonable. (6) Unbiased critic, gradient ascent method and replay buffer contribute to good performance in convergence stability for DPG and DDPG. Additionally, target networks of DDPG are updated in a *soft* way by $\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}$ and $\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$ that also contributes to convergence stability. (7) Networks of TRPO and PPO is updated monotonously. TRPO achieves this goal by the trust region constraint $D_{KL}^{max}(\theta_{old},\theta) \le \delta$, while PPO uses "surrogate" objective $L^{CLIP+VF+S}(\theta) = \mathbb{E}\left[L^{CLIP}(\theta) + c_1 L^{VF}(\theta) + c_2 S(\pi_\theta|s)\right]$ and KL-penalized objective $L^{KLPEN}(\theta) = \mathbb{E}\left[\frac{\theta(a|s)}{\theta_{old}(a|s)} A_{\theta_{old}}(s,a) - \beta \cdot D_{KL}^{max}(\theta_{old},\theta)\right]$. Hence, performance of their networks in convergence stability is good.

Future directions of robotic motion planning

Here a common but complex real-world motion planning task is firstly given: *how to realize long-distance motion planning with safety and efficiency* (e.g., long-distance luggage delivery by robots)? Then research questions and directions are obtained by analyzing this task according to processing

Table 6 Comparison of speed in convergence for RL

Algorithm	Speed of convergence	Reasons
Q learning	Poor	Over-estimation of action value
DQN	Reasonable	Replay buffer
Double DQN	Good	Replay buffer
		Another network for evaluation
Dueling DQN	Good	Replay buffer
		Division of action value $Q(s,a) = V(s,a) + A(s,a)$
PG	Poor	High variance by $\mathbb{E}_{\tau\sim\pi_\theta(\tau)}[R(\tau)]$
AC	Reasonable	Actor-critic architecture
A3C	Good	Actor-critic architecture
		Multi-thread method
		Policy entropy
A2C	Good	Actor-critic architecture
		Multi-thread method
		Policy entropy
DPG	Good	Replay buffer
		Actor-critic architecture
		Deterministic policy
		Unbiased critic network
DDGP	Good	Replay buffer
		Actor-critic architecture
		Deterministic policy
		Unbiased critic network
		Exploration noise
TRPO	Good	Actor-critic architecture
		Fixed trust region constraint;
PPO	Excellent	Actor-critic architecture
		"surrogate" objective
		KL-penalized objective

steps that include *data collection*, *data preprocessing*, *motion planning* and *decision making* (Fig. 28).

Data collection To realize mentioned task, these questions should be considered firstly: (1) how to collect enough data? (2) how to collect high-quality data? To collect enough data in a short time, multi-thread method or cloud technology should be considered. Existing techniques seem enough to solve this question well. To collect high-quality data, existing works use prioritized replay buffer (Oh et al., 2018) to reuse high-quality data to train network. Imitation learning (Codevilla et al., 2018; Oh et al., 2018) is also used for the network initialization with expert experience, therefore network can converge faster (e.g., deep V learning (Chen et al., 2016, 2019)). Existing methods in data collection work well, therefore it is hard to make further optimization.

Data preprocess *Data fusion* and *data translation* should be considered after data is obtained. Multi-sensor data fusion algorithms (Durrant-Whyte & Henderson, 2008) fuse data that is collected from same or different type of sensors. Data fusion is realized from *pixel*, *feature*, and *decision* levels, therefore partial understanding of environment is avoided. Another way to avoid partial understanding of environment is the data translation that interpretates data to new format, therefore algorithms can have a better understanding about the relationship of robots and other obstacles (e.g., attention weight (Chen et al., 2019) and relation graph (Chen et al., 2020)). However, algorithms in data fusion and translation cannot fit all cases, therefore further works is needed according to the environment of application.

Motion planning: In this step, the *selection* and *optimization* of motion planning algorithms should be considered: (1) if traditional motion planning algorithms (e.g., A*, RRT) are selected for the task mentioned before, global trajectory from the start to destination will be obtained, but this process is computationally expensive because of the large search space. To solve this problem, the *combination of traditional algorithms and other ML algorithms* (e.g., CNN, DQN) may be a good choice. For example, RRT can be combined with DQN (Fig. 29) by using action value to predict directions of tree expansion, instead of the heuristic or random search.

Table 7 Comparison of stability in convergence for RL

Algorithms	Stability of convergence	Reasons
Q learning	Poor	Update in each step
		Over-estimation of action value
DQN	Reasonable	Replay buffer
Double DQN	Reasonable	Replay buffer
		Evaluation network
Dueling DQN	Reasonable	Replay buffer
		Advantage network
PG	Good	Update by trajectory rewards
		Gradient ascent update
AC	Poor (in the beginning)	Update in each step
		Gradient ascent update
A3C	Reasonable	Update by multi-step rewards
		Gradient ascent update
A2C	Reasonable	Update by multi-step rewards
		Gradient ascent update
DPG	Good	Unbiased critic
		Gradient ascent update
		Replay buffer
DDGP	Good	Unbiased critic
		Gradient ascent update
		Replay buffer
		Soft update
TRPO	Good	Monotonous update
		Gradient ascent update
PPO	Good	Monotonous update
		Gradient ascent update

(2) It seems impossible to use supervised learning to realize task mentioned above safely and quickly. Global path is impossible to be obtained by supervised learning that outputs one-step prediction. (3) Global path cannot be obtained by optimal value RL or policy gradient RL, but their performance in safety and efficiency is good locally by performing trained RL policy that leads to quick reaction, safe distance to obstacles, and shortest travelled path or time. However, it is *time-consuming to train a RL policy because of deficiencies in network convergence*. Existing works made some optimizations to improve convergence (e.g., DDPG, PPO) in games and physical robots to shorten training time of RL, but there is still a long way to go in real-world engineering and manufacturing for commercial purposes. Recent trend to improve the network convergence is to create hybrid architecture that is the fusion of high-performance components (e.g., replay buffer, actor-critic architecture, policy entropy, multi-thread method). Apart from optimizations of motion planning algorithms, *hardware planning* may be a possible direction to improve the performance of future motion planning system. Hardware planning refers to the reconfiguration and adjustment of hardware of robotic system, therefore robots can be reconfigured into different shapes (Salemi et al., 2006; Wang et al., 2020) to improve their performance in motion planning.

Decision: Traditional algorithms (e.g., A*) feature the global trajectory planning, while optimal value RL and policy gradient RL feature the safe and quick motion planning locally. It is a good direction to realize task mentioned above by combining traditional algorithm with RL. This is achieved by fusing the commands generated by each algorithm or functional module (e.g., functional modules in ROS) via *algorithm-level* and *system-level* data fusions. Multi-sensor fusion technique can not only fuse information of sensors from pixel and feature level as the inputs, but also fuse different types of decisional commands from decisional level in a *loose-coupled way*. Hence, overall path of the robot is expected to approximate the shortest path, and safety and efficiency can be ensured simultaneously. Recent state of art borrows the group decision making theory to sensor fusion or data fusion to solve the problem of how weights or impacts of each decisional command are determined from the consensus level and confidence level (Ji et al., 2020). Some pioneering ideas about how to better combine or integrate algorithms can

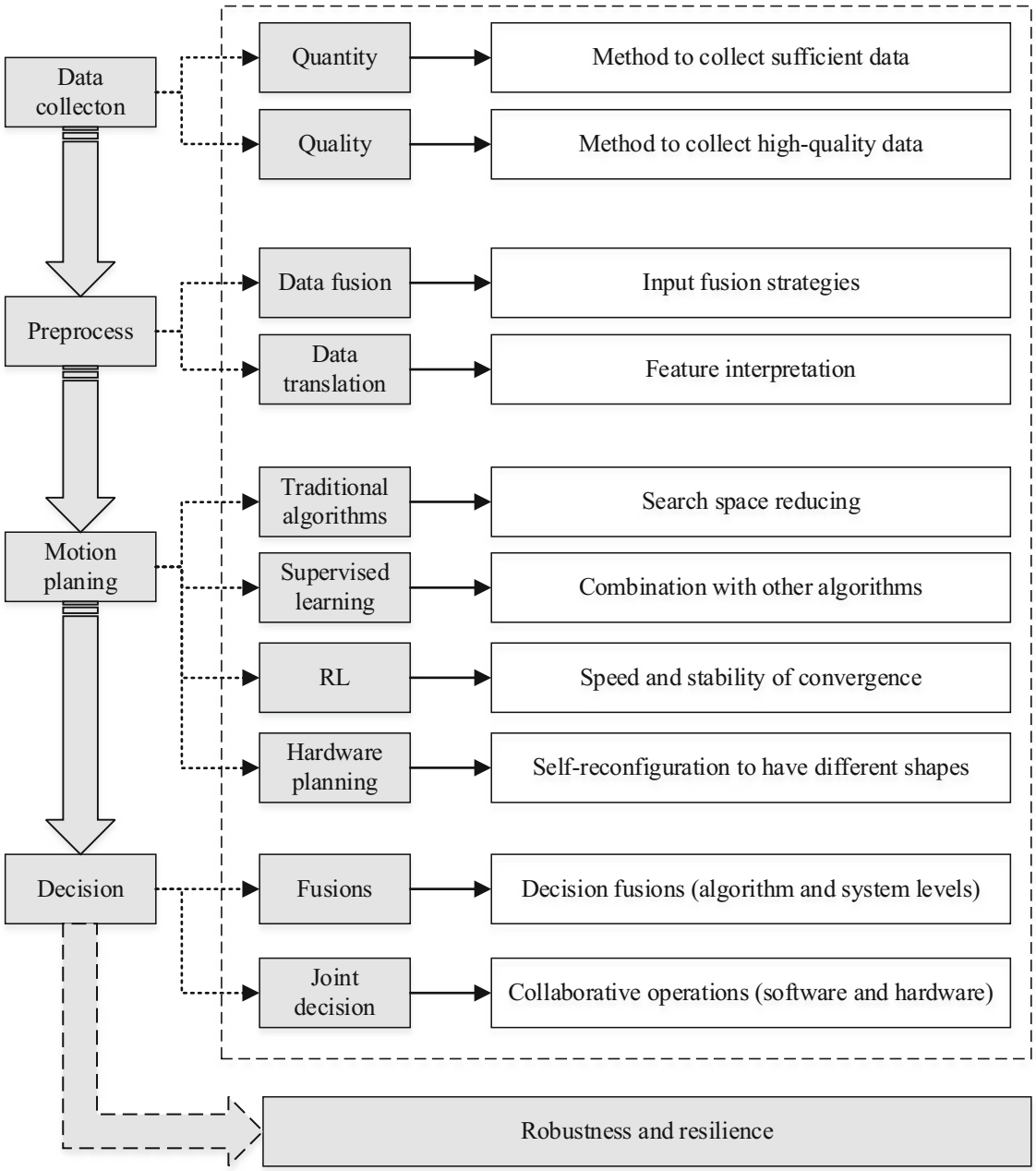


Fig. 28 Processing steps for motion planning task

be also seen in Zhang et al. (2019). Apart from hybrid algorithms or decisional commands, *joint decision making* may be helpful to improve the performance of future motion planning system. This is achieved via collaborative operations from software-based planning (algorithm-based planning) and hardware planning.

Finally, the *robustness* and *resilience* of motion planning system should be considered if robots are expected to have better performance in engineering and manufacturing for commercial use. Robustness refers to the ability to resist outside noise or attack (e.g., the cyber-attack in ROS), while resilience refers to the ability of the robot to recover its function after the robot is partially damaged (Wang et al., 2020). To conclude, Fig. 28 lists possible research directions, but attentions to improve the performance of robotic motion planning are expected to be paid on: (1) data fusion and translation of inputted features; (2) the optimization in traditional planning algorithms to reduce the search space by combining traditional algorithms with supervised learning or RL; (3) the optimization in network convergence for RL.

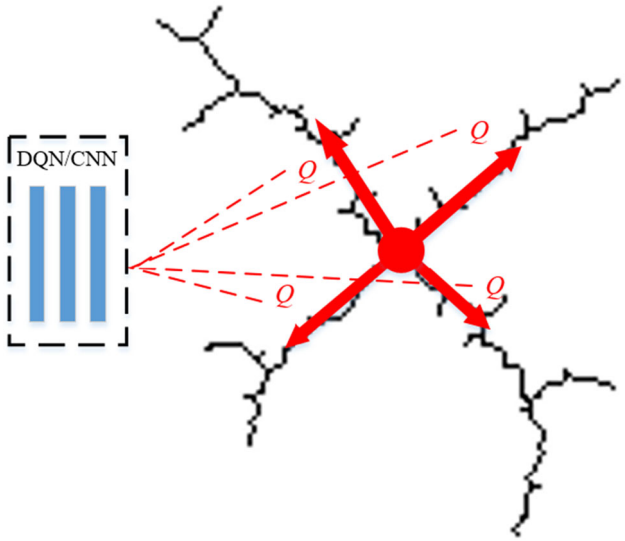


Fig. 29 Fusion of DQN or CNN with RRT

Conclusion

This paper carefully analyzes principles of robotic motion planning algorithms in section II-VI. These algorithms include traditional planning algorithms, classical ML, optimal value RL and policy gradient RL. Direct comparisons of these algorithms are made in section VII according to their principles. Hence, a clear understanding about mechanisms of motion planning algorithms is provided. Analytical comparisons of these algorithms are made in section VII according to the new criteria summarized that include local or global planning, path length, optimal velocity, reaction speed, safe distance, and time-sequential path. Hence, general performances of these algorithms and their potential application domains are obtained. The convergence speed and stability of optimal value RL and policy gradient RL are specially compared in section VII because they are recent research focus on robotic motion planning. Hence, a detailed and clear understanding of these algorithms in network convergence are provided. Finally, a common motion planning task is analyzed: long-distance motion planning with safety and efficiency (e.g., long-distance luggage delivery by robots) according to processing steps (data collection, data preprocessing, motion planning and decision making). Hence, potential research directions are obtained, and we hope they are useful to pave ways for further improvements of robotic motion planning algorithms or motion planning systems in academia, engineering, and manufacturing.

Author contributions Conceptualization: [CZ], [PF], [BH]; Methodology: [CZ]; Formal analysis and investigation: [CZ]; Writing—original draft preparation: [CZ]; Writing—review and editing: [PF], [BH]; Funding acquisition; Resources: [CZ], [BH], [PF]; Supervision: [PF], [BH].

Funding Open access funding provided by University of Eastern Finland (UEF) including Kuopio University Hospital. The authors did not receive support from any organization for the submitted work.

Declarations

Conflict of interests All authors certify that they have no affiliations with or involvement in any organization or entity with any financial interest or non-financial interest in the subject matter or materials discussed in this manuscript.

References

Arkin, R. C., Riseman, E. M., & Hansen, A. (1887). AuRA: an architecture for vision-based robot navigation. *Proceedings of the DARPA Image Understanding Workshop*, Los Angeles, CA, February 1987, pp. 417–413.

Babaeizadeh, M., Frosio, I., Tyree, S., Clemons, J., Kautz J. (2016). Reinforcement learning through asynchronous advantage Actor-Critic on a GPU. arXiv, arXiv:1611.06256 [cs.LG].

Bae, H., Kim, G., Kim, J., Qian, D., & Lee, S. (2019). Multi-robot path planning method using reinforcement learning. *Applied Science., 9*, 3057.

Bai, H., Cai, S., Ye, N., Hsu, D., & Lee, W. S. (2015). Intention-aware online POMDP planning for autonomous driving in a crowd. 2015 *IEEE International Conference on Robotics and Automation* (ICRA), Seattle, WA, pp. 454–460.

Bautista, G. D., Perez, J., Milanés, V., & Nashashibi, F. (2015). A review of motion planning techniques for automated vehicles. *IEEE Transactions on Intelligent Transportation Systems, 17*(4), 1–11.

Bautista, D. G., Perez, J., Lattarulo, R. A., Milanes, V., Nashashibi, F. (2014). Continuous curvature planning with obstacle avoidance capabilities in urban scenarios. *IEEE International Conference on Intelligent Transportation Systems*, pp. 1430–1435.

Bellemare, M. G., Dabney, W., Rémi, M. (2017). A distributional perspective on reinforcement learning. arXiv:1707.06887 [cs.LG].

Bilbeisi, K. M., & Kesse, M. (2017). Tesla: A successful entrepreneurship strategy. Morrow, GA: Clayton State University.

Borenstein, J., & Koren, Y. (1989). Real-time obstacle avoidance for fast mobile robots. *IEEE Transactions on Systems, Man, and Cybernetics, 19*(5), 1179–1187.

Borenstein, J., & Koren, Y. (1991). The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation, 7*(3), 278–288.

Bridle, J. S. (1990). Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern

recognition. In F. Fogleman Soulie & J. Herault (Eds.), *Neurocomputing: Algorithms, architectures and applications.* Springer-Verlag.

Brooks, R. A. (1986). A robust layered control system for a mobile robot. *IEEE Transactions on Robotics and Automation, 1*(1), 1–10.

Cai, M., Lin, Y., Han, B., Liu, C., & Zhang, W. (2017). On a simple and efficient approach to probability distribution function aggregation. *IEEE Transactions on Systems, Man, and Cybernetics: Systems, 47*(9), 2444–2453.

Chan, K. C., Lenard, C. T., Mills, T. M. (2012). An introduction to Markov chains. *In 49th Annual Conference of Mathematical Association of Victoria*, Melbourne, pp 40–47.

Chao, Y., Xiang, X., & Wang, C. (2020). Towards real-time path planning through deep reinforcement learning for UAV in dynamic environment. *Journal of Intelligent and Robotic Systems, 98*, 297–309.

Chen, Y., Liu, M., Everett, M., & How, J. P. (2016). Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning, arXiv:1609.07845v2 [cs.MA].

Chen, C., Liu, Y., Kreiss, S., & Alahi, A. (2019). Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning. *International Conference on Robotics and Automation (ICRA)*, pp. 6015–6022.

Chen, C., Hu, S., Nikdel, P., Mori, G., & Savva, M. (2020). Relational graph learning for crowd navigation. arXiv:1909.13165v3 [cs.RO].

Codevilla, F., Müller, M., López, A., Koltun, V. & Dosovitskiy, A. (2018). End-to-end driving via conditional imitation learning. *IEEE International Conference on Robotics and Automation (ICRA)*, **2018**, pp. 4693–4700.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms*. The MIT Press.

Coulom, R. (2006). Efficient selectivity and backup operators in Monte-Carlo tree search. *Computers and Games, 5th International Conference,* CG 2006, Turin, Italy, May 29–31, 2006.

Daniel, K., Nash, A., Koenig, S., & Felner, A. (2014). Theta*: Any-angle path planning on grids. *Journal of Artificial Intelligence Research, 39*(1), 533–579.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik, 1*, 269–271.

Dos Santos Mignon, A., & De Azevedo Da Rocha, R. L. (2017). An adaptive implementation of ϵ-greedy in reinforcement learning. *In Procedia Computer Science, 109*, 1146–1151.

Durrant-Whyte, H., & Henderson, T. C. (2008). Multi-sensor data fusion. In B. Siciliano & O. Khatib (Eds.), *Springer handbook of robotics. Springer handbooks.* Springer.

Everett, M., Chen, Y., How, J. P. (2018). Motion planning among dynamic, decision-making robots with deep reinforcement learning. *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems* (IROS), Madrid, pp. 3052–3059.

Evgeniou, T., & Pontil, M. (1999) Support vector machines: Theory and applications. In *Advanced Course on Artificial Intelligence*, Springer, Berlin, Heidelberg, pp. 249–257.

Fan, H., Zhu, F., Liu, C., Zhang, L., Zhuang, L., Li, D., Zhu, W., Hu, J., Li, H., & Kong, Q. (2008). Baidu apollo em motion planner. arXiv:1807.08048.

Farouki, R. T., & Sakkalis, T. (1994). Pythagorean-hodograph space curves. *Advances in Computational Mathematics, 2*(1), 41–66.

Ferguson, D., Howard, T. M., & Likhachev, M. (2008). Motion planning in urban environments. *Journal of Field Robotics, 25*(11–12), 939–960.

Ferguson, D., & Stentz, A. (2006). Using interpolation to improve path planning: The field D* algorithm. *Journal of Field Robotics, 23*(2), 79–101.

Fiorini, P., & Shiller, Z. (1998). Motion planning in dynamic environments using velocity obstacles. *International Journal of Robotics Research, 17*(7), 760–772.

Fortunato, M., Azar, M. G., Piot, B., et al. (2017). Noisy networks for exploration, arXiv:1706.10295 [cs.LG].

Fox, D., Burgard, W., & Thrun, S. (1997). The dynamic window approach to collision avoidance. *IEEE Robotics and Automation Magazine, 4*(1), 23–33.

Funke, J., Theodosis, P., Hindiyeh, R., et al. (2012). Up to the limits: Autonomous Audi TTS. IEEE Intelligent Vehicles Symposium (IV). *Alcala De Henares, 2012*, 541–547.

G., Samuel, G. (2017). Google sibling waymo launches fully autonomous ride-hailing service. *The Guardian 7*.

Gao, W., Hus, D., Lee, W. S., Shen, S., Subramanian, K. (2017). Intention-net: integrating planning and deep learning for goal-directed autonomous navigation. arXiv, arXiv:1710.05627 [cs.AI].

Gilhyun, R. (2018). Applying asynchronous deep classification networks and gaming reinforcement learning-based motion planners to mobile robots. *IEEE Robotics and Automation Society*, pp. 6268–6275.

Guy, S. J., Chhugani, J., Kim, C., Satish, N., Lin, M., Manocha, D., & Dubey P. (2009). Clearpath: Highly parallel collision avoidance for multi-agent simulation. *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 177–187.

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

Hasselt, H. V., Guez, A. & Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, no. 1.

He, K., Zhang, X., Ren, S., Sun, J. (2016). Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* Las Vegas, NV, pp. 770–778.

Hessel, M., Modayil, J., Van, H. H., et al. (2017). Rainbow: Combining improvements in deep reinforcement learning. arXiv:1710.02298 [cs.AI].

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation, 9*(8), 1735–1780.

Indrajaya, M. A., Affandi, A., & Pratomo, I. (2015). Design of geographic information system for tracking and routing using dijkstra algorithm for public transportation. *In 2015 1st International Conference on Wireless and Telematics (ICWT)*, pp. 1–4.

Inoue, M., Yamashita, T., & Nishida, T. (2019). Robot path planning by LSTM network under changing environment. In S. Bhatia, S. Tiwari, K. Mishra, & M. Trivedi (Eds.), *Advances in computer communication and computational sciences.* Springer.

Isele, D., Cosgun, A., Subramanian, K., Ffjimura, K. (2017). Navigating occluded intersections with autonomous vehicles using deep reinforcement learning. arXiv, arXiv:1705.01196 [cs.AI].

Jeon, J. H., Cowlagi, R. V., Peter, S. C., et al. (2013). Optimal motion planning with the half-car dynamical model for autonomous high-speed driving. *American Control Conference (ACC)*, pp. 188–193.

Ji, C., Lu, X., & Zhang, W. (2020). A biobjective optimization model for expert opinions aggregation and its application in group decision making. *IEEE Systems Journal, 15*(2), 2834–2844.

Jorgensen, T., Tama, A. (2019). Harnessing reinforcement learning for neural motion planning. arXiv, arXiv:1906.00214 [cs.RO].

Kakade, S., & Langford, J. (2002). Approximately optimal approximate reinforcement learning. *ICML, 2*, 267–274.

Kalos, M. H., & Whitlock, P. A. (2008). Monte Carlo methods. *Wiley-VCH.* ISBN 978-3-527-40760-6.

Kavraki, L. E., Svestka, P., Latombe, J. C., & Overmars, M. H. (2002). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation, 12*(4), 566–580.

Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research, 5*(1), 90–98.

Kim, B., Kaelbling, L. P., & Lozano-Perez, T. (2019). Adversarial actor-critic method for task and motion planning problems using planning experience. *AAAI Conference on Artificial Intelligence (AAAI), 33*(01), 8017–8024.

Kinnunen, T., Sidoroff, I., Tuononen, M., & Fränti, P. (2011). Comparison of clustering methods: A case study of text-independent speaker modeling. *Pattern Recognition Letters, 32*(13), 1604–1617.

Konda, V. R., Tsitsiklis, J. N. (2001). Actor-critic algorithms. *Society for Industrial and Applied Mathematics*, vol 42.

Krogh, B. (1984). A generalized potential field approach to obstacle avoidance control. *Proceedings of SME Conference on Robotics Research: The Next Five Years and Beyond*, Bethlehem, PA.

LaValle, S. M., & Kuffner, J. J. (1999). Randomized kinodynamic planning. *The International Journal of Robotics Research, 20*(5), 378–400.

Lecun, Y., Bottou, L., Bengio, Y., & Patrick, H. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE, 86*(11), 2278–2324.

Lei, X., Zhang, Z., & Dong, P. (2018). Dynamic path planning of unknown environment based on deep reinforcement learning. *Journal of Robotics, 2018*, 1–10.

Likhachev, M., Ferguson, D., Gordon, G., Stentz, A., & Thrun, S. (2008). Anytime search in dynamic graphs. *Artificial Intelligence, 172*(14), 1613–1643.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D. (2019). Continuous control with deep reinforcement learning. arXiv:1509.02971 [cs.LG].

Long, P., Fan, T., Liao, X., Liu, W., Zhang, H., & Pan, J. (2018). Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning. , arXiv:1709.10082v3 [cs.RO].

Mariescu, I. R., & Franti, P. (2018). Cell Net: Inferring road networks from GPS trajectories. *ACM Transactions on Spatial Algorithms and Systems, 4*(3), 1–22.

Masoud, A. A. (2007). Decentralized self-organizing potential field-based control for individually motivated mobile agents in a cluttered environment: A vector-harmonic potential field approach. *IEEE Transactions on Systems, Man, and Cybernetics-Part a: Systems and Humans, 37*(3), 372–390.

Meyes, R., Tercan, H., Roggendorf, S., Thomas, T., Christian, B., Markus, O., Christian, B., Sabina, J., Tobias, M. (2017). Motion planning for industrial robots using reinforcement learning. *In 50th CIRP Conference on Manufacturing Systems*, 63:107–112.

Meystel, A. (1990). Knowledge based nested hierarchical control. *Advances in Automation and Robotics, 2*, 63–152.

Minguez, J., Lamiraux, F., & Laumond, J. P. (2008). *Motion planning and obstacle avoidance*. Springer International Publishing.

Mnih, V., Kavukcuoglu, K., Silver, D., et al. (2013). Playing atari with deep reinforcement learning. arXiv, arXiv:1312.5602 [cs.LG].

Mnih, V., Kavukcuoglu, K., Silver, D., et al. (2015). Human-level control through deep reinforcement learning. *Nature, 518*, 529–533.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. arXiv, arXiv:1602.01783 [cs.LG].

Montemerlo, M., Becker, J., Bhat, S., et al. (2008). Junior: The stanford entry in the urban challenge. *Journal of Field Robotics, 25*(9), 569–597.

Munos, R., Stepleton, T., Harutyunyan, A., Bellemare, M. G. (2016). Safe and efficient off-policy reinforcement learning. arXiv, arXiv:1606.02647 [cs.LG].

Murphy, R. R. (2000). *Introduction to AI robotics*. MIT press.

Oh, J., Guo, Y., Singh, S. & Lee, H. (2018). Self-imitation learning. arXiv **2018**, arXiv:1806.05635v1 [cs.LG].

Panov, A. I., Yakovlev, K. S., & Suvorov, R. (2018). Grid path planning with deep reinforcement learning: Preliminary results. *Procedia Computer Science, 123*, 347–353.

Paxton, C., Raman, V., Hager, G. D., & Kobilarov, M. (2017). Combining neural networks and tree search for task and motion planning in challenging environments. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vancouver, BC, pp. 6059–6066.

Qureshi, A. H., Simeonov, A., Bency, M. J., Yip, M. C. (2018). Motion planning networks, arXiv:1806.05767 [cs.RO].

Reeds, J. A., & Shepp, L. A. (1990). Optimal paths for a car that goes both forward and backward. *Pacific Journal of Math, 145*(2), 367–393.

Rummery, G. A., & Niranjan, M. (1994). *On-line Q-learning using connectionist systems*. University of Cambridge.

Salemi, B., Moll, M., & Shen, W. M. (2006). SUPERBOT: A deployable multi–functional and modular self–reconfigurable robotic system. *Proceeding of IEEE/RSJ International Conference on Intelligent and Robots System*, pp. 3636–3641.

Schaul, T., Quan, J., Antonoglou, I., Silver, D. (2016). Prioritized experience replay. *International Conference on Learning Representations (ICLR)*.

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., & Abbeel, P. (2017). Trust region policy optimization. arXiv:1502.05477v5 [cs.LG].

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O. (2017). Proximal policy optimization algorithms. arXiv:1707.06347v2 [cs.LG].

Silver, D., Huang, A., Maddison, C. J., et al. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature, 529*(7587), 484–489.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., Riedmiller, M. (2014). Deterministic policy gradient algorithms. *In Proceedings of the 31st International Conference on International Conference on Machine Learning*, vol. 32. pp 387–395.

Smart, W. D., Kaelbling, L. P. (2002). Effective reinforcement learning for mobile robots. *IEEE International Conference on Robotics and Automation*, vol 4.

Stentz, A. (1994). Optimal and efficient path planning for partially-known environments. *Robotics and Automation*, pp. 203–220.

Sui, Z., Pu, Z., Yi, J., Tian, X. (2018). Path Planning of multiagent constrained formation through deep reinforcement Learning. *2018 International Joint Conference on Neural Networks (IJCNN)*.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. MIT Press.

Sutton, R., Mcallester, D. A., Singh, S., Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. *In Proceedings of the 12th International Conference on Neural Information Processing Systems*, MIT Press, Cambridge, MA, USA, pp 1057–1063.

Sutton, R. S., Maei, H. R., Precup, D., et al. (2009). Fast gradient-descent methods for temporal-difference learning with linear function approximation. *In 26th International Conference on Machine Learning*, Montreal, Canada.

Tobaruela, J. A. (2012). Reactive and path-planning methods for mobile robot navigation. PhD dissertation, Universitat de les Illes Balears. http://hdl.handle.net/11201/151880.

Tsitsiklis, J. N. (2003). On the convergence of optimistic policy iteration. *Journal of Machine Learning Research, 3*(1), 59–72.

Van den Berg, J., Lin, M., & Manocha, D. (2008). Reciprocal velocity obstacles for real-time multi-agent navigation. *IEEE International Conference on Robotics and Automation, 2008*, 1928–1935.

Van Den Berg, J., Guy, S. J., Lin, M., Manocha, D. (2011). Reciprocal n-body collision avoidance. *Robotics research,* **2011**, pp. 3–19.

Wang, L. (2005). On the euclidean distance of image. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 27*(8), 1334–1339.

Wang, F., Qian, Z., Yan, Z., Yuan, C., & Zhang, W. (2020). A novel resilient robot: Kinematic analysis and experimentation. *IEEE Access, 8*, 2885–2892.

Wang, Z., Freitas, N., Lanctot, M. (2015). Dueling network architectures for deep reinforcement learning. arXiv:1511.06581 [cs.LG].

Weston, J., Watkins, C. (1998). Multi-class support vector machines. Technical report, Department of computer science, Royal Holloway, university of London, May 20.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning, 8*, 229–256.

Xu, W., Wei, J., Dolan, J. M., Zhao, H., Zha, H. (2012). A real-time motion planner with trajectory optimization for autonomous vehicles. *IEEE International Conference on Robotics and Automation*, pp. 2061–2067.

Zhang, W. J., Wang, J. W., & Lin, Y. (2019). Integrated design and operation management for enterprise systems. *Enterprise Information Systems, 13*(4), 424–429.

Zhang, J. (2019) Gradient descent based optimization algorithms for deep learning models training, arXiv:1903.03614v1 [cs.LG].

Ziegler, J., Stiller, C. (2009). Spatiotemporal state lattices for fast trajectory planning in dynamic on-road driving scenarios. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1879–1884.

Terms and Conditions

Springer Nature journal content, brought to you courtesy of Springer Nature Customer Service Center GmbH (“Springer Nature”).
Springer Nature supports a reasonable amount of sharing of research papers by authors, subscribers and authorised users (“Users”), for small-scale personal, non-commercial use provided that all copyright, trade and service marks and other proprietary notices are maintained. By accessing, sharing, receiving or otherwise using the Springer Nature journal content you agree to these terms of use (“Terms”). For these purposes, Springer Nature considers academic use (by researchers and students) to be non-commercial.
These Terms are supplementary and will apply in addition to any applicable website terms and conditions, a relevant site licence or a personal subscription. These Terms will prevail over any conflict or ambiguity with regards to the relevant terms, a site licence or a personal subscription (to the extent of the conflict or ambiguity only). For Creative Commons-licensed articles, the terms of the Creative Commons license used will apply.
We collect and use personal data to provide access to the Springer Nature journal content. We may also use these personal data internally within ResearchGate and Springer Nature and as agreed share it, in an anonymised way, for purposes of tracking, analysis and reporting. We will not otherwise disclose your personal data outside the ResearchGate or the Springer Nature group of companies unless we have your permission as detailed in the Privacy Policy.
While Users may use the Springer Nature journal content for small scale, personal non-commercial use, it is important to note that Users may not:

1. use such content for the purpose of providing other users with access on a regular or large scale basis or as a means to circumvent access control;
2. use such content where to do so would be considered a criminal or statutory offence in any jurisdiction, or gives rise to civil liability, or is otherwise unlawful;
3. falsely or misleadingly imply or suggest endorsement, approval , sponsorship, or association unless explicitly agreed to by Springer Nature in writing;
4. use bots or other automated methods to access the content or redirect messages
5. override any security feature or exclusionary protocol; or
6. share the content in order to create substitute for Springer Nature products or services or a systematic database of Springer Nature journal content.

In line with the restriction against commercial use, Springer Nature does not permit the creation of a product or service that creates revenue, royalties, rent or income from our content or its inclusion as part of a paid for service or for other commercial gain. Springer Nature journal content cannot be used for inter-library loans and librarians may not upload Springer Nature journal content on a large scale into their, or any other, institutional repository.
These terms of use are reviewed regularly and may be amended at any time. Springer Nature is not obligated to publish any information or content on this website and may remove it or features or functionality at our sole discretion, at any time with or without notice. Springer Nature may revoke this licence to you at any time and remove access to any copies of the Springer Nature journal content which have been saved.
To the fullest extent permitted by law, Springer Nature makes no warranties, representations or guarantees to Users, either express or implied with respect to the Springer nature journal content and all parties disclaim and waive any implied warranties or warranties imposed by law, including merchantability or fitness for any particular purpose.
Please note that these rights do not automatically extend to content, data or other material published by Springer Nature that may be licensed from third parties.
If you would like to use or distribute our Springer Nature journal content to a wider audience or on a regular basis or in any other manner not expressly permitted by these Terms, please contact Springer Nature at

onlineservice@springernature.com