DEEP REINFORCEMENT LEARNING WITH CONSENSUS FOR MANIPULATORS

A thesis submitted to the University of Manchester
for the degree of Doctor of Philosophy
in the Faculty of Science and Engineering

2023

Wenxing Liu
School of Engineering
Department of Electrical and Electronic Engineering

Contents

Word count 34450 words

List of Figures

The University of Manchester

Wenxing Liu
Doctor of Philosophy
Deep Reinforcement Learning with Consensus for Manipulators
May 28, 2023

With the development of industrialization, the working environment of robotics gradually becomes complex, diverse, and fast. Most manipulators at present are still designed for simple action repetition, which means that the working environment is determined and the target should be relatively fixed. Therefore, they lack the ability to perceive the surrounding environment. The main purpose of this thesis is to develop consensus-based training and deep reinforcement learning methods that enable robot arms to interact with the environment autonomously.

First of all, a model-free off-policy actor-critic based deep reinforcement learning method is proposed to solve the classical path planning problem of a UR5 robot arm. The proposed method not only guarantees that the joint angle of the UR5 robotic arm lies within the allowable range each time when it reaches the random target point, but also ensures that the joint angle of the UR5 robotic arm is always within the allowable range during the entire episode of training.

Moreover, a self-supervised vision-based deep reinforcement learning method that allows robots to pick and place objects effectively and efficiently when directly transferring a training model from simulation to the real world is demonstrated. A height-sensitive action policy is specially designed for the proposed method to deal with crowded and stacked objects in challenging environments. The training model with the proposed approach can be applied directly to a real suction task without any fine-tuning from the real world while maintaining a high suction success rate. It is also validated that the training model can be deployed to suction novel objects in a real experiment with a suction success rate of 90% without any real-world fine-tuning.

Additionally, an algorithm that combines actor-critic based off-policy method with consensus-based distributed training is proposed to deal with multi-agent deep reinforcement learning problems. Specially, a convergence analysis of a consensus algorithm for a type of nonlinear systems with a Lyapunov method is developed, and this result is used to analyse the convergence properties of the actor and the critic training parameters. To validate the implementation of the proposed algorithm, a multi-agent training framework is proposed to train each UR5 robot arm to reach the random target position. Experiments are provided to demonstrate the effectiveness and feasibility of the proposed algorithm.

Finally, a Consensus-based Sim-and-Real deep reinforcement learning algorithm is developed for manipulator pick-and-place tasks. Agents are trained in both simulators and the real environment simultaneously to get the optimal policies for both sim-and-real worlds. The proposed algorithm saves required training time and shows comparable performance in both sim-and-real worlds.

Declaration

No portion of the work referred to in the thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright Statement

i. The author of this thesis (including any appendices and/or schedules to this thesis) owns certain copyright or related rights in it (the "Copyright") and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.

ii. Copies of this thesis, either in full or in extracts and whether in hard or electronic copy, may be made **only** in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has from time to time. This page must form part of any such copies made.

iii. The ownership of certain Copyright, patents, designs, trade marks and other intellectual property (the "Intellectual Property") and any reproductions of copyright works in the thesis, for example graphs and tables ("Reproductions"), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.

iv. Further information on the conditions under which disclosure, publication and commercialisation of this thesis, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy, in any relevant Thesis restriction declarations deposited in the University Library, and the University Library's regulations.

Abbreviations

A3C Asynchronous Advantage Actor-critic

CAD Computer Aided Design

CCD Cyclic-Coordinate Descent

CSAR Consensus-based Sim-and-Real

DDPG Deep Deterministic Policy Gradient

DRL Deep Reinforcement Learning

FABRIK Forward and Backward Reaching Inverse Kinematics

FK Forward Kinematics

GAN Generative Adversarial Network

IK Inverse Kinematics

MAS Multi-agent System

NN Neural Network

OMPL Open Motion Planning Library

PPO Proximal Policy Optimization

PRM Probabilistic Roadmaps Method

RGB-D Red Green Blue-Depth

RL Reinforcement Learning

ROS Robot Operating System

RRT Rapidly-exploring Random Tree

SAC Soft Actor-critic

SCP Sequential Convex Programming

SQP Sequential Quadratic Programming

SARSA State-Action-Reward-State-Action

TD3 Twin Delayed Deep Deterministic Policy Gradient

TRPO Trust Region Policy Optimization

UR5 Universal Robot 5

Symbols

$\mathbb{R}$ Space of real numbers

$\mathbb{R}^n$ Space of real vectors of dimension n

$\mathbb{R}^{m\times n}$ Space of real matrices of size $m \times n$

$\mathbf{0}_n$ A column vector of size n with all entries equal to zero

$\mathbf{0}_{m\times n}$ A $m \times n$ matrix with all zeros

$\mathbf{1}_n$ A column vector of size n with all entries equal to one

$\mathbf{1}_{m\times n}$ A $m \times n$ matrix with all ones

A Adjacency matrix

D In-degree matrix

G Graph

$\mathcal{A}$ Action space

$\mathcal{S}$ State space

$\mathbf{diag}\{a_1, \cdots, a_n\}$ A diagonal matrix with diagonal entries a_1 to a_n

$\mathbf{max}\{\cdot\}$ Maximum elements

$\mathbf{min}\{\cdot\}$ Minimum elements

$\|\cdot\|$ Standard L_2 Euclidean norm of a vector

X^T Transpose of matrix X

X^{-1} Inverse of matrix X

I_n $n \times n$ Identity matrix

$\otimes$ Kronecker product

$f : \mathbb{A} \to \mathbb{B}$ Function f with domain $\mathbb{A}$ and range $\mathbb{B}$

J^+ Pseudo Inverse of J

π Reinforcement learning policy

$V^\pi(s)$ Reinforcement learning policy function

$Q^\pi(s, a)$ Reinforcement learning value function

ξ Reinforcement learning critic training parameter

η Reinforcement learning actor training parameter

Publications

[1] **W. Liu**, H. Niu, I. Jang, G. Herrmann and J. Carrasco, "Distributed Neural Networks Training for Robotic Manipulation With Consensus Algorithm," in IEEE Transactions on Neural Networks and Learning Systems, 2022,
doi: 10.1109/TNNLS.2022.3191021.

[2] **W. Liu**, H. Niu, M. N. Mahyuddin, G. Herrmann and J. Carrasco, "A Model-free Deep Reinforcement Learning Approach for Robotic Manipulators Path Planning," 2021 21st International Conference on Control, Automation and Systems (ICCAS), 2021, pp. 512-517, doi: 10.23919/ICCAS52745.2021.9649802.

[3] **W. Liu**, H. Niu, W. Pan, G. Herrmann and J. Carrasco, "Sim-and-Real Reinforcement Learning for Manipulation: A Consensus-based Approach," accepted by IEEE Conference on Robotics and Automation (ICRA 2023).

[4] **W. Liu**, H. Niu, R. Skilton and J. Carrasco, "Deep Reinforcement Learning with Manipulators for Pick-and-place," In progress.

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisor Dr. Joaquin Carrasco, for his professional guidance and continuous support during my PhD study. Dr. Carrasco is an expert in control engineering and has a very rigorous attitude towards academic research. Under his influence, I also gradually become a very rigorous person. Dr. Carrasco is also very patient and friendly. We have meetings every week and he always provides me with his feedback as soon as possible. We also hold weekly workshops on Deep Reinforcement Learning to discuss new ideas and questions. It is my honour to be a PhD student under his supervision.

I am extremely grateful to my cosupervisor, Prof. Guido Herrmann, for his professional academic guidance during the past three years. Prof. Herrmann is very wise and knowledgeable. I feel more confident to submit my papers after Prof. Herrmann has reviewed them.

I would also like to thank Dr. Wei Pan for his collaboration during my PhD study. I sincerely thank Dr. Farshad Arvin for his trust in my abilities. Thank Dr. Hanlin Niu and Dr. Junyan Hu for their continuous help in the past three years.

I owe my deepest appreciation to my family. I would like to show my deep gratitude to my mother, Yu Zhang, and my father, Xiaozhou Liu, for their continued unconditional support. I am deeply indebted to my grandmother, Xinhua Guo, for her accompany and encouragement all the time.

Last but not least, words cannot express my gratitude to my grandfather, Zhongtan Liu, who gave me endless love and guidance over the past 24 years. His greatest wish

in life is to see me succeed in my studies. I could not have undertaken this PhD journey without his trust and belief. He used to tell me the importance of persistent studying when I was a child, which kept my motivation and spirits high during my entire learning progress. My biggest regret is that he would not be able to see my achievement and graduation in person. I miss him so much and he will always be in my heart.

This thesis is dedicated to my grandfather,
Zhongtan Liu,
in loving memory.

You always believed in my ability to be successful in the academic arena.
Your belief in me has made this thesis possible.

Chapter 1

Introduction

1.1 Background

Robots are complex machines created with the purpose of performing tasks independently or with minimal human input. With the rapid development of science and technology, robots can be widely applied in a variety of different tasks such as path planning [4; 5], obstacle avoidance [6; 7], pick-and-place objects [8; 9], goal reaching [10; 11], etc. Robots come in different sizes and shapes, and they can be used for various applications. Small and straightforward robots are typically used in factories to perform repetitive tasks, while more complex systems are designed to function in unpredictable and dynamic environments. The use of robots is becoming increasingly popular in fields such as manufacturing [12], healthcare [13], exploration [14; 15] and assembly [16]. With advancements in technology and ongoing research, robots are becoming more versatile and adaptable, allowing them to handle a growing range of complex and demanding tasks.

One of the most widespread applications of robotics is the robotic manipulator [17]. A robot manipulator [18] is a type of robot that is specifically designed to manipulate objects in its environment. It achieves this by using mechanical arms and end-effectors, such as grippers. The manipulator usually consists of a series of articulated arms or links that are connected by joints or hinges, enabling the manipulator to move in various degrees of freedom [19]. The gripper is typically attached to the last link of the manipulator, and it is used to grasp and manipulate objects. Robot manipulators

are widely used in various industries such as manufacturing [20], assembly [21], and other industrial settings [22]. They are particularly useful for performing repetitive tasks with high precision and efficiency. Robot arms have a high technical value and a wide range of practical applications which play an irreplaceable role in production mode. The dynamic analysis of the robot arm has always been one of the hotspots in scientific research.

1.1.1 Pick and Place

Pick-and-place is a common task performed by robots that involve picking up an object and moving it to a different location [23; 24]. This task has a wide range of applications, from industrial settings such as assembly lines [25], packaging [26], and material handling [27], to everyday tasks like sorting [28] and organizing [29; 30]. To perform a pick-and-place task, the robot arm must be equipped with a gripper that can securely grasp the object, and it must be able to locate, approach, and move the object to the desired location with precision [31; 32]. The success rate of the pick-and-place task depends on the accuracy of the robot arm and its ability to identify objects of various shapes, sizes, and materials [33; 34]. To execute a successful pick-and-place task, the robot arm needs to have a perception system that can detect and locate objects in the environment reliably [35]. Once the object has been located, the robot needs to plan its motion to grasp the object using various techniques including Inverse Kinematics (IK) [36; 37], path planning [38], or trajectory planning [39; 40]. Additionally, obstacles must be taken into consideration when planning a path, and movements should be adjusted accordingly to avoid collisions [41; 42]. In summary, pick-and-place is an important task in both simulated and real-world applications, and it relies on advanced robotic techniques like perception, motion planning, and manipulation to perform with remarkable precision and dependability [43; 44].

Recently, there has been a growing interest in picking and placing objects [45]. For instance, an industrial robot system with several stationary cameras was mentioned in [46] to ensure safe human-robot cooperation. In [47], an application of visual serving to a 4 degree-of-freedom robot manipulator was proposed to pick and place a target using the edge detection method as visual input. A remote-controlled mobile robot

was developed and designed in [48] to deal with the pick-and-place task. In [49], the software development for a vision-based pick-and-place robot was presented to provide the computational intelligence required for its operation. In order to eliminate human intervention or error and work more precisely, a pick-and-place robot using Robo-Arduino was developed in [50] for any pick-and-place functions. A pick-and-place robot which offered sensing, control and manufacturing assistance was present in [51], which improved productivity and reduced the risk of injury because of repetitive tasks. Nevertheless, in the works above, the authors focus on model-based grasping with specific robotic manipulators, which may be regarded as a limitation. The pick-and-place process requires a robot arm and end effector to work in coordination with precision [52]. As the robot moves the object, it must be able to adjust its grip to keep the object securely in place [53; 54]. Upon reaching its destination, the robot must accurately position and orient the object [55; 56]. Pick-and-place is an important task in both simulated and real-world applications, and it relies on advanced robotic techniques like perception, motion planning, and manipulation [57]. Highly efficient and sophisticated pick-and-place systems can handle a wide variety of objects and tasks with remarkable precision and dependability [58; 59].

1.1.2 Motion Planning

Motion planning [60] is a critical process in the field of robotics that involves generating a safe and feasible path or trajectory for a robot or autonomous system to move from one state to another while satisfying specific constraints and objectives. The aim of motion planning is to enable the robot to perform its intended task, such as transporting an object or moving to a particular location [61]. Since the development of effective and robust motion planning algorithms is crucial for the widespread implementation of automation technologies, researchers have been actively investigating this field. To tackle the various challenges involved in motion planning, several different techniques and algorithms have been developed, including optimization-based techniques [62], heuristic-based techniques [63], and sampling-based techniques [64]. The contributions of these techniques have been vital in achieving safe and efficient motion planning in robotics. The early work on motion planning [65] provided a foundation for further research in this field.

Optimization-based approaches are a particular type of motion planning algorithm that aims to find a suitable path for a robot by formulating the problem as an optimization problem [66; 67]. These algorithms aim to find a path for the robot by formulating the problem as an optimization problem, which involves identifying the best path that either maximizes or minimizes a specific objective function while also meeting certain constraints [68]. This approach has proven to be successful in various scenarios, leading to the development of several advanced algorithms that can tackle the complex optimization problems that arise in motion planning. The Covariant Hamiltonian Optimization for Motion Planning (CHOMP) is an optimization-based motion planning algorithm presented in [69]. It aims to generate a collision-free trajectory for a robotic system by minimizing a cost function that takes into account both task objectives and obstacle avoidance. Sequential Convex Programming (SCP) [70] is another optimization-based approach for solving motion planning problems, which is particularly useful for systems with non-linear constraints. The goal of SCP is to find an optimal solution to a problem by iteratively solving a sequence of convex sub-problems, where each sub-problem is a simplified version of the original problem that is easier to solve [71]. Optimization-based approaches have gained popularity in motion planning due to their capability to handle problems with intricate constraints and locate solutions that are globally optimal [72]. However, these approaches may not be ideal for dynamic environments where the surroundings of robots undergo frequent change. This is because solving the optimization problem may need to be performed repeatedly to account for the changing environment, which can be time-consuming and computationally intensive.

Heuristic-based approaches have been widely used in motion planning algorithms, leveraging heuristics or rules of thumb to guide the search for a solution [73]. These algorithms aim to identify feasible and safe paths for a robot to traverse while satisfying constraints and reaching a desired goal [74]. The fundamental idea behind heuristic-based approaches is to employ a heuristic function that can estimate the distance or cost between two points in space [75]. These approaches are popular in motion planning due to their efficiency, ease of implementation, and efficacy in low-dimensional

spaces. One of the most well-known heuristic-based approaches in motion planning is the A* algorithm [76]. This algorithm uses a heuristic function to guide the search for the shortest path between two points in a graph. Another popular heuristic-based approach is the D* algorithm [77], which is designed to update a precomputed path in the presence of dynamic obstacles. Cyclic-Coordinate Descent (CCD) algorithm [78; 79] is another heuristic-based approach to performing motion planning. This method is easy to implement and computationally fast, but may need longer iterations to reach the target point. Besides, this method is more suitable for solving snake robots rather than the Universal Robot 5 (UR5) robot arm. Heuristic-based approaches have been widely used in motion planning due to several strengths, including their ability to handle problems with well-defined goals and constraints, their simplicity and ease of implementation, and their potential for efficiency in low-dimensional spaces. Nevertheless, heuristic-based approaches may struggle with complex constraints. Besides, achieving desirable performance requires careful tuning of the heuristic function.

Sampling-based approaches are a class of motion planning algorithms that construct a graph or a tree of feasible paths through random sampling of points in the robot's configuration space [80; 81]. These approaches have become increasingly popular due to their ability to handle motion planning problems with high-dimensional state spaces and complex constraints. In addition, sampling-based algorithms are often computationally efficient, making them an attractive option for real-time motion planning [82]. Kavraki [83] illustrated the Probabilistic Roadmaps method (PRM) which connects the sampled points based on a roadmap to manage obstacle avoidance motion planning. In [84], a geometry-based, multilayered synergistic approach was developed to solve motion planning problems for mobile robots involving temporal goals. How to effectively combine a sampling-based method with the PRM was introduced in [85] to deal with the problem of single query motion planning. Sampling-based approaches are able to find feasible paths in a wide range of problems, making them versatile solutions for many different applications. However, these algorithms may face challenges in problems with narrow passages or obstacles with small gaps, which can make it difficult to construct a valid path. Additionally, long time fine-tuning may be necessary to achieve optimal performance in more complex scenarios.

1.1.3 Reinforcement Learning

Reinforcement Learning (RL) [86] is a subfield of machine learning that involves training agents to make decisions and take actions within an environment by receiving feedback in the form of rewards or penalties. RL enables intelligent agents to learn and adapt to complex environments through experience and feedback [87]. Traditional rule-based approaches [88; 89] to problem-solving can be limited by the need for extensive human expertise and may not be suitable for uncertain and complex environments. In contrast, RL allows agents to learn from experience and optimize their decision-making policies based on feedback from the environment, without requiring explicit instructions or prior knowledge from a human [90]. The so-called RL refers to the learning of the mapping relationship from the environment state to the action space, so as to maximize the cumulative reward the system obtains from the environment [91]. The goal of RL is to develop intelligent agents that can learn to perform tasks through trial and error without explicit instructions from a human. The concept of reward in RL was first introduced in [92] to train the machine to make it understand how to play chess games. Consequently, the machine is able to make better decisions after several training steps. Sutton [93] illustrated a model-free algorithm called the Temporal Difference error algorithm which could be used in both on-line RL and off-line RL. The update of a model state depends on the value function of the next state and the reward to reach the next state. As a result, RL has been widely applied in the field of model control and prediction.

RL has become a very active branch in the field of machine learning research. RL has been successfully applied in a wide range of fields, including robotics [94; 95], game playing [96], natural language processing [97; 98], finance [99; 100], etc. One strength of applying RL to robots is that it does not need an accurate dynamic model of robots [101], [102]. The agent is not told what to do in the learning process, but must obtain a certain strategy by interacting with the environment, which can guide the agent to obtain the maximum reward from the environment [103; 104]. In many scenarios, the current behaviour will affect not only the immediate rewards but also the subsequent actions and final rewards. Lange [105] demonstrated an algorithm called Deep Fitted

Q learning which could be applied in vehicle control. RL has also been utilized in creating game-playing agents that can outperform world champions in games like Go and Chess [106]. In 2017, Google DeepMind [107] published their latest result in Nature. A new artificial intelligence system called Alpha Zero is able to beat Alpha Go after only three days of training. Compared to Alpha Go which mainly uses deep learning, Alpha Zero is based on RL which does not rely on any chess manual provided by humans or any human chess-playing experience. The way how Alpha Zero acquires chess playing skills is only by competing against itself. RL is an important area of research in artificial intelligence, and it continues to advance our understanding of how intelligent agents can be designed and trained to achieve sophisticated tasks [108].

When dealing with random and unpredictable tasks, RL methods [109; 110] have shown great benefits. RL methods enable agents to learn the mapping relationship between the environment and their actions [111; 112]. Therefore, each agent can adjust its action on the basis of the feedback signal in order to maximize the cumulative reward from the environment [113; 114]. Off-policy RL is independent of the agent's actions, which means that it can figure out the optimal policy regardless of the actions [115]. Recently, many RL methods have been investigated to perform multi-agent tasks. In [116], Tan claimed that agents could share instantaneous information such as actions, rewards, etc. Therefore, each agent is able to benefit from the information of neighbours. An actor–critic framework for Multi-agent Systems (MASs) was developed in [117] to deal with the input of a system with consensus control. In [3], a single-agent actor-critic algorithm was combined with a consensus-like algorithm to improve the convergence speed of a distributed actor-critic algorithm. One major challenge of RL is the requirement for large amounts of data to train the agent effectively. In addition, designing appropriate reward functions that accurately reflect the desired behaviour is not always straightforward and may require careful consideration. Furthermore, RL can be computationally expensive, requiring significant computational resources to train and optimize the agent's policies [118].

With the rapid development of artificial intelligence, great achievements have been made in the research and application of RL. One of the core technologies used is Deep

Reinforcement Learning (DRL) [119]. DRL is a specialized field within RL that leverages the power of deep Neural Networks (NNs) to enable agents to learn and improve their decision-making abilities in complex environments [120]. DRL algorithms use deep NNs to approximate either the value function or policy function of an agent in an RL problem, allowing them to handle high-dimensional input spaces and learn representations that capture the underlying structure of the input data [121; 122]. This capability enables DRL algorithms to learn directly from raw input data such as images or audio and extract relevant features and patterns that are useful for decision-making [123]. DRL enables end-to-end behavioural decision-making by learning decisions directly from features from raw information [124]. In recent years, DRL has developed vigorously to provide new ideas for autonomous learning and control of robotic arms. In [125], a robust end-to-end closed-loop grasping DRL model was trained using grasping demonstrations by people, which was able to grasp novel or moving objects in various scenarios. In [126], an end-to-end DRL approach was proposed for a UR5 arm to jointly learn pushing and grasping objects. A grasping and throwing system called TossingBot was developed in [127] to make a UR5 robot arm throw arbitrary objects into boxes via DRL, which could also generalize to novel objects. A DRL algorithm that mapped the elementary movement to the meta-parameters of its representation was introduced in [128] for hitting movements of robot arms such as playing table tennis. In [129], a Form2Fit system was designed using DRL to learn assembling a wide variety of objects and kits. When using DRL to train models, one of the most challenging tasks is dealing with a large number of failed experiment samples. Hindsight Experience Replay [130] is a technique that has been developed for DRL, aimed at improving the sample efficiency and effectiveness of DRL algorithms. It involves reframing failed experiences as successes, thereby providing agents with additional learning opportunities. This technique has proven to be particularly useful in tasks with sparse rewards or where the desired outcome may be difficult to achieve. While Hindsight Experience Replay has several strengths, including the ability to improve sample efficiency, it also has limitations. For instance, careful goal definition is necessary to avoid overfitting a specific goal.

Training a DRL model directly in a real environment is difficult since it relies on several conditions. One of the major challenges of DRL is the need for large amounts of data [131], which can be time-consuming and expensive to collect. Another challenge is the difficulty in training and optimizing deep NNs [132], which require careful parameter tuning and can be computationally expensive. Additionally, DRL algorithms may require significant computational resources to train and optimize, making it difficult to scale up to larger and more complex problems [133]. Collecting real-world training images will take a lot of time, which increases the training cost significantly [134]. Sim-to-real DRL is particularly useful when it is difficult or expensive to train robots in the real world due to safety concerns or limited access to hardware [135]. Therefore, an alternative method is training on simulated images and then adapting the features to real-world data [136]. Existing methods can be roughly divided into two categories: high-fidelity simulation and domain randomization.

High-fidelity simulation means adjusting the virtual environment through real data, thereby improving the suction success rate in the real environment. High-fidelity simulation is an approach in which a simulated environment is designed to closely mirror the real world, incorporating realistic graphics, physics simulations, and other details that accurately capture the dynamics of the real world [137]. This approach is particularly valuable when it is possible to create a simulation that accurately models the real world, allowing for a detailed and accurate exploration of the system. In [138], an end-to-end pipeline was proposed to generate realistic depth data from 3D models. By modelling vital real-world factors such as sensor noise and surface geometry accurately, the proposed framework was able to accomplish more realistic results than baseline methods. An approach to creating pixel-accurate semantic label maps for images extracted from modern computer games was developed in [139]. Data created with the proposed method could improve the performance of semantic segmentation models on real-world images, which reduced the requirement for expensive real-world labelling. In [140], the problem of transferring policies from simulation to the real world was solved by training on the distribution of simulated scenarios. A few real-world samples were used to adapt the simulation parameter distribution instead of tuning the randomization of simulations manually. By matching the policy behaviour

in both environments, the simulated policy could be successfully transferred to the real world [141].

The utilization of high-fidelity simulation has the potential to enhance the efficiency and effectiveness of robot learning, as the robot can leverage the model to plan and make decisions strategically [142]. By minimizing the discrepancy between the simulation and the real world, a complete system framework was proposed in [143] to optimize trajectories of a bipedal robot with a very small number of real-world experiments. In [144], a framework was developed for identifying the mechanical parameters of robots before real-world deployment. The simulation parameters were well approximated in order to match real-world trajectories. In [145], a new Grounded Action Transformation algorithm was fully implemented and evaluated using a high-fidelity simulator. However, additional models for simulator inverse dynamics and real-world forward dynamics were required in [145]. An iterative optimization framework was implemented in [146] to speed up robot learning with an imperfect simulator. After optimising the behaviour in the simulation, the resulting behaviour was tested on the real robot and the simulator was modified according to the real-world performance. By training in high-fidelity simulation, models can learn to perform well in the target environment, even if the target environment may not be available during training [147]. Nevertheless, humans were required in the loop to choose the best simulation training parameters, which might be regarded as a limitation.

Domain randomization [148] is randomly adjusting the shape, brightness and size of objects in the simulated environment in order to make the real-world experiment more robust and feasible. The main goal of domain randomization is to expose the robot to a diverse set of environments during training, with the hope that it will learn to generalize to real-world conditions that it has not previously encountered [148]. By training the robot in a variety of simulated environments, it will be better equipped to handle the unpredictability and variability of real-world scenarios, and thus improve its ability to perform tasks in the real world [149]. A simple technique for training models was developed in [150] on sim-to-real image transfer by randomizing rendering in the simulator. If the simulator had enough variability, the model might treat the

real world as just another variation. In [151], a pick-and-place task trajectory was computed in a simulator to collect a series of control velocities. In order to generalize to real-world images, domain randomisation was used to map observed images to velocities. In [152], in-hand manipulation was completely trained in simulation and performed in the real world. A large number of physical properties such as friction coefficients were randomized to perform sim-to-real transfer accurately. A CAD2RL method was proposed in [153] to perform collision-free indoor flight in the real world on the basis of 3D Computer Aided Design (CAD) models. By highly randomizing the rendering settings in simulation, the policy in simulation could generalise to the real world. The key strength of domain randomization is that it does not require a detailed model of the real-world environment, making it easier to apply in practice [154]. Nevertheless, the problem of large sample complexity happened in the works mentioned above, which may be viewed as a restriction.

When a DRL model is transferred from simulation to the real world, the adoption problem becomes challenging as real-world environments contain unpredictable disturbances [149]. Fine-tuning has been widely used to bridge the gap between simulated and real environments [155; 156; 157; 158]. However, fine-tuning usually takes a long time to perform parameter adaptation, which increases the experimental cost. Some recent works use only simulation but work well in the real world. For instance, with only simulation, a distance function was trained in [159] between the current pose and the nearest optimal pose. In [160], a grasp quality network was proposed to evaluate robust grasp configuration based on the antipodal grasping sampling method. The key idea of these two papers is to use depth data rather than Red Green Blue (RGB) images since depth images contain less information. Nevertheless, it is challenging for a depth camera to measure thin, dark colour objects because of their physical properties in the real world. Under this condition, performance cannot be guaranteed.

Sim-and-real training [161; 162] is a recent research topic compared with the sim-to-real training method. By training simulated and real environments at the same time with the DRL method, the mixed training model is able to benefit from both environments, which improves the performance in real-world experiments. A model that discovers

cross-domain relations with Generative Adversarial Networks (GANs) was mentioned in [163]. Unlike other methods, the authors investigated the relationship between real systems and simulations. The simulated data were utilized for learning a generalized perception system and the real data were used to learn the dynamics of the system. A novel domain adaptation approach for robot perception was developed in [164] to close the sim-and-real gap by finding common features of real and synthetic data. In [165], the difference between the simulation and the real world was diminished by applying models that made synthetic images more realistic. In [140], the agent's parameters in the simulation were updated to match the behaviour in the real world. In [166], a method was proposed to learn from both simulation and interaction with the real environment at the same time. With the aim of balancing less accurate but cheap samples in simulation and accurate but costly samples from the real world, a simulated environment was selected by an agent with probability and interacts with it. Simultaneously, the agent also chose real-world actions with probability from the replay butter. However, generating transitions is inevitable in the aforementioned works, which is less effective and efficient.

1.1.4 Consensus Control

Since it is difficult to use one agent to deal with the problem of competition and coordination among multiple decisions, it is necessary to extend to multi-agent DRL[167; 168]. A MAS refers to a group of autonomous agents that collaborate with each other and with the environment to achieve a shared objective [169]. Each agent has its own sensors and actuators, which allow it to sense its local environment and make decisions based on its observations and interactions with other agents. The agents can differ in terms of their capabilities, goals, and strategies [170]. In multi-agent DRL, each agent has its own NN that learns from its own experiences and the experiences of other agents in the system. These agents may have diverse perceptions of the environment and may have varying objectives or reward systems, but must coordinate their actions to achieve the common goal [116]. An interactive partially observable Markov decision process which considered each agent separately was introduced in [171] for multiple agents within a common environment. In [116], Q learning was extended straightforwardly to multiple agents. However, the performance of MASs was not satisfactory

when applying single-agent DRL algorithms directly to multiple agents [172]. The problems faced in DRL in complex task environments were large-scale information acquisition and exchange [173; 174]. A multi-agent bidirectionally coordinated network was proposed in [175] to coordinate multiple agents to defeat their enemies as a team in the StarCraft combat game. In [176], a novel counterfactual multi-agent policy was introduced to address the challenges of multi-agent information exchange with a centralized critic. In [177], an adaptation of actor-critic methods was developed to learn complex coordination policies successfully with centralized critics. Nevertheless, in the works mentioned above, centralized training is implemented in multi-agent DRL instead of decentralized training. The proposed systems may suffer from limited bandwidth when a large number of agents are taken into consideration.

For MASs, the communication channel between agents will introduce fundamental constraints. A key challenge in MAS communication is finding the right balance between information sharing and privacy [178]. While agents need to share information to coordinate their actions effectively, they also need to maintain privacy to safeguard their individual goals and strategies [179]. Achieving this balance is complicated by the fact that agents in a MAS may have varying levels of trust and may be competing against each other [180]. Another challenge is minimizing communication overhead and avoiding redundancy by selectively communicating relevant information [181]. In the simplest case, each agent performs its own local training and no information sharing happens between any agent [182]. Since each agent cannot learn from the training of other agents, the training time of this training method is lengthy. Another approach guarantees the communication between agents, but with a large communication bandwidth caused by real time information sharing [183]. Furthermore, because of the hardware and software limitations, processing the data communicated between agents is daunting [184]. The communication between different agents may cause privacy issues. In [185], an efficient method to solve the privacy violation problem based on weighted averages of the update vectors taken over a random subset of users was proposed. A distributed and privacy-preserving algorithm was developed in [186] for dealing with user-generated data streams.

The robustness and reliability of agents have aroused considerable interest in the research of MASs [187]. A pulse width modulation protocol was mentioned in [188] for the distributed consensus of MASs. A distributed multi-vehicle controller system was introduced in [189] to deal with coordinating platoon formation. In [190], a two-step distributed model predictive control strategy was developed to solve the cooperative vehicle platooning problem. Collaborative learning [191; 192] plays an important role in the field of robotics, where MASs can learn jointly from the environment.

Nowadays, consensus control has been widely used in the coordination and cooperation of MASs [193; 194]. Consensus control is a field of control theory that is concerned with achieving consensus or agreement among a group of agents in a distributed system [195]. The aim of consensus control is to develop control strategies that enable the agents in a network to converge to a common decision or a shared state [196; 197]. In nature, the phenomenon of consensus is very common, such as the migration of birds [198], the parade of fish [199], etc. These simple biological entities can complete relatively complex tasks through cooperation, thereby improving survival rates. Consensus control has numerous applications in various fields such as robotics [200; 201], swarm intelligence [202; 203], social networks [204; 205], distributed computing [206], etc. In robotics, it is used to coordinate the motion of multiple robots to achieve a common goal, such as exploring hazardous environments [207] or transporting heavy objects [208]. In swarm intelligence, consensus control can help to achieve coordinated behaviour among a large group of agents, such as a swarm of drones [209]. In social networks, consensus control can be utilized to analyze the spread of information and opinion among a group of individuals [210]. In distributed computing, consensus control plays a crucial role in reaching an agreement among a group of computing nodes on a shared decision or a computation result [211]. A distributed consensus problem was discussed in [212] for linear heterogeneous MASs with both bounded unknown control input and matching uncertainties. In [213], a linear consensus protocol was introduced to solve the consensus problem of dynamic MASs. In [214], observer-based controllers were designed for distributed consensus control of MASs under different denial-of-service attacks on different channels. A distributed feedback controller was proposed in [215] to handle cluster consensus problems for linear MASs under directed

graph topologies. An event-triggered control mechanism was developed in [216] to deal with the security consensus problem for time-varying MASs with parameter uncertainties and false data-injection attacks. In [217], novel finite-time adaptive control algorithms were introduced for leaderless nonlinear consensus control of MASs with parametric uncertainties. Consensus control can not only effectively reduce the complexity of MASs, but also greatly improve the working efficiency [218].

Achieving efficient and timely convergence to a desired consensus state while ensuring robustness and stability against disturbances and uncertainties is one of the main challenges in consensus control [219]. This requires a careful design of the control law that considers various factors, such as the dynamics of the agents, the communication topology, and the consensus objective [220]. Scalability is another critical consideration in consensus control algorithms, particularly in large-scale systems with a significant number of agents [221; 222]. Despite these challenges, consensus control has proven to be a powerful tool for achieving coordinated behaviour in distributed systems, with numerous applications in various fields [223].

Consensus control of MASs has become an emerging topic in the field of robotics. Different from centralized control, distributed consensus control has stronger robustness and better scalability, which solves the shortcoming of the whole system getting paralyzed due to a certain link failure [224]. A smooth time-varying controller was proposed in [225] to deal with leaderless consensus control problems in vehicles under communication delays. An adaptive distributed consensus tracking protocol was developed in [226] for a class of nonlinear mobile robots with mismatched uncertainties. In [227], a well-designed radial basis function NN was used to solve the problem of nonlinear time-delay consensus control for multiple collaborative manipulator systems. In [228], a NN-based consensus control framework was introduced for multiple robotic manipulator systems under both fixed and switching leader-follower communication topologies. However, in the aforementioned works, the authors only verify their theorems from a theoretical point of view. Real-world robot experiments are not considered to validate the feasibility of the proposed theorems.

In recent years, many adaptive [229; 230] and consensus-based distributed training methods [231] have been developed to perform tracking and multi-agent tasks. By implementing consensus control, multiple agents are able to exchange their information and coordinate their positions under the control of a distributed protocol. Ultimately, their respective agents will be kept in a consistent state and some specific tasks such as monitoring or rounding up can be accomplished. In [232], an event-triggered control algorithm was developed to solve the consensus problem of multiple single-link robot arms in a finite time. In [233], a framework to solve bearing-only collision-free formation of distributed multiple agents based on directed information flow over the formation graph was proposed. Consensus algorithms with a time-varying reference state were proposed in [234] for multi-vehicle formation control. Even though consensus-based distributed training has been extensively developed, an open question remains in handling completely random and unpredictable tasks with consensus-based distributed training.

1.2 Motivation

Although motion planning, DRL, and consensus control have gained significant attention in the field of robotics, several challenges and research gaps still need to be addressed. These limitations and gaps can potentially hinder the performance of these techniques in real-world applications. Addressing these limitations and research gaps is crucial for advancing the capabilities and practical use of these techniques in robotics applications.

The field of motion planning algorithms for robots faces challenges in handling complex environments and tasks while maintaining computational efficiency and adaptability. The demand for faster and more efficient algorithms capable of handling complex environments and tasks has been increasing [235]. Additionally, these algorithms must be robust enough to cope with minor changes in initial conditions and adaptable to changing environments and tasks [236].

One of the main challenges of DRL algorithms is improving their data efficiency, as

they typically require a large amount of data to learn robust policies [237]. Another challenge is ensuring their safety and reliability in safety-critical applications, which is crucial for their practical deployment [238]. Additionally, developing interpretable DRL algorithms that can explain the decision-making process of the learned policies is a research gap that can enhance the transparency and trustworthiness of these algorithms in real-world applications.

Consensus control algorithms are facing challenges related to scaling up to large-scale MASs with minimal communication overhead and computational complexity [239; 240]. Communication networks between agents must be reliable and robust to ensure the performance of consensus control algorithms [241; 242]. Additionally, developing algorithms that can handle heterogeneous agents with different dynamics and objectives is a challenging problem that requires further research.

Addressing these limitations and research gaps is critical to advancing the field of robotics and enabling robots to operate effectively and safely in real-world environments. Motivated by the aforementioned works and challenges, the aim of this thesis is to develop a theory that combines the strengths of DRL with consensus control and implements it in real-world applications to improve the performance and scalability of robotic systems. The specific objectives of this thesis include:

- Developing new algorithms that integrate the strengths of DRL with consensus control to achieve robust control of MASs.
- Evaluating the proposed algorithms through simulations and real-world experiments to demonstrate their effectiveness and practicality in solving complex robotic tasks.
- Investigating the scalability and generalizability of the proposed algorithms, particularly in scenarios with high-dimensional state and action spaces, and in environments with significant uncertainties and dynamic changes.
- Analyzing the trade-offs between different approaches in terms of computational complexity, memory requirements, and learning speed, and providing insights into the design choices and parameter tuning for optimal performance.

1.3 Contributions and Thesis Organisation

1.3.1 Contributions

This thesis focuses on DRL with manipulators, particularly UR5 robot arms. The contributions of this thesis can be concluded as follows.

- A novel off-policy DRL method is developed to tackle the UR5 robot arm path planning problem. A standard path planning method has been implemented in the actual UR5 robot arm as a baseline in order to make a comparison of the advantages and disadvantages of both methods.
- After exploring the path planning problem, a vision-based self-supervised DRL method is proposed for UR5 robot arms to learn to pick and place objects. The proposed method can pick and place stacked and crowded objects safely in challenging environments.
- Considering the benefits of collaborative learning, it is interesting to expand from single robotic arm control to multiple robotic arm control. A novel algorithm which combines actor-critic based off-policy DRL with consensus-based distributed training is proposed for nonlinear MASs.
- After the study on multiple robotic arm control, sim-and-real training is considered for manipulators. A Consensus-based Sim-and-Real (CSAR) method is illustrated for UR5 robot arms to learn pick-and-place tasks.

1.3.2 Thesis Organisation

This thesis is organized as follows:

Chapter 2: Preliminaries and Literature Review

This chapter introduces some related preliminaries including robot kinematics, motion planning, RL methods, and consensus control.

Chapter 3: A Deep Reinforcement Learning Approach for Robotic Manipulators

A new off-policy DRL method is proposed to handle the problem of the UR5 robot arm path planning. Unlike standard path planning methods, the proposed method can guarantee a smooth movement of the UR5 robot arm. During each movement, all joint angles of the UR5 robot arm lie within the allowable range. Moreover, a standard path planning method has been applied to the real UR5 robot arm as a baseline to make a comparison of the benefits and drawbacks of both methods. The results in this chapter have been published in [243].

Chapter 4: Deep Reinforcement Learning with Manipulators for Pick-and-place

A complete self-supervised vision-based DRL method is developed for manipulators to learn to pick and place objects. Under the implementation of the height-sensitive action policy, the proposed method is able to pick and place crowded and stacked objects safely in challenging environments. The performance of the proposed approach is validated in both simulated and real-world environments. By stimulating the UR5 robot arm to suction the area closer to the centre of targets, the training model with the proposed method can accomplish pick-and-place tasks with a suction success rate of 90% without any fine-tuning from the real world. The presented approach can also be implemented to novel objects with a suction success rate of 90%.

Chapter 5: Distributed Neural Networks Training for Robotic Manipulation with Consensus Algorithm

This chapter illustrates a novel approach which combines distributed NNs with consensus algorithms. A multi-agent training algorithm with actor-critic based off-policy DRL and consensus-based distributed training is proposed. A multi-agent training framework is developed to validate the implementation of the proposed algorithm. The convergence of this algorithm is verified in the presence of the actor and critic training parameter. Compared with traditional centralized training, the proposed distributed multi-agent training framework has better scalability with a limited bandwidth when dealing with lots of agents and protects the privacy of each agent. The efficiency and feasibility of the proposed method are validated by experiments with several groups of UR5 robot arms. The results in this chapter have been published in [224].

Chapter 6: Sim-and-Real Reinforcement Learning for Manipulation: A Consensus-based Approach

This chapter describes a complete CSAR method for manipulators to learn pick-and-place tasks. By implementing consensus-based training, the proposed approach saves training time and decreases the number of required real-world robot training steps while keeping a comparable suction success rate, which is also cost-effective. A lightweight and end-to-end NN is developed to train the suction policy, which uses visual data directly. The feasibility and effectiveness of the CSAR method are validated via simulation and real-world experiments.

Chapter 7: Conclusion and Future Work

This chapter summarizes this thesis and discusses the potential future work.

Chapter 2

Preliminaries and Literature Review

Some related preliminaries are illustrated in this chapter. The concept of forward and inverse kinematics are first demonstrated, together with some existing motion planning methods. Additionally, mathematical formulation and typical RL algorithms are elucidated, which provides inspiration for the proposed methods in this thesis. Moreover, Kronecker product and graph theory are also introduced, and some existing consensus theorems are reviewed.

2.1 Robot Kinematics

Over the past two decades, robots have already established their worth in both theory and practical applications, including motion planning tasks [244], formation techniques [231], human-robot interaction [245], collision avoidance [233; 102], pick and place tasks [246]. Since the applications of robot arms are quite comprehensive, there are many studies that are related to the kinematics of the robot arm. One of the fundamental goals of robotics is to control a robot's movements and accurately determine its position and orientation in space. Two fundamental concepts that are widely used in robotics to achieve this goal are Forward Kinematics (FK) [247] and IK [248]. FK [249] refers to the process of determining the position and orientation of a robot's end effector based on its joint angles. IK [250] refers to the process of determining the joint angles required to position a robot's end effector at a particular location and orientation in

space.

2.1.1 Forward Kinematics

FK is a relatively straightforward concept and is widely used in robotics for tasks such as trajectory planning and motion control. As depicted in [251], FK refers to using joint angles and kinematic equations of robot arms to compute the position of the end effector. FK is a key concept in robotics that plays a significant role in determining the position and orientation of a robot's end-effector based on the joint angles and link lengths of the robot's arm [252]. The FK model essentially maps the robot's joint space to its task space, making it a crucial component in enabling the robot to move to a desired position and orientation and perform a specific task. This makes FK an essential tool for various industrial applications where precise movements are required, including manufacturing [253] and assembly lines [254]. Accurately reaching the desired position and orientation of the end-effector of a robot is critical to ensure successful task execution.

When dealing with the FK of robot arms, there is only one solution to a robot arm movement and for that reason, the rotation angles of each joint in a robot arm are provided. The mathematical representation of FK in robotics is commonly based on the Denavit–Hartenberg parameters [255], which is a widely used convention in robotics due to its simplicity and computational efficiency. As a result, the final position of the robot arm end effector can be easily calculated by multiplying Denavit–Hartenberg parameters of each link. The Denavit–Hartenberg parameters can be computed as follows:

$$T_d = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d \\ 0 & 0 & 0 & 1 \end{bmatrix}, R_\theta = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.1}$$

$$T_a = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, R_\alpha = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

$$T_{D-H} = T_d R_\theta T_a R_\alpha = \begin{bmatrix} \cos\theta & -\sin\theta\cos\alpha & \sin\theta\sin\alpha & a\cos\theta \\ \sin\theta & \cos\theta\cos\alpha & -\cos\theta\sin\alpha & a\sin\theta \\ 0 & \sin\alpha & \cos\alpha & d \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

where T_d stands for the translation along z-axis, R_θ is the rotation around z-axis, T_a represents the translation along x-axis and R_α stands for the rotation around x-axis.

In addition to the Denavit–Hartenberg method, Dual Quaternions [256] and Screw theory [257] are two widely-used mathematical tools in robotics for modelling and controlling the motion of robot manipulators. Dual Quaternions are a mathematical representation that combines a quaternion and its dual, providing a concise and computationally efficient way to represent rigid body transformations [247]. This approach is often used to model the kinematics of robotic systems in a way that is easy to implement and computationally efficient. Screw theory is a mathematical framework that describes the motion of a rigid body as a combination of a translational and a rotational component [258]. This approach is particularly useful for modelling the motion of robots that have both rotational and translational degrees of freedom. By integrating Denavit–Hartenberg parameters to describe the kinematic structure of the robot, dual quaternions to represent the rigid body transformations, and screw theory to model the motion, it is possible to develop advanced algorithms for motion planning, control, and simulation of robotic systems.

2.1.2 Inverse Kinematics

Although the position of the robot arm end effector is unique by applying FK, it cannot deal with the demand of making the robot arm end effector reach a specific point. In order to accomplish this requirement, the IK solutions of a robot arm are needed.

By using IK, it is possible to calculate each joint angle rotation of each robot arm link if the end effector position of the robot arm is given. Nevertheless, the nonlinear property of the robot movement makes it harder to find analytical solutions in working out robot IK solutions compared to finding numerical solutions. Due to the limitations of each robot joint rotation, using numerical solutions to handle robot arm motion planning is more convenient than using analytical solutions.

IK is a crucial concept in robotics that deals with determining the joint angles of a robot necessary to achieve a desired end-effector pose. One of the most widely used methods for solving IK problems is Jacobian IK [36], which leverages the Jacobian matrix to relate the velocities of the end-effector to the velocities of the joints. The Jacobian matrix maps the joint velocities to the end-effector velocities in a particular configuration. Jacobian IK method is able to solve the IK problem recursively by updating the inverse Jacobian of the robot arm and the position of the robot arm end effector inside each loop. The end effector velocity of a robot arm can be represented by

$$\dot{x} = J\dot{\theta} \tag{2.4}$$

where J stands for the Jacobian matrix, $\dot{x}$ represents the velocity of end effector, $\dot{\theta}$ is a vector of joint angles. Therefore, equation (2.4) can be rewritten as

$$\dot{\theta} = J^{-1}\dot{x} \tag{2.5}$$

where J^{-1} stands for the inverse matrix of Jacobian. However, the inverse of the Jacobian does not exist if J is not a square matrix. Furthermore, there is always a possibility that the Jacobian matrix could be a singular matrix, which means the determinant of J will be 0. As a result, the value of J^{-1} will be infinity and the robot arm will suffer from Singularities. Singularities arise when the Jacobian matrix becomes non-invertible. In such cases, it is impossible to determine the required joint velocities for achieving the desired end-effector pose. Another limitation of the Jacobian IK method is the presence of local minima, which occurs when the solution space is not unique and there are multiple solutions that can result in the desired end-effector pose [259].

In order to solve this problem, the Jacobian pseudo-inverse is used to deal with singularity problems [37]. As stated in [39], the solution of equation (2.5) can be illustrated as

$$\dot{\theta} = J^{+}\dot{x} + (I - J^{+}J)\dot{\zeta} \tag{2.6}$$

If J is full row rank, the pseudo-inverse of the Jacobian can be expressed as

$$J^{+} = J^{T}(JJ^{T})^{-1} \tag{2.7}$$

where J^{+} is the pseudo-inverse of the Jacobian, $\dot{\zeta}$ stands for an arbitrary n-dimension vector in $\dot{\theta}$ space. Although using J^{+} is able to handle the non-square matrix problem of J, Equation (2.7) can not be used if J does not have full rank. Moreover, it cannot make the robot arm get rid of oscillation and movement discontinuities when its position is close to a singularity. Due to calculation inaccuracy, the joint values calculated by the pseudo-inverse method could be a hundred times larger than the desired movement when the robot arm moves near singularities.

With the aim of solving the drawback of J^{+} above, Wampler [260] and Nakamura [261] had developed damped least square method which substitute J^{+} as

$$J^{+} = J^{T}(JJ^{T} + \lambda^{2}I)^{-1} \tag{2.8}$$

Compared to equation (2.7), an extra parameter, λ, was added to the denominator of J^{+}. As a consequence, when the value of JJ^{T} is close to zero, the value of the extra parameter will make the denominator part of J^{+} larger, thereby decreasing the whole value of J^{+} in order to minimize the robot arm oscillation when it reaches singularities.

An extension of the damped least square method was illustrated by Maciejewski [262] which combined the damped least square method with singular value decomposition. Therefore, equation (2.8) can be illustrated as

$$J^{+} = \sum_{i=1}^{n} \frac{\sigma_i}{\sigma_i^{2} + \lambda^{2}} v_i u_i^{T} \tag{2.9}$$

where σ_i, v_i and u_i provide singular value decomposition of J. Compared to traditional singular value decomposition, an extra term λ^{2} was added into the denominator of J^{+}. Similar to equation (2.8), the movement discontinuities of a robot arm when it gets

close to the singularities can be significantly diminished, thus making the performance of the robot arm smoother near singularities.

In addition to the Jacobian IK method, there exist several other advanced IK algorithms, such as Artificial Neural Network (ANN) based IK [263], Genetic Algorithm (GA) based IK [264], Simultaneous Localization And Mapping (SLAM) based IK [265], etc. ANN-based IK leverages a NN to map the end-effector position to joint angles and can handle high-dimensional systems through training with large amounts of data [266]. However, it can suffer from overfitting and necessitates substantial computational resources. GA-based IK involves using a population of candidate solutions and iteratively selecting and breeding the fittest individuals to find an optimal solution [267]. This approach can handle non-linear and non-convex optimization problems but may require careful tuning of parameters and can be computationally expensive. SLAM-based IK employs sensor data to estimate the robot's pose and environment and then uses this information to perform IK [268]. This technique can handle complex and uncertain environments but necessitates sophisticated sensors and algorithms.

2.2 Motion Planning

Motion planning is a fundamental problem in robotics and control systems [269]. It involves finding a collision-free path for a robot to move from its initial position to a goal position [270]. The problem can be considered as finding a feasible trajectory for the robot to follow while satisfying certain constraints and objectives. To solve the motion planning problem, various approaches have been developed over the years, including optimization-based methods [271], heuristic-based methods [272; 273], and sampling-based methods [274; 275]. Optimization-based methods [276] formulate the motion planning problem as an optimization problem and use optimization techniques to find the optimal solution. These methods are often computationally expensive but can find optimal solutions. Heuristic-based methods [277] use a set of rules or heuristics to generate feasible trajectories for the robot. These methods are fast and efficient, but they may not always find an optimal solution. Sampling-based methods [278] generate a set of random configurations and connect them to form a collision-free

path. These methods are often probabilistically complete, meaning that they can find a solution if one exists.

2.2.1 Optimization-based Method

Optimization-based methods are commonly employed for motion planning in robotics [279]. These approaches typically involve formulating the motion planning problem as an optimization problem, which is then solved using numerical optimization techniques. The objective function of the optimization problem is usually defined to minimize some cost, such as the distance travelled, the time required, or the energy consumed, while taking into account various constraints, including obstacle avoidance, collision avoidance, and kinematic constraints [280]. One of the most representative examples is TRAC-IK proposed by Beeson [276]. Aiming at finding the best possible motion planning solution, the optimization target function of Sequential Quadratic Programming (SQP) can be represented as

$$\text{argmin } (q_{seed} - q)^T (q_{seed} - q) \tag{2.10}$$

$$\text{s. t. } f_i(q) \leq b_i, i = 1, ..., m \tag{2.11}$$

where q_{seed} stands for the n-dimensional seed value of joints and $f_i(q)$ represents inequality constraints which are composed by joint limits of the robot arm, the error of the euclidean distance, and the error of angular distance. The minimized objective function for SQP-SS in [276] can be described as

$$\phi_{ss} = p_{\text{err}} p_{\text{err}}^T \tag{2.12}$$

The TRAC-IK algorithm is made up of both the KDL-RR algorithm and the SQP-SS algorithm, where the KDL-RR algorithm represents the traditional pseudo-inverse Jacobian algorithm. Two algorithms work simultaneously when the whole TRAC-IK algorithm is initiated until either algorithm finds the one IK solution of the robot arm. As a result, the average solution time of TRAC-IK is less than the single SQP-SS algorithm and the IK solve rate of TRAC-IK is higher than the single KDL-RR algorithm. As shown in [276], taking Atlas 2015 as an example, the average solution time of TRAC-IK is 0.1ms while the average solution time of the SQP-SS algorithm is 0.25ms. The average solve rate of TRAC-IK is 96.56% while the average solve rate

of KDL-RR is only 94.13%.

The main limitations of TRAC-IK are having self-collisions, reaching singularities, and dealing with movement discontinuities. In order to refine these imperfections, Rakita introduced an optimization-based IK solver called RelaxedIK [281]. Similar to Trac-IK, optimization target functions are also used in RelaxedIK. The key optimization function of RelaxedIK in [281] can be demonstrated as

$$\theta = \text{argmin } f(\theta) \tag{2.13}$$

$$\text{s.t. } c_i(\theta) \geq 0 \tag{2.14}$$

$$c_e(\theta) = 0 \tag{2.15}$$

$$l_i \leq \theta_i \leq u_i, \tag{2.16}$$

where $c_i(\theta)$ stands for inequality constraints, $c_e(\theta)$ stands for equality constraints, l_i is the lower bound of robot joints, u_i is the upper bound of robot joints, f represents the target objective function which is composed of a weighted sum of individual optimization functions such as end effector position matching, singularity distance, and minimum joint jerk. The value of $f(\theta)$ in equation (2.13) can be expended as follows:

$$f(\theta) = \sum_{i=1}^{n} w_i h_i(\theta, v(t)) f_i(\theta, \omega_i) \tag{2.17}$$

where w_i is each weight term value, $h_i(\theta, v(t))$ stands for the dynamic weight function, $f_i(\theta, \omega_i)$ stands for the single objective function. Compared to TRAC-IK, RelaxedIK trains a NN to teach the robot arm to avoid self-collision. Although the solution time of RelaxedIK is more than using TRAC-IK, the joint jerk of RelaxedIK is much less than that using TRAC-IK. In other words, the joint performance of RelaxedIK is much smoother than that of TRAC-IK.

Although optimization-based motion planning has achieved significant progress, there are still several challenges and limitations that need to be overcome. One of the primary challenges is the high computational complexity and memory requirements of these methods, especially for high-dimensional problems. Additionally, formulating an accurate and tractable cost function that captures real-world objectives and constraints is another challenging task. Furthermore, optimization-based methods may

be sensitive to the initial conditions and can get stuck in local minima, which may lead to suboptimal solutions.

Trajectory optimization methods, such as SQP [271], Interior-Point methods [282], and Gradient-Based Optimization [283], represent state-of-the-art approaches in optimization-based motion planning. These methods employ various techniques, such as gradient descent and nonlinear optimization, to find the optimal solution to the motion planning problem [69]. Trajectory optimization methods have the advantage of producing high-quality trajectories that satisfy constraints and optimize the cost function.

2.2.2 Heuristic-based Method

Heuristic-based methods are a class of problem-solving approaches that rely on practical and intuitive techniques to find solutions when exact methods are not feasible or efficient [284]. Unlike exact algorithms, heuristic methods do not guarantee optimal solutions, but instead aim to find solutions that are acceptable or satisfactory within a reasonable amount of time [285]. Heuristic-based methods are often useful in scenarios where the problem is complex or has a large number of variables, making it challenging to find an exact solution using traditional methods.

Despite the lack of optimality guarantees, heuristic-based methods have proven to be efficient in finding satisfactory solutions within a reasonable time frame. One of the advantages of heuristic-based methods is their flexibility and adaptability, making them suitable for various problem types and sizes [272; 277]. Heuristic iterative search algorithms are able to simplify the joint chain problem of multiple joints into a single joint problem. Since minimizing the objective function of one single joint is quite simple, the computation speed of each iteration can be quite fast. CCD algorithm is one of the most representative heuristic iterative search algorithms. This algorithm was first introduced by Lander [78] in 1998, after which many scholars have made their own contributions to improve the performance of the CCD algorithm with the aim of meeting typical demands. Algorithm 2.1 details this algorithm.

Algorithm 2.1 Cyclic-Coordinate Descent (CCD) algorithm

1: Initialize the joint angles of the robot arm to an initial guess.
2: Compute the FK of the robot to obtain the position and orientation of the end-effector.
3: Compute the vector from the current joint to the end-effector.
4: For each joint, starting from the first joint and ending at the last joint:
   a. Compute the vector from the current joint to the end-effector.
   b. Compute the vector from the current joint to the next joint.
   c. Compute the angle between the two vectors.
   d. If the angle is greater than a threshold value, update the joint angle by rotating the joint about its axis in the direction that brings the end-effector closer to the target pose.
5: Compute the FK again to obtain the new position and orientation of the end-effector.
6: If the difference between the desired end-effector pose and the actual end-effector pose is less than a tolerance value, terminate the algorithm. Otherwise, go to step 3.

CCD is a heuristic-based optimization algorithm commonly used in motion planning and IK problems for manipulators with a serial chain structure [78]. CCD optimizes each joint angle sequentially in a cyclical manner until the end effector of the robot reaches the desired position and orientation. The algorithm has gained popularity due to its simplicity, fast convergence, and ability to handle complex kinematic structures efficiently.

The CCD algorithm offers several advantages that make it a popular choice for solving motion planning and IK problems. Firstly, it is computationally efficient and can handle complex robotic systems with numerous degrees of freedom. Additionally, the algorithm has demonstrated the ability to converge to a solution from a wide range of initial joint configurations. However, one limitation of the CCD algorithm is that it can be sensitive to the ordering of the joints, which can lead to different solutions for different joint orderings. Additionally, it can get stuck in local minima, which can result in suboptimal solutions. If the location of the final desired point is close to the

baseline, the trajectory of the robot arm will be a zigzag curve rather than a straight line. Thus, the motion planning of the robot arm needs to be optimized. Aiming at solving this problem, Mukundan [79] developed an improved CCD algorithm called the improved triangulation algorithm. Instead of moving all the links of the robot arm one by one from the end effector to the base, this improved triangulation algorithm divided the links of the robot arm through midpoints. As a result, half of the robot arm moves together instead of a single link of a robot arm, thereby enabling large-angle rotations rather than oscillating trajectories.

There are also other state-of-art heuristic-based methods for solving motion planning problems in robotics, such as Forward and Backward Reaching Inverse Kinematics (FABRIK) [286]. FABRIK uses heuristics to iteratively update the joint angles until an acceptable solution is found. What distinguishes itself from the CCD algorithm is that FABRIK updates the position of the joints from both the start and end of the chain. FABRIK is a heuristic-based method that is based on the assumption that each link in a chain can be modelled as a straight line. The algorithm iteratively updates the joint angles of the chain until the end effector position is within an acceptable tolerance of the desired position. FABRIK is computationally efficient and can be used in real-time applications. However, it should be noted that FABRIK does not guarantee an optimal solution and can result in suboptimal solutions [273]. Despite this limitation, FABRIK's ability to solve for both position and orientation simultaneously makes it a versatile method that can be used for a wide range of robotic applications.

2.2.3 Sampling-based Method

Sampling-based motion planning algorithms are employed to find a feasible path for a robot to move from an initial to a desired goal configuration, while ensuring that it avoids obstacles in the environment [275]. Unlike optimization-based methods, which utilize mathematical models to compute an optimal trajectory, sampling-based methods generate a set of feasible paths by randomly sampling the configuration space of the robot. From the generated set of paths, the best path is then selected.

Algorithm 2.2 Probabilistic Roadmaps Method (PRM)

1: Create a roadmap by randomly sampling configurations of the robot in the environment. Each configuration is checked for collision with obstacles in the environment using collision detection.
2: Connect nodes in the roadmap to form edges between them if the straight-line path between them is collision-free.
3: Given a start configuration and a goal configuration, find the nodes in the roadmap that are closest to the start and goal configurations. Then, use a search algorithm (such as A* [76]) to find the shortest path between the start and goal configurations in the roadmap.
4: The resulting path may not be collision-free, so refine the path by performing local planning to adjust the path to avoid collisions. This can be done by applying IK to find the joint angles that correspond to the waypoints in the path and then smoothing the resulting path.
5: Execute the refined path on the robot.

Sampling-based motion planning methods are preferred in many cases because of their simplicity, scalability, and ability to handle high-dimensional configuration spaces [274]. Among these methods, PRM [83] is one of the most widely used algorithms. PRM constructs a roadmap, which is a graph representation of the free space, and can be used to efficiently answer queries regarding motion planning in that space. The PRM algorithm works by randomly sampling the free configuration space of a robotic system, and connecting these samples in a graph using collision-free paths. This graph can then be used to find a feasible path between any two configurations of the robotic system [287]. Algorithm 2.2 details this algorithm.

The PRM algorithm has several advantages over other motion planning algorithms, including scalability to high-dimensional configuration spaces and the ability to handle complex obstacle shapes [287]. It is computationally efficient and can handle high-dimensional configuration spaces. However, it can be difficult to tune the parameters of the algorithm to obtain good performance, and it may not always find a feasible path if the configuration space is highly cluttered or has narrow passages.

Sampling-based motion planning algorithms are a powerful tool for solving complex motion planning problems with high-dimensional configuration spaces [288]. Apart from the PRM method, there exist numerous other state-of-the-art sampling-based motion planning algorithms that have been developed and shown to be effective in solving a wide range of motion planning problems. With the aim of solving the single-query motion planning problems, the Rapidly-exploring Random Trees (RRTs) method [289; 290] has been widely used. RRT generates a tree-like structure by iteratively adding new nodes to the tree through random sampling of the configuration space. The RRT algorithm has been successfully applied to various robotic tasks such as path planning for mobile robots [291] and manipulators [292]. The core of the RRT algorithm is to generate random points evenly in order to find the possible route solution [293]. During each iteration, a random point is generated and the RRT algorithm will try to link the newly generated point with the current RRT tree [294]. An RRT-connect method was proposed in [295] with the aim of improving the efficiency of finding motion planning solutions. In the proposed algorithm, two RRT trees can be generated both from the start point and the goal point. If two RRT trees are connected, they should be swapped to reverse their roles. As a consequence, the performance of using two RRT trees at the same time is much better than using a single RRT tree. One of the limitations of using the RRT algorithm to do motion planning is that the performance of the RRT tree is not that desirable when the algorithm is used to find a path inside a maze. This is because the extending point is totally randomly generated, which leads to a large number of unnecessary RRT tree expansions.

2.3 Reinforcement Learning

RL is a branch of machine learning that focuses on training an agent to make decisions that lead to maximum cumulative reward in a given environment [296]. Unlike other popular machine learning approaches such as supervised [297] or unsupervised learning [298], RL is characterized by trial-and-error learning. During training, the agent interacts with the environment, taking actions that lead to rewards or penalties, which inform future decision-making [299]. RL autonomously enables the machine to

learn the mapping relationship between the state of the environment and its actions [300]. Therefore, the machine can adjust its action output according to the feedback signal in order to maximize the cumulative reward from the environment [301].

RL can be particularly advantageous in complex and dynamic environments where the optimal solution may change over time or where the rules of the game are not fully understood [302]. Moreover, RL algorithms can efficiently handle high-dimensional and continuous state spaces, making them suitable for a wide range of applications [115]. Agents are encouraged to learn mapping relationships between the environment and action with RL techniques [303]. Recently, many RL methods have been investigated to solve classical problems. For instance, an actor-critic based RL approach was developed in [304] to deal with the input of a classical formation control problem. In [305], a solution for achieving consensus of multiple agents under sudden total communication failure was proposed by using the actor-critic RL method. A distributed off-policy actor-critic method was developed in [2] to deal with RL problems in a MAS. To summarize, RL has become increasingly popular in recent years due to its ability to solve complex problems that are difficult or impossible to solve using traditional rule-based programming or other machine learning approaches [306].

RL has the potential to transform the field of robotics by enabling robots to learn to perform tasks that are too complex for conventional control methods [307]. By leveraging RL, robots can learn from their experiences and dynamically adapt their behaviours to changes in their environment [308; 309]. This enables them to complete tasks with greater accuracy and speed than traditional control methods. One of the main advantages of RL in robotics is that it allows robots to learn from their experiences, instead of relying solely on pre-programmed instructions [310]. As a result, robots can now perform tasks that are too complex or too dynamic for traditional control methods. Furthermore, RL enables robots to learn from their mistakes, which is critical for tasks that demand high precision [311]. However, there are also some drawbacks to using RL in robotics. One of the primary challenges is that RL requires a large number of trials before a robot can learn a control policy that performs well [312]. This can be a time-consuming and expensive process, particularly for tasks that

require a lot of interaction with the environment. Additionally, RL is susceptible to overfitting, which can result in poor performance when the robot is placed in a new environment [127]. Despite these challenges, RL has been successfully applied to a broad range of robotic tasks, including grasping [313; 314], navigation [315], and manipulation [316; 317].

With the rapid development of science and technology, RL is required to handle complex tasks [318; 319]. In order to solve this problem, people combine deep learning with RL with the aim of tackling the complex real environment [320]. Deep learning [321] is a subfield of machine learning that utilizes NNs with multiple layers to model and solve complex problems. Deep learning [322] requires extracting feature information from raw data to complete learning through NN [323], which has been used extensively in computer vision [324; 325], natural language processing [326; 327], etc. RL is a subset of machine learning that deals with training agents to make decisions in an environment to maximize rewards [328]. DRL [329] is an exciting and rapidly evolving field of Artificial Intelligence that combines the power of deep NNs [330] with RL algorithms to enable agents to learn from high-dimensional sensory input and make intelligent decisions in complex and dynamic environments. DRL is a method that utilizes deep NNs to approximate the optimal policy function of a RL problem. This approach is particularly effective in learning complex and high-dimensional state-action spaces, and it enables the agent to make decisions based on raw sensory input [331].

In recent years, DRL has emerged as a powerful tool in the field of robotics, as it has enabled robots to learn complex motor skills [224]. By combining deep NNs with RL, DRL has enabled robots to adapt to changing environments and perform a wide range of tasks [332]. An efficient real-time hybrid path planning scheme was proposed in [333] to handle the uncertain dynamics of a robot manipulator by combining the PRM method with DRL. In [131], a robot arm was trained to learn how to fold a towel diagonally with DRL. In [334], a robotic manipulator was trained using DRL to solve the task of grasping an initially invisible object via a sequence of grasping and pushing actions. A high-precision peg-in-hole target task was selected in [335] for force-controlled robotic assembly with DRL. Specifically, the force and moment of the

robotic manipulator end effector were chosen as the state. Although DRL has shown great promise in various applications, it is not without its challenges and limitations. One of the main hurdles is the requirement for vast amounts of data to train deep NNs. Furthermore, the process of training DRL agents can be computationally expensive and time-consuming, posing a significant practical challenge in some cases [336]. Designing appropriate reward functions that effectively capture the true objectives of the task is another challenging aspect of DRL. Incorrectly specified reward functions can lead to undesirable behaviours, and even cause harm in certain situations.

2.3.1 Mathematical Formulation

Markov Decision Process [303] is a mathematical framework for modelling decision-making problems in situations where outcomes are partly random and partly under the control of a decision-maker. It is a fundamental concept in the field of RL and decision theory. In Markov Decision Process, the decision-maker takes actions that transition the system from one state to another according to a probabilistic rule called the transition function. The transition function defines the probability of moving from one state to another as a result of a specific action. In addition, the reward function defines the amount of reward or penalty that the decision-maker receives when transitioning between states. The objective of the Markov Decision Process is to find a policy that maximizes the expected cumulative reward over time. Figure 2.1 introduces the interaction between the agent and environment in a Markov decision process. At each iteration t, the agent receives information from the environment's state s_t and selects an action a_t. At next iteration $t+1$, the agent receives the reward r_{t+1} and finds itself in a new state s_{t+1} [303]. The key elements of Markov decision process can be listed as follows [303]:

Action: A decision made by the agent at a particular state. The action can affect the state and the reward.

State: A particular situation or condition that the agent is in at a given time. The state can affect the action and the reward.

Agent: The entity that is learning and making decisions based on the environment. The agent takes actions and receives rewards from the environment.

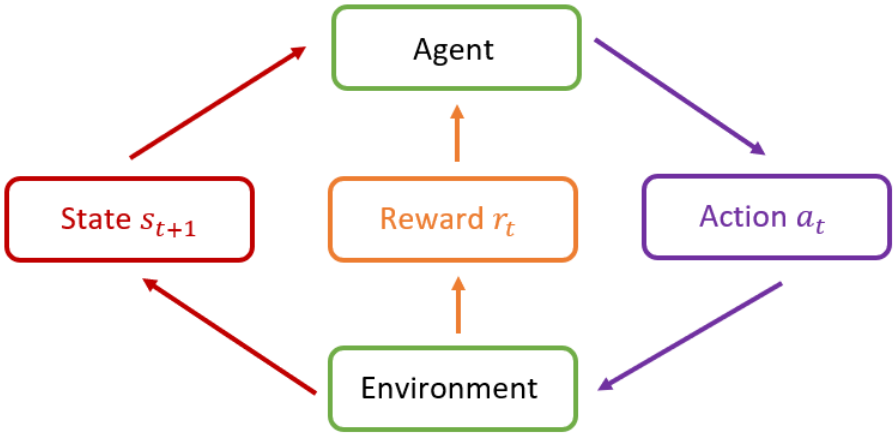


Figure 2.1: The interaction between agent and environment in a Markov decision process. State represents the current status. Agent is an individual that performs the action. Action stands for agent behaviour. Reward is the feedback given after completing the action. Environment can judge the next state and provide a reward.

Reward: A numerical value that the agent receives from the environment based on the action taken and the current state. The goal of the agent is to maximize the cumulative reward over time.

Environment: The external system or process that the agent interacts with. The environment provides feedback to the agent in the form of rewards based on the actions taken and the current state.

Discounted Reward

The discounted reward is a way to quantify the cumulative future rewards an agent can expect to receive when following a specific policy in an environment. As stated in [303], the goal of the agent is to maximize the total amount of reward. In the simplest case, the reward can be represented by

$$G_t = R_{t+1} + R_{t+2} + \cdots + R_T \tag{2.18}$$

where T is the final step. However, we take future rewards into consideration in the actual RL implementation. Therefore, the discounted reward is given by

$$G_t = R_{t+1} + \rho R_{t+2} + \rho^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \rho^k R_{t+k+1} \tag{2.19}$$

where $0 \leq \rho \leq 1$, and ρ is the discounted factor.

State Value Function

If a policy π denotes a mapping from states to probabilities of choosing each possible action [303], the state value function $v^{\pi}(s)$ of a state s under π can be defined by

$$v^{\pi}(s) = \mathbb{E}[G_t|S_t = s] = \mathbb{E}[\sum_{k=0}^{\infty} \rho^k R_{t+k+1}|S_t = s] \tag{2.20}$$

where $\mathbb{E}$ is the expected value, S_t stands for the current state.

Action Value Function

Similarly, by following [303], the action value function $q^{\pi}(s, a)$ taking action a in state s can be described as follows

$$q^{\pi}(s, a) = \mathbb{E}[G_t|S_t = s, A_t = a] = \mathbb{E}[\sum_{k=0}^{\infty} \rho^k R_{t+k+1}|S_t = s, A_t = a] \tag{2.21}$$

where $\mathbb{E}$ is the expected value, S_t stands for the current state, A_t is the current action.

Bellman Equation

As stated in [303], RL satisfies recursive relationships. For any policy π and any state s, the following condition holds

$$\begin{aligned} v^{\pi}(s) &= \mathbb{E}[G_t|S_t = s] \\ &= \mathbb{E}[R_{t+1} + \rho G_{t+1}|S_t = s] \\ &= \sum_{a} \pi(a|s) \sum_{s',r} P(s', r|s, a)[r + \rho v^{\pi}(s')] \end{aligned} \tag{2.22}$$

where s' is the next state, a stands for the action, r represents the reward, $P(s', r|s, a) = Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\}$. Equation (2.22) is the Bellman equation. The value function $v^{\pi}(s)$ is the unique solution to its Bellman equation [303].

Optimal Policy and Optimal State Value Function

As described in [303], there always exists at least one policy which is better than or equal to all other policies. This is the definition of optimal policy. The optimal state value function $v^*(s)$ is defined as

$$v^*(s) = \max_{\pi} v^{\pi}(s) \tag{2.23}$$

for all $s \in \mathcal{S}$.

Optimal Action Value Function

Following [303], the optimal action value function $q^*(s, a)$ can be described by

$$q^*(s, a) = \max_{\pi} q^{\pi}(s, a) \tag{2.24}$$

for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$.

As stated in [303], the optimal action value function $q^*(s, a)$ can be written in terms of the optimal state value function $v^*(s)$:

$$q^*(s, a) = \mathbb{E}[R_{t+1} + \rho v^*(S_{t+1}) | S_t = s, A_t = a] \tag{2.25}$$

where $\mathbb{E}$ is the expected value, S_{t+1} stands for the next state, R_{t+1} represents the next reward, ρ is the discounted factor.

2.3.2 On-policy and Off-policy Algorithms

RL algorithms can be broadly classified into two categories: on-policy and off-policy algorithms [303]. Both these algorithms are used to learn an optimal policy for an agent to take actions in an environment to maximize its rewards. On-policy algorithms learn from the policy that they are currently following, while off-policy algorithms learn from a different policy than the one being used to select actions.

On-policy Algorithms

On-policy algorithms [303] are a class of RL algorithms that learn optimal policies by generating experience from the current policy. These algorithms work by using the current policy to both generate trajectories and update the policy parameters. In other words, the policy used to collect experience is the same policy that is being learned and updated. On-policy algorithms are often used in situations where it is difficult or expensive to obtain data from other policies, or when the current policy is already close to optimal and only small refinements are needed [337].

The most common on-policy algorithm is the SARSA (State-Action-Reward-State-Action) algorithm [303] which is shown in Figure 2.2. The SARSA algorithm updates

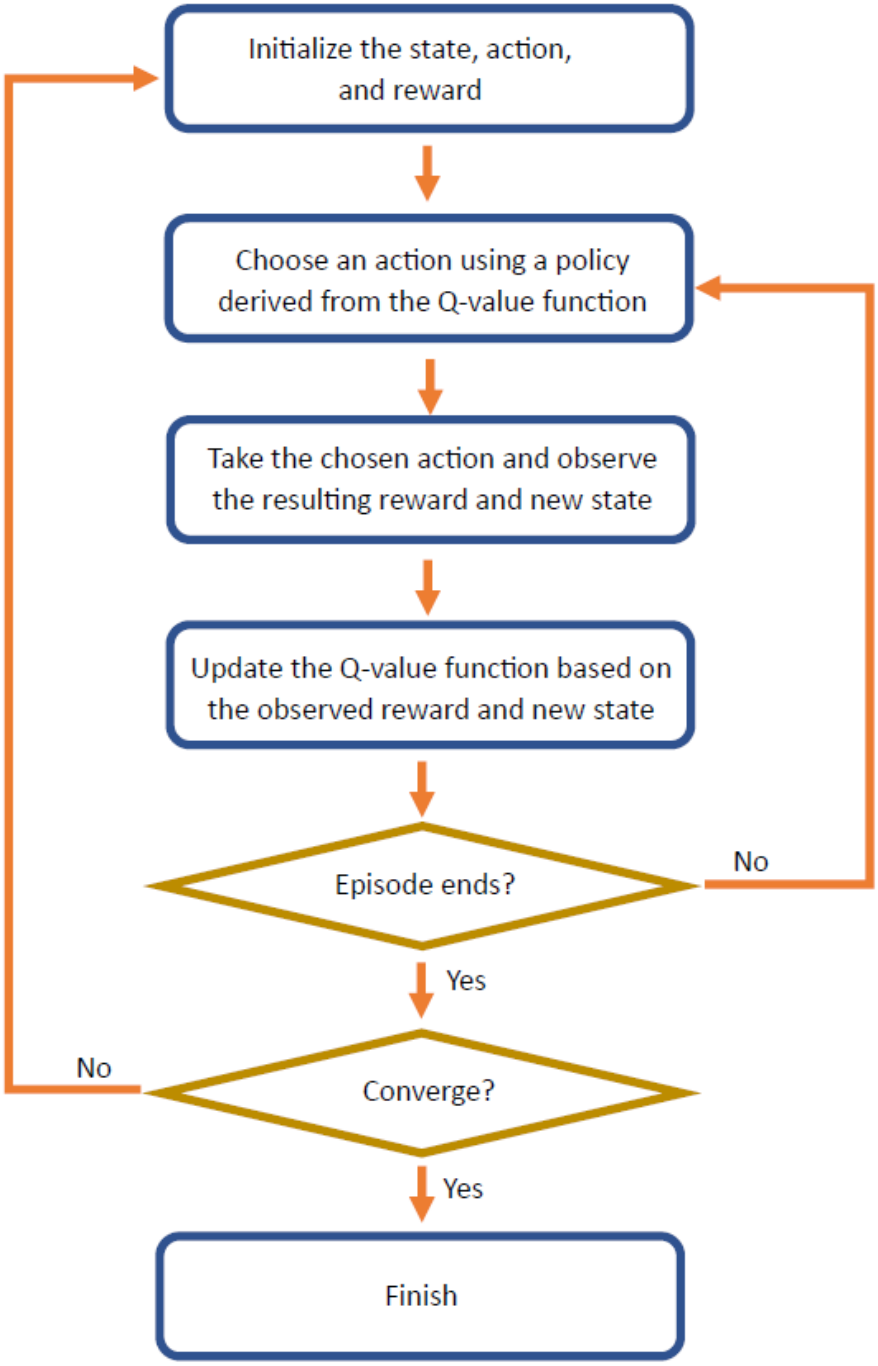


Figure 2.2: SARSA algorithm.

the Q-function using the current policy's action and the resulting next state and reward. The policy is then updated to be epsilon-greedy with respect to the updated Q-function. SARSA updates its policy based on the actions actually taken under the current policy, allowing it to learn policies that are better suited to changing environments. However, SARSA may be less effective than off-policy algorithms like Q-learning in environments where exploration is difficult or expensive, as SARSA relies on the current policy for exploration. This can make it more challenging for SARSA to learn an optimal policy, as it may not explore all possible actions in a given state.

Activation Functions

SARSA requires a finite number of states and actions, which can be a limiting factor in certain applications where the state space is continuous or the action space is infinite. To address this issue, deep learning approximates infinite-dimensional functions ((2.23) and (2.24)) by finite-dimensional functions parameterized by NNs. As stated in [338], if the parameterization is linear, the state value function $v(s)$ and the action function $q(s, a)$ can be described as:

$$v(s) = \eta^T \phi(s) \tag{2.26}$$

$$q(s, a) = \xi^T \psi(s, a) \tag{2.27}$$

where η and ξ are training parameters. $\phi(s)$ and $\psi(s, a)$ denote the feature functions that correspond to the state value function and the action value function, respectively.

Activation functions [339] are a critical element in the architecture of artificial NNs, as they enable the introduction of non-linearity into the output of neurons. These functions receive the weighted sum of the inputs and a bias term and then transform it into an output value, which determines the degree to which the neuron is activated. By introducing non-linearity, activation functions allow NNs to approximate complex non-linear relationships between inputs and outputs [340]. An example of a NN is visualized in Figure 2.3. The input signal enters from the input layer, passes through the nonlinear activation function and then is transmitted to the next hidden layer. This process will repeat until the input signal reaches the output layer. Thus, the NN has enough capacity to extract features from input. If there is no activation function,

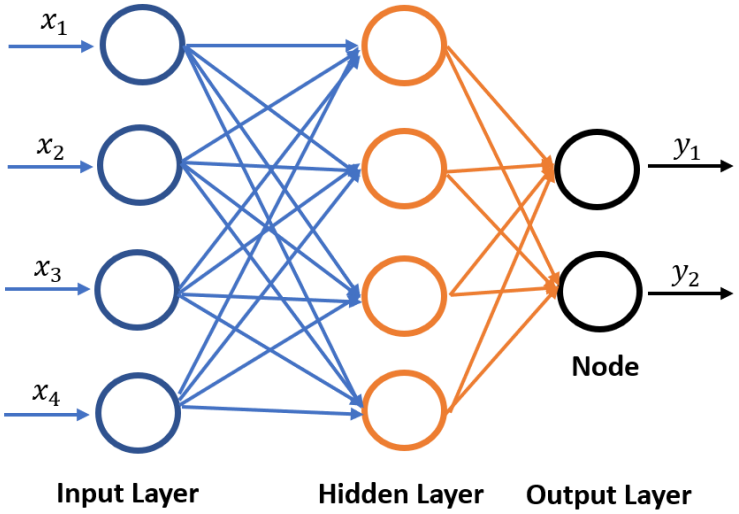


Figure 2.3: An example of NN. x stands for the input signal. y represents the output signal.

the NN can only process linear mapping, which makes it equivalent to a single-layer network. To sum up, the NN is able to acquire hierarchical nonlinear modelling ability by using the activation functions [341].

As illustrated in Figure 2.4, there are three activation functions that are mainly used in the NN. The first one is the sigmoid function [342], which is defined as follows:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.28}$$

where $\sigma(x)$ is the sigmoid function. Although the sigmoid function is convenient for derivation, sometimes it causes gradient vanishing. In this situation, the NN cannot be optimized and the training parameters cannot be updated. Moreover, the output of the sigmoid function is always greater than 0, which is not zero-centred. This will slow down the convergence speed of NN.

The second activation function is the tanh function [343], which is given by

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.29}$$

where $\tanh(x)$ represents the tanh function. Compared with the sigmoid function, it solves the zero-centred problem, which allows for faster convergence speed. Nevertheless, the gradient vanishing issue still exists when using the tanh activation function.

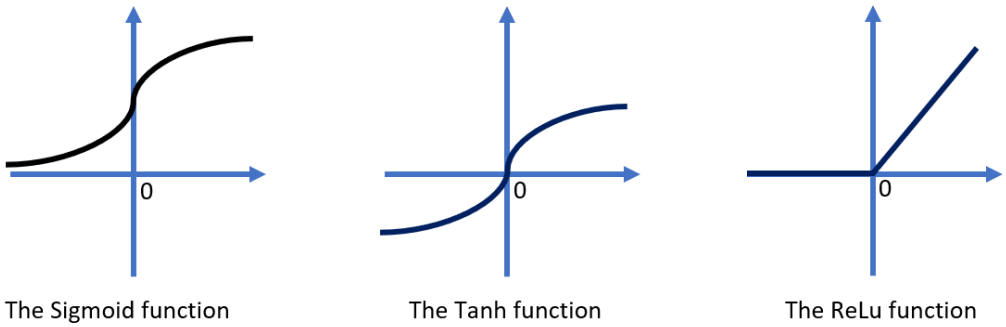


Figure 2.4: Three activation functions.

The last activation function is the ReLu function [344], which is computed by

$$\delta(x) = \max(0, x) \tag{2.30}$$

where $\delta(x)$ is the ReLu function. In the positive interval, the ReLu function is able to solve the gradient vanishing problem. The computation speed of the ReLu function is very fast since it only needs to judge whether the input is greater than 0. Consequently, the convergence speed of the ReLu function is much faster than the sigmoid function and the tanh function.

Off-policy Algorithms

Off-policy algorithms [303] are a type of RL method that learns a policy from a different policy than the one being optimized. Specifically, they learn a target policy based on data generated by a behaviour policy. This enables the algorithm to learn from a broader range of experiences than on-policy algorithms, which can only use data generated by the current policy. Off-policy methods are often preferred in scenarios where exploration is expensive or difficult because they allow for the use of previously collected data to improve the policy [345].

Q learning [346] is a well-known RL algorithm that has been applied to solve problems in a variety of domains. It is categorized as a model-free method since it does not require an explicit model of the environment to learn. Instead, it approximates the optimal action-value function, which estimates the expected reward for taking a specific action in a given state and following an optimal policy thereafter. Q learning is a variant of Temporal Difference (TD) learning, which means that it updates

the estimates of the action-value function based on the observed rewards and state transitions. The flowchart of the Q learning algorithm is shown in Figure 2.5.

2.3.3 Deep Q Learning

Algorithm 2.3 Deep Q Learning

Initialize training parameter ξ, target training parameter ξ', learning rate α, discounted factor γ, replay buffer R_p, action value function Q, target action value function Q'.
for episode = 1, N **do**
  Reset the environment.
  Receive state s_t.
  for t = 1, T **do**
    Choose random action a_t with probability ϵ.
    Otherwise select $a_t = \arg\max_a(Q(\xi, s_t, a))$.
    Execute action a_t and get reward r_t, next state s_{t+1}.
    Store (s_t, r_t, s_{t+1}, a_t) to R_p.
    Sample a minibatch (s_t, r_t, s_{t+1}, a_t) from R_p.
    $y_t = r_t + \gamma \max_{a'}(Q'(\xi', s_{t+1}, a'))$.
    Perform a gradient descent step on Loss L with respect to ξ:
    $L = \frac{1}{2}(y_t - Q(\xi, s_t, a_t))^2$.
    Every M steps reset $Q' = Q$.
  end for
end for

Deep Q Learning [347] is a variation of Q Learning that utilizes deep NNs to approximate the Q-value function. The idea is to use a deep NN as a function approximator for the Q-value function, instead of a table. In Deep Q Learning, the NN takes in the state as input and outputs the Q-values for all possible actions. The network is trained using an experience replay buffer and a target network to stabilize the learning process.

Deep Q Learning has several advantages including the ability to deal with high-dimensional state spaces and the ability to generalize well to unseen state-action pairs.

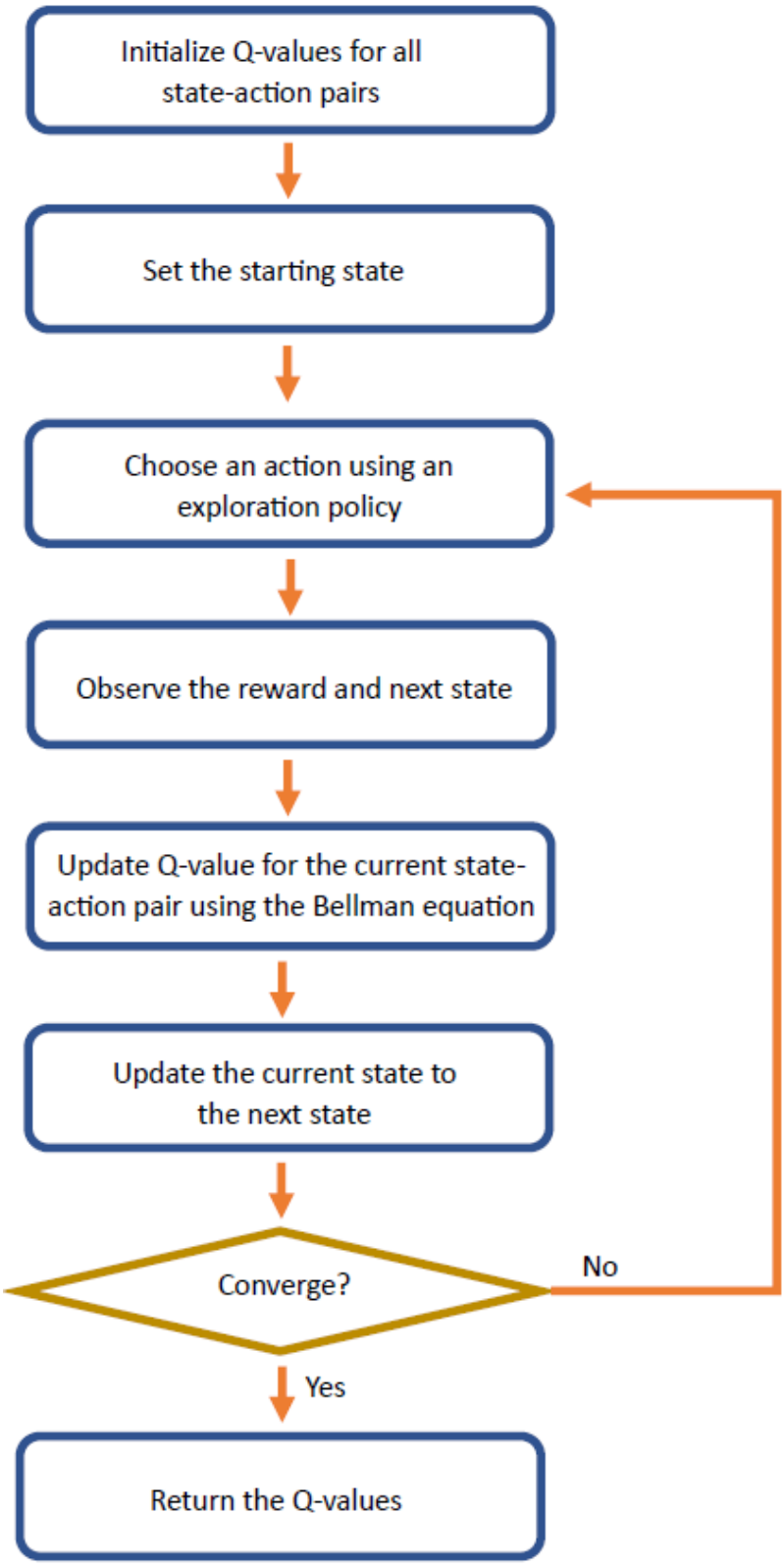


Figure 2.5: Q learning algorithm.

However, Deep Q Learning also presents some challenges that must be addressed to ensure its effectiveness. One of the main difficulties is the instability of training, which stems from the use of non-linear function approximators that can lead to overfitting and divergence of the Q-values. Another challenge is the need for large amounts of training data, which can be particularly time-consuming and computationally expensive in certain scenarios. The Deep Q learning algorithm can be described as shown in Algorithm 2.3.

2.3.4 Policy Gradient Method

The Policy Gradient method [348] is a well-known class of RL algorithms that directly optimizes the expected cumulative reward via gradient ascent to learn a policy. Unlike value-based methods such as Q learning, which estimate the optimal action-value function, the policy gradient method learns a parameterized policy function that maps states to actions directly. This feature enables the agent to learn policies that are continuous and stochastic, making it a suitable method for handling high-dimensional and continuous state spaces.

The policy gradient method aims to learn a parameterized policy by directly optimizing the expected cumulative reward. To achieve this, the method estimates the gradient of the expected reward with respect to the policy parameters using stochastic gradient ascent. The objective function, which represents the expected cumulative reward under the policy, is typically used as the optimization criterion. The policy parameters are then updated by following the gradient direction of the objective function, which can be estimated using Monte Carlo methods.

The policy gradient method has several advantages, including the ability to learn policies that can handle continuous and high-dimensional state and action spaces. These advantages make the policy gradient method a promising option for a variety of real-world applications. However, it also has some drawbacks, including a tendency to converge slowly and get stuck in local optima. Tuning the hyperparameters of the policy gradient method can also be a challenging task, which can lead to poor performance. The flowchart of the policy gradient method is shown in Figure 2.6.

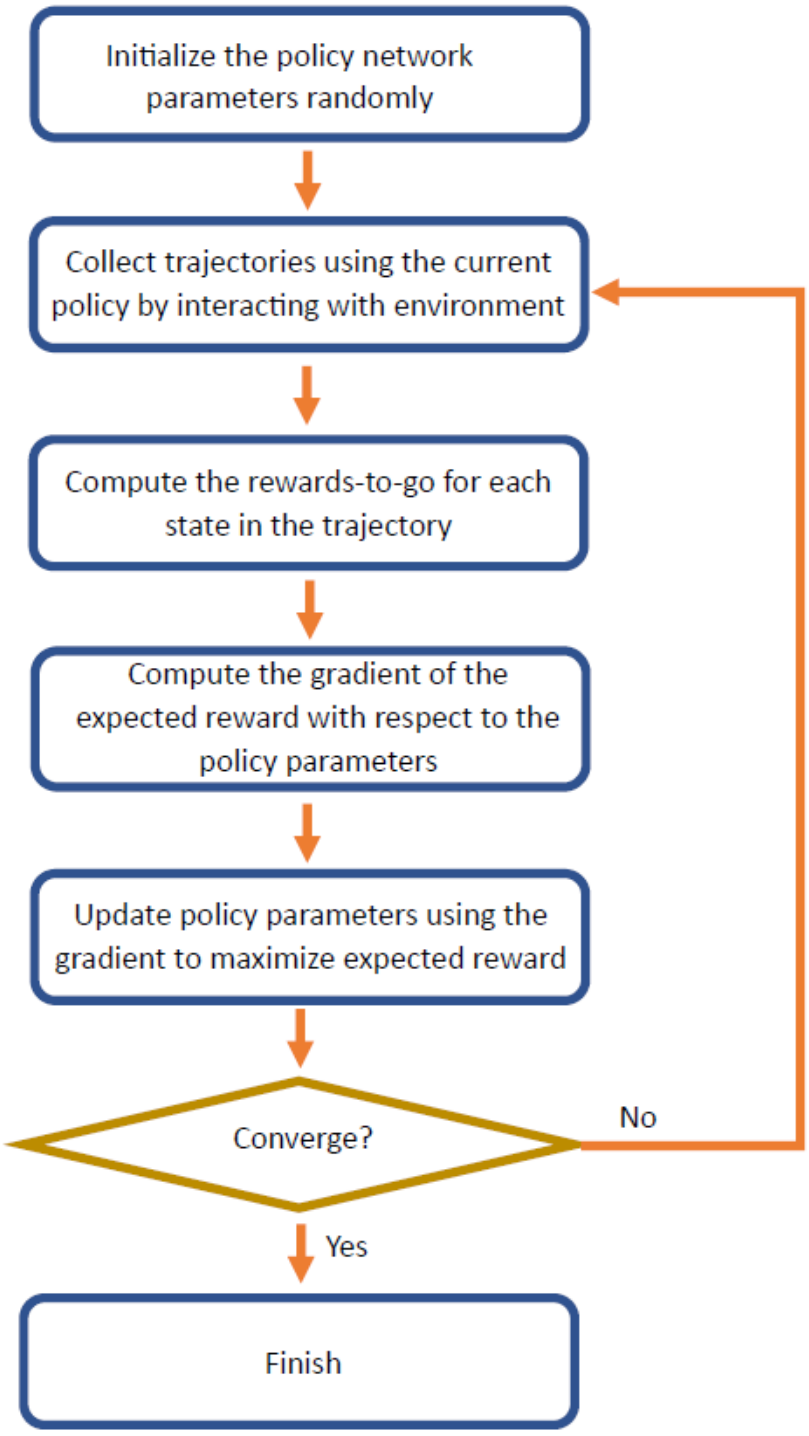


Figure 2.6: Policy Gradient Method.

2.3.5 Actor-critic Method

Algorithm 2.4 Actor-critic Method

1: Initialize actor training parameter η_0, critic training parameter ξ_0, critic learning rate α, critic network g, policy function π, actor learning rate β, initial state s_0, fixed behaviour policy μ, discounted factor ρ, maximum time limit $\hat{T}$, maximum episode limit $\hat{N}$, exploration noise O_t.
2: **for** episode = 1, $\hat{N}$ **do**
3: Reset the environment.
4: Receive initial state s_1.
5: **for** t = 1, $\hat{T}$ **do**
6: Select action $a_t = \mu(s_t) + O_t$ for action exploration according to the fixed behaviour policy and the exploration noise.
7: Execute a_t, calculate reward r_{t+1} and get new state s_{t+1}.
8: Store transition (s_t, a_t, r_t, s_{t+1}) in the replay buffer R_p.
9: Sample a minibatch (s_t, a_t, r_t, s_{t+1}) from transitions in R_p.
10: Compute temporal difference error:
$\kappa_t = r_{t+1} + \rho g_\xi(s_{t+1}, a_{t+1}) - g_\xi(s_t, a_t)$.
11: Update critic training parameter:
$\xi_{t+1} = \xi_t + \alpha\kappa_t \nabla_\xi g(s_t, a_t)$.
12: Update actor training parameter:
$\eta_{t+1} = \eta_t + \beta \nabla_\eta \pi(s_t) \nabla_a g_\xi(s_t, a)|_{a=\pi_\eta(s)}$.
13: **end for**
14: **end for**

Actor-critic method [338] is a popular RL algorithm that combines the benefits of both policy-based and value-based methods. This algorithm consists of two components: the Actor and the Critic. The Actor is responsible for selecting actions based on the current state, while the Critic evaluates the action taken by the Actor by estimating the value function. The value function provides feedback to the Actor, which is then used to improve the policy. Compared to pure policy-based or value-based methods, actor-critic method is more efficient because it utilizes the value function to decrease the variance of the policy gradient. This is achieved by subtracting a baseline from

the action-value estimate to get an advantage estimate, which reduces the gradient estimate's variance. Actor-critic method can also be used with function approximation techniques, such as deep NNs, to learn complex policies and value functions.

One of the main strengths of the actor-critic method is its ability to learn from experience, which allows it to adapt to changing environments and tasks. Moreover, it can efficiently handle high-dimensional state and action spaces, making it suitable for many real-world applications. However, the actor-critic method is sensitive to the choice of hyperparameters, and can suffer from instability and convergence issues if not properly tuned. Additionally, the use of function approximation techniques may lead to overfitting. Actor-critic method can be briefly described in Algorithm 2.4.

2.3.6 Deep Deterministic Policy Gradient Method

Deep Deterministic Policy Gradient (DDPG) [349] is a popular algorithm in RL that can handle continuous action spaces, which are often encountered in robotics and control applications. DDPG is an actor-critic method that combines the advantages of policy gradient and Q-learning. DDPG is an off-policy algorithm that uses experience replay to learn from previous experiences. The agent stores experiences in a replay buffer and randomly samples them to train the networks. DDPG has demonstrated strong performance in various environments and is a powerful tool for addressing complex problems in continuous action spaces.

One of the major advantages of the DDPG algorithm is its ability to handle continuous action spaces, which are challenging for discrete action algorithms such as Q-learning. Moreover, compared to other RL methods, DDPG is more sample-efficient and has a faster convergence rate. Despite these strengths, DDPG has several limitations. It is sensitive to hyperparameters and can be unstable during training, potentially leading to divergent policies. Additionally, DDPG requires significant memory to store the replay buffer, which may be a concern in resource-limited environments. A brief algorithm of how DDPG works is shown in Algorithm 2.5.

Algorithm 2.5 DDPG Method

Initialize actor network $\phi(s|\eta)$ with weight η, critic network $Q(s, a|\xi)$ with weight ξ, maximum time limit T, maximum episode limit N, replay buffer R, maximum size of the replay buffer R_m, target actor network ϕ' with weight $\eta' \leftarrow \eta$, target critic network Q' with weight $\xi' \leftarrow \xi$.
for episode = 1, N **do**
  Reset the environment.
  for t = 1, T **do**
    Initialize a random exploration noise o_t for action exploration.
    Choose action $a_t = \phi(s_t) + o_t$ according to the current policy and exploration noise o_t.
    Execute action a_t, calculate reward r_t and get new state s_{t+1}.
    Store (s_t, a_t, r_t, s_{t+1}) in the replay buffer R.
    Sample a minibatch (s_j, a_j, r_j, s_{j+1}) from R if $size(R) > R_m$.
    Set $y_j = r_j + \gamma Q'(s_{j+1}, \phi'(s_{j+1}|\eta')|\xi')$.
    Update critic by minimizing the loss:
    $L = \frac{1}{2}(y_j - Q(s_j, a_j|\xi))^2$
    Update actor with policy gradient ascent:
    $\nabla_\eta J = \mathbf{E}[\nabla_a Q(s, a|\xi)|_{s=s_j, a=\phi(s_j)} \nabla_\eta \phi(s|\eta)|s_j]$
    Update the target actor and critic networks:
    $\eta' \leftarrow \tau\eta + (1 - \tau)\eta'$
    $\xi' \leftarrow \tau\xi + (1 - \tau)\xi'$
  end for
end for

2.3.7 State-of-Art Algorithms

RL is a subfield of machine learning that focuses on training an agent to interact with an environment and learn from the feedback it receives in the form of rewards [115]. Over the years, various RL algorithms have been proposed to solve different problems, ranging from classic control tasks to complex games and robotics applications [350]. State-of-art RL algorithms typically have the ability to handle high-dimensional input spaces, model complex dynamics, and generalize well to unseen data [351]. Some of

the most prominent state-of-art RL algorithms include Soft Actor-critic (SAC) [352], Proximal Policy Optimization (PPO) [353], Trust Region Policy Optimization (TRPO) [354], Asynchronous Advantage Actor-critic (A3C) [355], and Twin Delayed Deep Deterministic Policy Gradient (TD3) [356], etc.

SAC [352] is a recent RL algorithm that has shown outstanding performance in continuous control tasks. This algorithm combines ideas from maximum entropy RL and soft Q-learning, and it is capable of learning optimal policies with high sample efficiency in continuous state and action spaces. SAC is an off-policy, model-free algorithm that maintains a stochastic policy and learns an approximation of the state-action value function by minimizing the mean squared Bellman error. One of the main characteristics of SAC is its use of the maximum entropy framework, which encourages the policy to explore more widely and learn more efficiently by maximizing an objective function that combines the expected cumulative reward with an entropy term. This results in a policy that is more robust and adaptive to changing environments. Furthermore, SAC uses a target network, similar to the one used in deep Q learning, to reduce the correlation between the Q-function updates and the policy updates. Additionally, SAC uses automatic entropy tuning to optimize the trade-off between exploration and exploitation, leading to better performance and faster convergence.

PPO [353] is another state-of-the-art RL algorithm that was introduced by OpenAI in 2017. PPO is an on-policy algorithm that directly optimizes the policy objective. PPO aims to maximize the expected cumulative reward obtained from executing a policy in a given environment, which is also less sensitive to hyperparameter choices. Additionally, PPO has a strong theoretical foundation and is guaranteed to converge to a locally optimal policy. PPO achieves its state-of-the-art performance through several key innovations. One of the most important is the use of a clipped surrogate objective, which helps to prevent the policy from deviating too far from the current policy distribution. Another key innovation is the use of a value function that is learned simultaneously with the policy. This helps to improve sample efficiency and reduce variance in the policy gradient estimates.

TRPO [354] is a policy optimization method that is commonly used in RL. TRPO is designed to find an optimal policy that maximizes the expected cumulative reward of an agent, subject to constraints. One of the key features of TRPO is that it is a trust region method, which means that it limits the amount of change to the policy during each update. This helps to prevent large policy updates that could potentially lead to instability or divergence in the learning process. By constraining the size of the update, TRPO ensures that the new policy remains close to the previous policy while still allowing for improvement. TRPO utilizes a surrogate objective function that approximates the performance of the true objective function, which is the expected cumulative reward. This surrogate objective function ensures that the new policy is always an improvement over the previous policy. Additionally, TRPO includes a constraint on the size of the policy update to ensure that the new policy is not too different from the previous policy. One of the limitations of TRPO is that it requires a large amount of data to achieve good performance, which can be a bottleneck in many applications.

A3C [355] is a state-of-art RL algorithm that combines Asynchronous and Advantage Actor-critic methods. The algorithm has multiple agents, each running an independent copy of the same NN. These agents are trained asynchronously, meaning that they are not synchronized with each other, and each agent interacts with the environment and updates the shared network independently. A3C utilizes an actor-critic architecture, with the actor network responsible for selecting actions and the critic network estimating the value of state-action pairs. To reduce the variance in the estimated value, A3C employs an Advantage function that estimates the advantage of taking a particular action in a given state. This feature helps to improve the stability of the algorithm. A3C offers several benefits, including its ability to handle high-dimensional state spaces such as images or sensor readings. It can also handle continuous action spaces, making it a promising choice for robotic control tasks. Additionally, the asynchronous training scheme used in A3C enables scalability and makes it well-suited for large-scale distributed systems. However, A3C also has some limitations. The algorithm requires a significant amount of training data to learn accurately and may require careful hyperparameter tuning to achieve optimal performance.

TD3 [356] is also a state-of-the-art RL algorithm that is designed to solve continuous control tasks. It is an extension of the original DDPG algorithm, which improves its stability and sample efficiency. TD3 employs a twin network architecture that consists of two NNs, both of which learn the same function but with different noise added to their outputs. The goal of this approach is to reduce overestimation of the Q-function, which is a common issue in Q-learning-based methods. In addition to the twin network architecture, TD3 also uses a delayed policy update mechanism. This means that the policy network is updated less frequently than the Q-network, which reduces the variance of the policy updates and improves the stability of the learning process. Furthermore, TD3 uses target policies for both the Q-function and the policy network. These target policies are updated slowly using a moving average of the main network weights, which ensures that the target values are more stable and less prone to changes. One of the primary advantages of TD3 is its sample efficiency. By reducing the overestimation of the Q-function and using delayed updates, TD3 requires fewer data than other algorithms to achieve good performance. Additionally, TD3 is capable of handling high-dimensional continuous action spaces, which is a common challenge in robotics and control applications. Despite its advantages, TD3 still suffers from the problem of local optima, which can limit its performance. Moreover, tuning the hyperparameters can be a challenging task.

State-of-art RL algorithms are continuously evolving, and researchers are constantly proposing new ideas and techniques to improve their performance and applicability. As the field of RL continues to grow, more advanced algorithms that can handle even more complex tasks and improve the efficiency and stability of RL-based systems will be carried out.

2.4 Consensus Control

Consensus control is a field of control theory that deals with the problem of coordinating a group of agents to achieve a common goal [357; 358]. The goal of consensus control is to develop control strategies that enable the agents to converge to a common state or behaviour through local interactions with their neighbours, without the need

for a centralized controller [359]. The problem of consensus control is motivated by a wide range of applications, including robotics [360], swarms [361], and social networks [362]. In these applications, the agents are often distributed and autonomous, and the communication among them is subject to delays [363], noise [364], etc.

The research on the issue of consensus can be traced back to 1974 when DeGroot [365] first proposed the concept of consensus. The problem of asynchronous consensus control and its application in distributed systems was first proposed in [366]. In 1987, Reynolds [367] proposed a "Boid" model that uses a computer program to simulate the motion behaviour of birds. In 1995, Vicsck [368] studied the motion behaviour of particle swarms and proposed the simplest swarm kinematics model. Later, Jadbabaiel [369] further discussed the "Vicsck" model and made the first theoretical research analysis on distributed consensus. The problem of consensus control has attracted significant attention from researchers over the past few decades, leading to the development of various control strategies [370]. Broadly, these strategies can be categorized into two groups: decentralized [371] and centralized [372] control. Decentralized control methods rely on local interactions between agents without the need for a central authority to coordinate their actions [373]. Centralized control approaches involve a central controller that collects and processes information from the agents and provides them with appropriate control signals [374].

Consensus control is a popular approach for MASs due to its desirable properties such as scalability, robustness, and adaptability [375]. However, there are still several challenges that need to be addressed to make consensus control more effective in real-world applications. One of the key challenges is the ability to design control strategies that can handle uncertainties and disturbances in the system, which can greatly affect the performance of the system [376]. Another important consideration is the scalability of the consensus control strategies, which can be limited by the communication bandwidth and computational resources available to the agents [377]. Moreover, the implementation of consensus control strategies in real-world applications can be challenging due to the need for reliable communication networks and accurate sensing and actuation devices [378]. In recent years, there have been quite a few satisfying results in

various types of consensus control [379; 380]. A first-order continuous-time integrator was used in [369] to describe the dynamic model of the agent. Moreaul [381] analyzed the consensus theory of discrete systems and proved the relationship between communication network connectivity, convexity conditions and consensus. Ren and Bread [382] proved that the sufficient condition for the first-order MAS to achieve consensus is that the fixed communication topology should have a directed spanning tree.

2.4.1 Kronecker Product

The Kronecker product [383] is a mathematical operation that combines two matrices to create a larger block matrix. The Kronecker product is particularly useful in representing and manipulating large, complex systems that can be decomposed into smaller subsystems [384]. The Kronecker product of matrices $X \in \mathbb{R}^{n\times m}$ and $Y \in \mathbb{R}^{p\times q}$ is defined as

$$X \otimes Y = \begin{bmatrix} x_{11}Y & x_{12}Y & \dots & x_{1m}Y \\ x_{21}Y & x_{22}Y & \dots & x_{2m}Y \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1}Y & x_{n2}Y & \dots & x_{nm}Y \end{bmatrix} \in \mathbb{R}^{np\times mq}$$

As stated in [383], important properties of the Kronecker product are listed in the following equations:

$$X \otimes (Y + Z) = X \otimes Y + X \otimes Z$$
$$(cX) \otimes Y = X \otimes (cY) = cX \otimes Y$$
$$(W \otimes X)(Y \otimes Z) = WY \otimes XZ$$
$$(X \otimes Y)^{-1} = X^{-1} \otimes Y^{-1}$$
$$(X \otimes Y)^{T} = X^{T} \otimes Y^{T}$$

where W, X, Y, and Z are the matrices with compatible dimensions for multiplication.

2.4.2 Graph Theory

Graph theory is a field of mathematics concerned with studying graphs, which are mathematical structures that are used to represent pairwise relationships between objects [385]. A graph consists of a set of vertices and a set of edges connecting pairs of

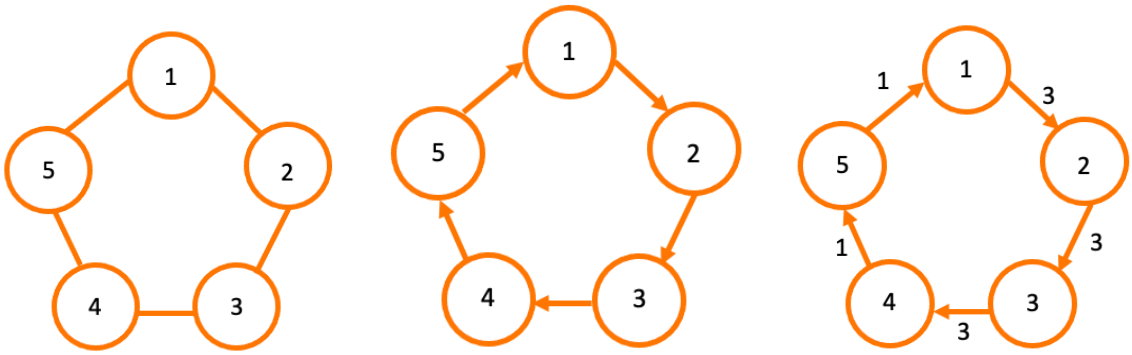


Figure 2.7: (a) An undirected graph (b) A directed graph (c) A weighted graph.

vertices [386]. The degree of a vertex, which is the number of edges that are incident to it, is a fundamental concept in graph theory [387]. Paths are another important concept in graph theory, which refer to a sequence of edges connecting a sequence of vertices [385]. Paths can be used to determine the connectivity and distance between vertices [388]. While graph theory is a powerful tool that has numerous applications, it also has limitations. One of the main challenges in graph theory is computational complexity, particularly when working with large-scale graphs [389].

Graphs are mathematical structures used to model pairwise relationships between objects [390]. There are several types of graphs, including undirected graphs, directed graphs, and weighted graphs [391]. An undirected graph is a graph where edges have no direction. That is, the relationship between the two nodes connected by an edge is symmetric, as shown in Figure 2.7 (a). When it comes to the directed graph, edges have a direction associated with them. This means that the relationship between two nodes connected by an edge is asymmetric, as depicted in Figure 2.7 (b). A directed graph is said to have a spanning tree if there exists at least one node having a directed path to all the other nodes [392]. A weighted graph [393] is a graph where edges have weights or values associated with them. These weights can represent various things, such as the distance between two nodes or the cost of traversing an edge.

The interaction topology of a MAS with N agents is depicted by a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ [394]. $\mathcal{V}$ is a vertex set $\mathcal{V} = \{1, 2, \cdots, N\}$. The edge set $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$. If p^{th} and k^{th} agents can share information with each other, $(p, k) \in \mathcal{E}$ [395]. If $(p, k) \in \mathcal{E}$,

$a_{pk} > 0$, where a_{pk} represents the connection strength [396]. Let the Adjacency matrix $\mathcal{A} = [a_{pk}] \in \mathbb{R}^{N \times N}$ of $\mathcal{G}$ be

$$[a_{pk}] = \begin{cases} a_{pk}, & p \neq k, \, (p,k) \in \mathcal{E}, \\ 0, & \text{otherwise.} \end{cases} \tag{2.31}$$

The degree matrix $\mathcal{D} \in \mathbb{R}^{N \times N}$ is a diagonal matrix which indicates the number of connections in each vertex [391]. The degree matrix $\mathcal{D}$ can be expressed as follows:

$$\mathcal{D} = \begin{bmatrix} d(v_1) & 0 & 0 & 0 \\ 0 & d(v_2) & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & d(v_N) \end{bmatrix} \tag{2.32}$$

where $d(v_p) = \sum_{k=0, k \neq p}^{N} a_{pk}$ stands for the degree of vertex.

The Laplacian matrix [397] is a fundamental tool in graph theory that describes the structure of a graph. It is a matrix that captures the relationship between the nodes in the graph and is defined as the difference between the degree matrix and the adjacency matrix [398]. The Laplacian matrix is a symmetric positive-semidefinite matrix. The Laplacian matrix's eigenvectors are important because they represent the graph's modes of vibration or oscillation, and the eigenvalues represent the frequencies of these modes [399]. The smallest eigenvalue of the Laplacian matrix, also known as the graph's algebraic connectivity, is a measurement of how well-connected the graph is. According to [391], the Laplacian matrix $\mathcal{L}$ of $\mathcal{G}$ can be defined as

$$\mathcal{L} = \mathcal{D} - \mathcal{A} \tag{2.33}$$

Lemma 1. *[391] If $\mathcal{G}$ contains at least one spanning tree, the Laplacian matrix $\mathcal{L}$ has one eigenvalue equal to zero with associated right eigenvector $\mathbf{1}_N$, where $\mathbf{1}_N = [1, \cdots, 1]^T$, and all the other eigenvalues have nonnegative real parts. Specially, $\mathcal{L}$ is positive semi-definite if the graph $\mathcal{G}$ is undirected.*

2.4.3 Consensus for a Single Integrator System

As stated in [391], the dynamics of the i^{th} agent with a single integrator system can be represented by

$$\dot{x}_i(t) = u_i(t) \tag{2.34}$$

where $x_i \in \mathbb{R}$ represents the state of each agent, and u_i denotes the control input of each agent.

Following [391], all the agents are said to achieve consensus if

$$\lim_{t\to\infty} \|x_i(t) - x_j(t)\| = 0, \quad \forall i, j \in \{1, \cdots, N\}. \tag{2.35}$$

Accordingly, the consensus protocol for a single integrator system in (2.34) can be designed as

$$u_i(t) = \sum_{i=1}^{N} a_{ij}(x_j(t) - x_i(t)). \tag{2.36}$$

As mentioned by [391], let the globe vector x be $x = [x_1, \cdots, x_N]^T$, the compact form of (2.34) under the consensus protocol (2.36) can be described as

$$\dot{x} = -\mathcal{L}x \tag{2.37}$$

Theorem 1. *[400] The consensus of a single integrator system in* (2.34) *can be ensured by the protocol* (2.36) *if and only if* $\mathcal{G}$ *has one spanning tree. Furthermore, denote* $p = [p_1, \cdots, p_N]^T$ *as normalised left eigenvalue of* $\mathcal{L}$ *for the zero eigenvalue, the consensus state* $\bar{x}$ *of each agent is given by*

$$\bar{x} = \sum_{i=1}^{N} p_i x_i(0) \tag{2.38}$$

where $x_i(0)$ *denotes the initial value of the* i^{th} *agent.*

2.4.4 Consensus for a Linear Time-Invariant System

Considering a linear time-invariant system with N agents, the dynamics of the i^{th} agent can be expressed as

$$\dot{x}_i(t) = Ax_i(t) + Bu_i(t) \tag{2.39}$$

where $x_i \in \mathbb{R}^n$ represents the state of each agent, and $u_i \in \mathbb{R}^m$ denotes the control input of each agent. $A \in \mathbb{R}^{n\times n}$ and $B \in \mathbb{R}^{n\times m}$ are constant matrices.

According to [401], the consensus protocol for single integrator system in (2.39) can be designed as

$$u_i(t) = cK \sum_{i=1}^{N} a_{ij}(x_i(t) - x_j(t)), \tag{2.40}$$

where $c \in \mathbb{R}^+$ is the positive coupling gain, and $K \in \mathbb{R}^{m \times n}$ denotes the feedback gain matrix.

Define the globe vector x as $x = [x_1^T, \cdots, x_N^T]^T$, the compact form of (2.39) under the consensus protocol (2.40) can be described as

$$\dot{x} = (I_n \otimes A + c\mathcal{L} \otimes BK)x \tag{2.41}$$

Theorem 2. *[402] Suppose $\mathcal{G}$ has at least one spanning tree, The consensus of the linear time-invariant system* (2.39) *can be ensured by the protocol* (2.40) *if c and K are selected such that*

$$A + c\lambda_i BK, \quad i = 1, 2, \cdots, N \tag{2.42}$$

are Hurwitz, where λ_i are the nonzero eigenvalues of $\mathcal{L}$. Furthermore, denote $p = [p_1, \cdots, p_N]^T$ as normalised left eigenvalue of $\mathcal{L}$ for the zero eigenvalue, the consensus state $\bar{x}$ of each agent is given by

$$\bar{x} = (p^T \otimes e^{At})x(0) \tag{2.43}$$

where $x(0) = [x_1^T(0), \cdots, x_N^T(0)]^T$ denotes the initial value of each agent.

2.5 Summary

Robot kinematics, motion planning, RL, and consensus control are all important research areas in robotics and control. Despite significant progress in these areas, there are still many research gaps to be addressed.

FK is a widely used method in robotics to determine the position and orientation of the end-effector. FK offers several benefits, including simplicity, speed, and ease of implementation in real-time applications. However, FK suffers from limitations in terms of accuracy and precision. Despite the significant advancements in modelling and computation techniques, the precision of the joint angle and position sensors is still a limiting factor in the accuracy of the FK model. Additionally, environmental factors, such as temperature, can also affect the accuracy of the FK model.

IK is a fundamental component of robotics that enables the calculation of joint angles or positions necessary for a robotic system to achieve a desired end-effector position or orientation. Compared to FK, which determines the end-effector's position and orientation given the joint angles or positions, IK provides greater flexibility and enables the robot to perform more complex tasks. However, IK problems can be challenging due to the existence of multiple solutions and non-solutions, which can lead to difficulties in finding the optimal solution. Therefore, finding efficient and reliable methods to handle multiple solutions and non-solutions is a critical research gap.

Motion planning is a crucial aspect of robotics research that seeks to create efficient and safe algorithms for planning the motion of robots within a given environment. This technology has the potential to optimize robot trajectories for improved performance, reduced errors, and increased safety. However, the present state-of-the-art algorithms for motion planning can be computationally demanding and may require a significant amount of time to converge. Moreover, the algorithm may struggle if the environment contains uncertainty and incomplete information.

RL is a prominent research area in machine learning that focuses on developing algorithms and methodologies for training agents to make sequential decisions in an environment. RL does not require labelled training data, and the agent learns from interactions with the environment, making it suitable for situations where manual labelling is either expensive or impractical. Additionally, RL has the capability to adapt to changes in the environment and can generalize to new situations. Despite its advantages, RL still faces several challenges that need to be addressed. One significant challenge is the trade-off between exploration and exploitation, where the agent must balance between learning from new experiences and exploiting its current knowledge to make decisions. Another challenge is the slow convergence of RL algorithms, which can be a significant issue in large-scale problems with high-dimensional state spaces. Additionally, RL algorithms can suffer from high variance and instability during training, which can significantly affect the overall performance of the agent.

Consensus control is a critical area of research in control theory that deals with the

coordination of agents in networked systems to reach an agreement or consensus. The primary objective is to develop control strategies that allow the agents to converge to a common state or decision despite the presence of communication constraints or external disturbances. One of the main advantages of consensus control is its ability to achieve cooperation and coordination among agents in a decentralized manner, which can lead to improved system performance, scalability, and robustness. Moreover, consensus control is flexible, allowing for the addition or removal of agents without significantly affecting the overall system behaviour. Nonetheless, there are several challenges to using consensus algorithms to handle uncertainties and disturbances in the system. Additionally, optimizing consensus control algorithms for large-scale systems can be difficult due to limited computational resources and communication bandwidth.

To sum up, the continued development and integration of robot kinematics, motion planning, RL, and consensus control will play a critical role in advancing the field of robotics. There is a need for combining consensus control with other fields such as RL to address more complex and dynamic systems.

Chapter 3

A Deep Reinforcement Learning Approach for Robotic Manipulators

When dealing with path planning problems, standard methods are always considered as the primary solution. However, how to avoid the occurrence of jerks in path planning remains an open question. Therefore, a vital question will be ensuring the smoothness of the manipulator trajectory while accomplishing path planning tasks. In order to overcome the difficulties encountered in a path planning problem, RL techniques have been widely used to solve challenging problems in engineering, computer science and robotics [333]. However, the problem of solving path planning problems with a DRL method is not considered in the aforementioned works.

The goal of this chapter is to propose a new off-policy DRL method to deal with the problem of path planning of the UR5 robot arm. Different from standard path planning methods, this method is able to guarantee a smooth movement of the UR5 robot arm, and all joint angles of the UR5 robot arm lie within the allowable range during each movement. Moreover, a standard path planning method has been implemented on the real UR5 robot arm as a baseline to compare and contrast the benefits and drawbacks of both methods.

3.1 The Proposed Method

In this section, we present the reward design, action space, observation space, NN structure, and training details of the proposed method.

3.1.1 Reward Design

The reward should make the UR5 robot arm reach the target position smoothly while keeping the end effector orientation of the UR5 robot arm as straight as possible downwards. The reward function is given as follows:

$$r = r_d + r_o + r_a + r_k \tag{3.1}$$

where r represents the total reward, r_d stands for the distance reward, r_o represents the orientation reward, r_a is the arrive reward, and r_k stands for the smoothness reward. The distance reward r_d is computed as follows:

$$r_d = d_p - d_c \tag{3.2}$$

where d_p stands for the previous distance between the end effector position of the UR5 robot arm and the target position, and d_c denotes the current distance between the end effector position of the UR5 robot arm and the target position. The time step between d_p and d_c is set to 0.01 seconds. If d_p is smaller than d_c, the UR5 robot arm will receive a negative reward since it gets further to the target position.

The orientation reward r_o is given by

$$r_o = \begin{cases} r_{op} & \text{if } |o_e - o_d| < o_{th} \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

where r_{op} equals to a positive orientation reward, o_e denotes the current end effector orientation of the UR5 robot arm, o_d represents the desired end effector orientation of the UR5 robot arm which is as straight as possible downwards, o_{th} is the orientation threshold. By introducing r_o in the reward function, the end effector orientation of the UR5 robot arm is more likely to be as straight as possible downwards.

The arrive reward r_a can be calculated by

$$r_a = \begin{cases} r_{ap} & \text{if } |d_c| < r_{ah} \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

where r_{ap} equals to a positive arrive reward, and r_{ah} represents the distance threshold. Therefore, r_a equals a positive reward if $|d_c|$ lies within the distance threshold, or it remains 0.

The smoothness reward r_k can be computed by

$$r_k = \begin{cases} -r_j & \text{if } |r_d| > r_{th} \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

where r_j represents a positive smoothness reward, and r_{th} stands for the smoothness threshold. As a result, r_k will receive a negative reward if $|r_d|$ is above the smoothness threshold. By introducing r_k in the total reward, the smooth movement of the UR5 robot arm can be ensured.

3.1.2 Action Space and Observation Space

The shoulder pan joint of the UR5 robot arm is actuated in this training scenario, together with the shoulder lift joint, elbow joint and wrist 1 joint. As a result, the action space is a vector that contains the joint angle of the shoulder pan joint, shoulder lift joint, elbow joint and wrist 1 joint. The observation space includes the action space, the location of the elbow joint and wrist 2 joint, the distance between the elbow joint and the goal, the distance between the wrist 2 joint and the goal and the distance between the end effector and the goal.

3.1.3 Neural Network Structure

The proposed method is on the basis of actor-critic off-policy DRL method. The structure of the actor network is shown in Fig. 3.1 (a). The input of the actor network is a vector that contains 17 elements, which are the joint angle of shoulder pan joint, shoulder lift joint, elbow joint and wrist 1 joint, the location of the elbow joint and

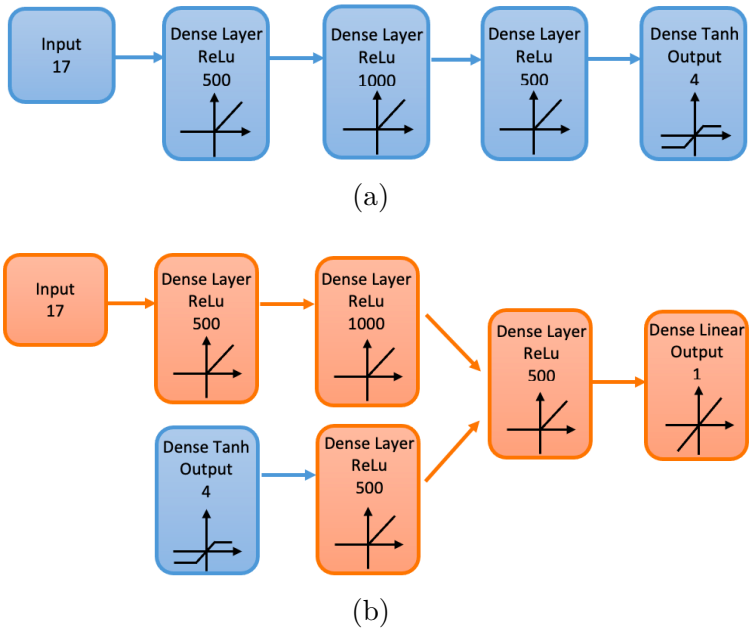


Figure 3.1: The architecture of actor network (a) and critic network (b).

wrist 2 joint, the distance between the elbow joint and the goal, the distance between the wrist 2 joint and the goal and the distance between the end effector and the goal. The input of the actor network is connected with three dense layers with ReLu [403] activation function, which also belongs to the input of the critic network architecture. The output of the actor network contains 4 elements, which are the joint angle of the shoulder pan joint, shoulder lift joint, elbow joint and wrist 1 joint. Tanh function is the activation function of the output of the actor network, which also belongs to the input of the critic network architecture, as depicted in Fig. 3.1 (b). The output of the critic is a Q value which is engendered by a linear activation function.

3.1.4 Training Details

The proposed method was trained in CoppeliaSim [404] simulation environment for efficiency. The process of the UR5 robot arm reaching the target position is demonstrated in Fig. 3.2. The UR5 robot arm is connected with a suction gripper. The experiment is considered to be successful if the UR5 robot arm is able to reach the position of the pink disc within the allowable range during each iteration. The first three pictures describe how the UR5 robot arm gradually reaches the target point within the allowable range, while the last three pictures depict another solution. Algorithm 3.1 details the proposed off-policy actor-critic based DRL method.

Algorithm 3.1 The proposed method

Initialize actor network $\phi(s|\eta)$ with weight η, critic network $Q(s, a|\xi)$ with weight ξ, maximum time limit T, maximum episode limit N, replay buffer R, maximum size of the replay buffer R_m, target actor network ϕ' with weight $\eta' \leftarrow \eta$, target critic network Q' with weight $\xi' \leftarrow \xi$.
for episode = 1, N **do**
  Reset the environment.
  for t = 1, T **do**
    Initialize a random exploration noise o_t for action exploration.
    Choose action $a_t = \phi(s_t) + o_t$ according to the current policy and exploration noise o_t.
    Execute action a_t, calculate reward r_t and get new state s_{t+1}.
    Store (s_t, a_t, r_t, s_{t+1}) in the replay buffer R.
    Sample a minibatch (s_j, a_j, r_j, s_{j+1}) from R if $size(R) > R_m$.
    Set $y_j = r_j + \gamma Q'(s_{j+1}, \phi'(s_{j+1}|\eta')|\xi')$.
    Update critic by minimizing the loss:
    $L = \frac{1}{2}(y_j - Q(s_j, a_j|\xi))^2$
    Update actor with policy gradient ascent:
    $\nabla_\eta J = \mathbf{E}[\nabla_a Q(s, a|\xi)|_{s=s_j, a=\phi(s_j)} \nabla_\eta \phi(s|\eta)|s_j]$
    Update the target actor and critic networks:
    $\eta' \leftarrow \tau\eta + (1 - \tau)\eta'$
    $\xi' \leftarrow \tau\xi + (1 - \tau)\xi'$
  end for
end for

3.2 Standard Path Planning Method

A standard path planning method was implemented in the real UR5 robot arm with Robot Operating System (ROS) [405]. The Open Motion Planning Library (OMPL) [406] is a widely used open-source software library for motion planning that provides a set of standard path planning methods. It is integrated into the ROS and can be used with various robotic platforms. OMPL includes a variety of algorithms for solving

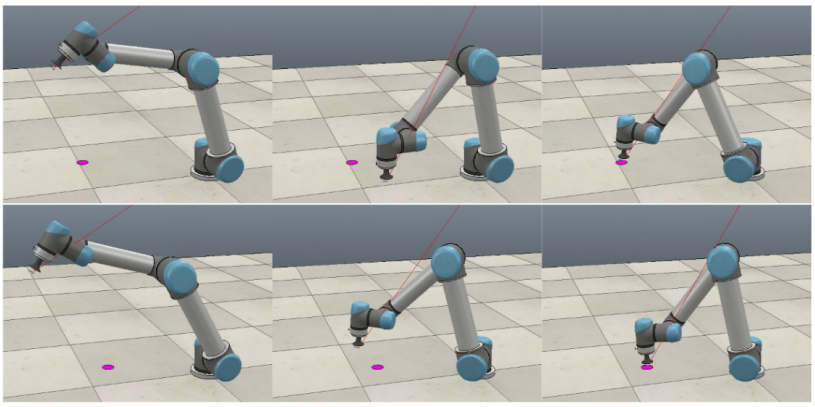


Figure 3.2: The process of the UR5 robot arm reaching the target position. The pink disc depicts the position of the target.

motion planning problems, such as the RRT algorithm and its variants, which have proven to be effective in complex and high-dimensional environments. Robotiq 3 finger gripper was tested in accomplishing pick and place tasks with 3D camera rs-visard. Fig. 3.3 shows the rviz [407] view of the actual UR5 robot arm, which demonstrates how the UR5 robot arm can accomplish pick and place tasks via the standard path planning method. To avoid potential crashes on the actual UR5 robot arm, OMPL [406] was used in rviz to move the actual UR5 robot arm to the position above the aruco marker, as shown in the first picture of Fig. 3.3. Then the actual UR5 robot arm went down to grasp the box first, then move the box to the target location and release the box in the end.

3.3 Result and Analysis

In this section, we demonstrate the evaluation on the training of the proposed method and compare the performance of the proposed method with the standard path planning method.

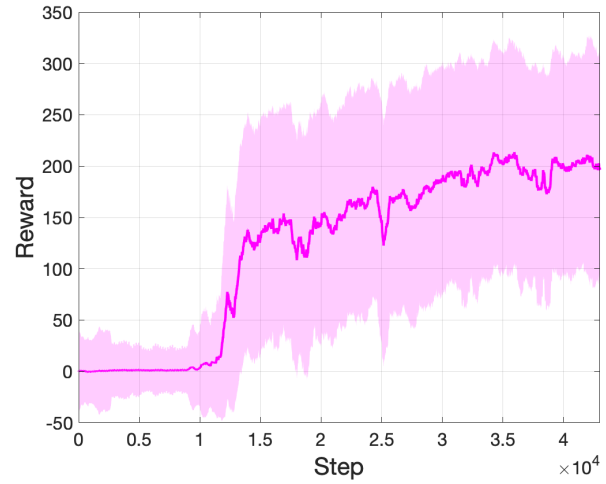
3.3.1 Evaluation on Training of the Proposed Method

Fig. 3.4 (a) demonstrates the average reward of the proposed method and Fig. 3.4 (b) describes the average Q value of the proposed method. The batch value is set to 512. Each average value is computed by averaging the results within a fixed batch size. The maximum episode limit is set to 215 and the maximum time limit is set

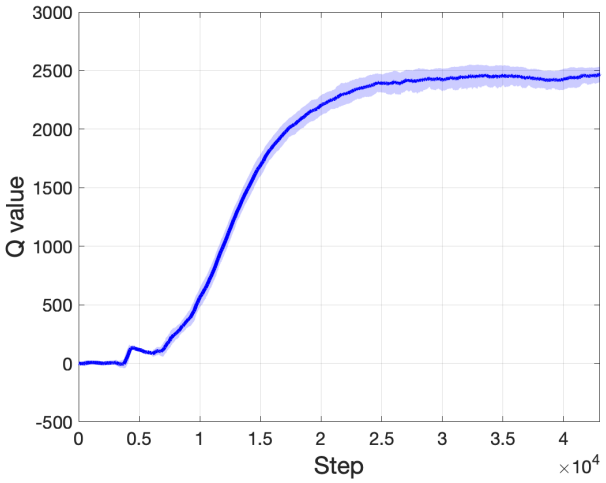
Figure 3.3: Pick and place tasks via the standard path planning method. The UR5 robot arm with orange colour denotes the initial position of the actual UR5 robot arm. The UR5 robot arm with grey colour represents the current position of the actual UR5 robot arm. The image viewer at the bottom right corner represents the view from the 3D camera. The location of the box can be computed by the aruco marker [1] on top of it.

to 200. The simulation of the UR5 robot arm is trained to arrive at the position of the target. In the initial stage, the robotic arm performs action exploration, so the reward at the beginning did not increase immediately. As can be seen from the figure, the reward and Q value get stable when it reaches 35000 steps, which indicates the off-policy actor-critic based DRL training is successful.

Fig. 3.5 shows solutions of reaching random target positions after training for 43000 steps. The aim of the proposed method is to train the UR5 robot arm to reach the target position (the pink disc) smoothly while keeping the end effector orientation of the UR5 robot arm as straight as possible downwards. The first and the last picture depict how the UR5 robot arm is trying to find path planning solutions at the right half plane, where the joint value of the shoulder pan joint, shoulder lift joint, elbow joint and wrist 1 joint are all within the allowable range. The second and the third picture depict how the UR5 robot arm is trying to find path planning solutions at the left half plane. It can be deduced from Fig. 3.5 that the proposed training method is feasible.



(a)



(b)

Figure 3.4: (a) Average reward of the proposed method. (b) Average Q value of the proposed method. The transparent area indicates the standard deviation of the results.

3.3.2 Comparison with Standard Path Planning Method

Unlike the standard path planning method, the proposed method can not only guarantee that the joint angle of the UR5 robotic arm is within the allowable range each time when it reaches the target point, but also ensure that the joint angle of the UR5 robotic arm is always within the allowable range during the entire episode of training.

Fig. 3.6 shows an example of failed path planning. It can be seen from the pictures in Fig. 3.6 that the planned trajectory generated by the standard path planning method contains a severe jerk, which makes it impossible to execute in the actual experiment.

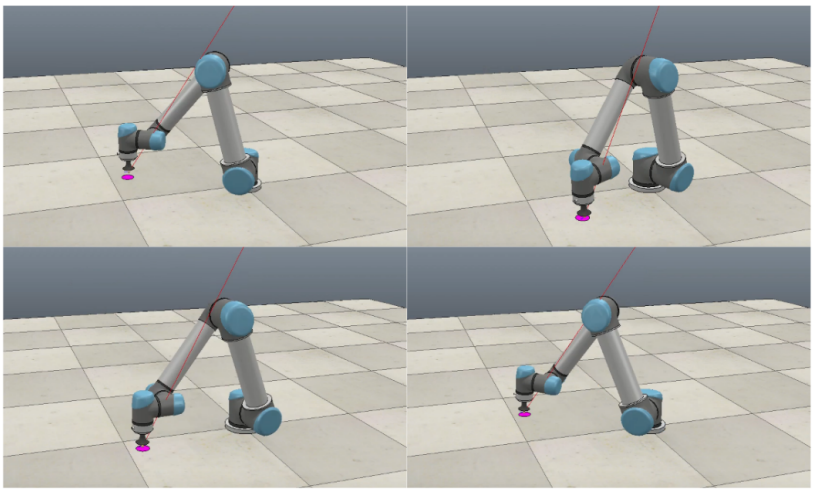


Figure 3.5: Solutions of reaching random target positions with the proposed method.

Table 3.1 depicts the method of successful rate comparison for path planning of both methods. If the planned trajectory does not contain any severe jerk, path planning is considered to be successful. It can be inferred that the standard path planning method failed to complete the task around 20% more than the proposed method. On one hand, the allowable joint angle range of both methods is different. With the proposed method, each joint on the UR5 robot arm has its unique allowable range, thereby saving training time and improving the training speed. Nevertheless, with the standard path planning method in ROS, all the joint range of the UR5 robot arm is set to $[-2\pi, 2\pi]$ due to various application environments. On the other hand, there exist multiple solutions when performing path planning with the standard method. This may lead to path planning joint solutions with large jerk manoeuvre. The total reward of the proposed method contains smoothness reward according to equation (3.1). By introducing the smoothness reward inside the total reward, it is more likely to reduce the occurrence of a large jerk manoeuvre when performing path planning with the proposed method. A minor drawback of the proposed method falls in the requirement of training before using it. However, this can be overcome by implementing the proposed method in the simulation environment.

Figure 3.6: Failed path planning with the standard path planning method. The orange UR5 robot arm stands for the initial position of the actual UR5 robot arm, the grey UR5 robot arm stands for the current position of the actual UR5 robot arm, and the transparent UR5 robot arm denotes the planned trajectory for the actual movement of the real UR5 robot arm generated by OMPL in rviz.

Table 3.1: Algorithm comparisons for path planning.

Success rate (%)	10 trials	20 trials	40 trials
Standard method	80	75	77.5
The proposed method	100	100	100

3.4 Summary

An off-policy actor-critic based DRL method is proposed to solve the problem of path planning. The simulation results in CoppeliaSim validated the feasibility of the proposed method. When it comes to the hardware implementation, a standard path planning method has been implemented as a baseline to compare and contrast both methods. It can be deduced that the proposed method can generate a smooth trajectory and keep the joint angles of the UR5 robot arm always within the allowable range. In future work, the proposed method will also be implemented on the real UR5 robot arm. Besides, vision information can be applied in the proposed method to accomplish more complicated tasks.

Chapter 4

Deep Reinforcement Learning with Manipulators for Pick-and-place

In the last chapter, we develop an off-policy DRL method to solve the path planning problem of the UR5 robot arm. In this chapter, we focus on real-world applications and propose a self-supervised vision-based DRL method for manipulators to learn to pick and place objects.

Transferring a DRL model from simulation to the real world is difficult because it depends on several conditions. Since the simulation cannot imitate the real world very well in many cases, fine-tuning usually requires lots of time to adapt simulated model parameters to real environments. However, the time to use a robot arm in a real environment is limited. Therefore, a vital question will be to reduce the required real-world fine-tuning time while maintaining high accuracy when picking and placing objects in the real environment.

With the emergence of robotic technology, robots have witnessed innovations in goal reaching [224; 231], picking and placing objects [408; 246], formation tracking [409], human-robot interaction [410], collision avoidance [233; 102], path planning [243; 411], etc. DRL has been widely used in robotic applications as an important component in robotic control. Although there is no instruction for the agent in the learning process, it must gradually generate a certain strategy by interacting with the environment to obtain the maximum reward. Compared to using traditional methods to pick and

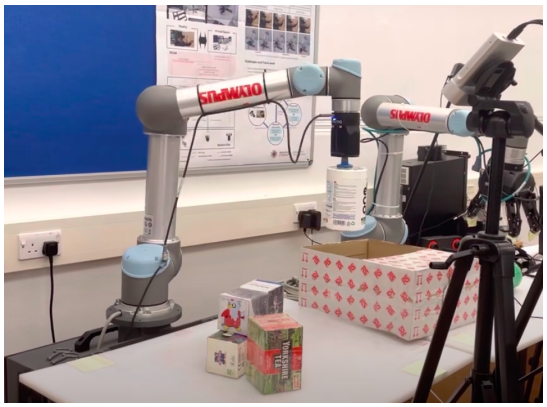


Figure 4.1: Pick-and-place objects with the proposed method

place objects, the challenges of using DRL are 1) feature extraction, for instance, using NNs to extract features from visual information to suction objects, 2) novel objects generalisation, such as suctioning objects with various shapes and heights, 3) data collection, for example, developing a self-supervised approach to avoid pre-labelling training data, 4) challenging environments adaptation, such as suctioning crowded and stacked objects. We introduce a self-supervised end-to-end DRL approach that allows robots to pick and place objects effectively and efficiently when directly transferring a training model from simulation to the real world, as shown in Fig. 4.1.

The main goal of this chapter can be summarized as follows. A complete self-supervised vision-based DRL method is proposed for manipulators to learn to pick and place objects. By encouraging the UR5 robot arm to suction the area close to the centre of target objects, the training model with the proposed method is able to accomplish pick-and-place tasks in the real world with a suction success rate of 90% without any real-world fine-tuning. Specially, a height-sensitive action policy is developed for the proposed self-supervised vision-based DRL method to suction in a challenging environment, i.e., crowded and stacked objects. The performance of the proposed method is validated in both simulated and real environments. The presented approach can also be applied to novel objects with a suction success rate of 90% without any fine-tuning from the real world.

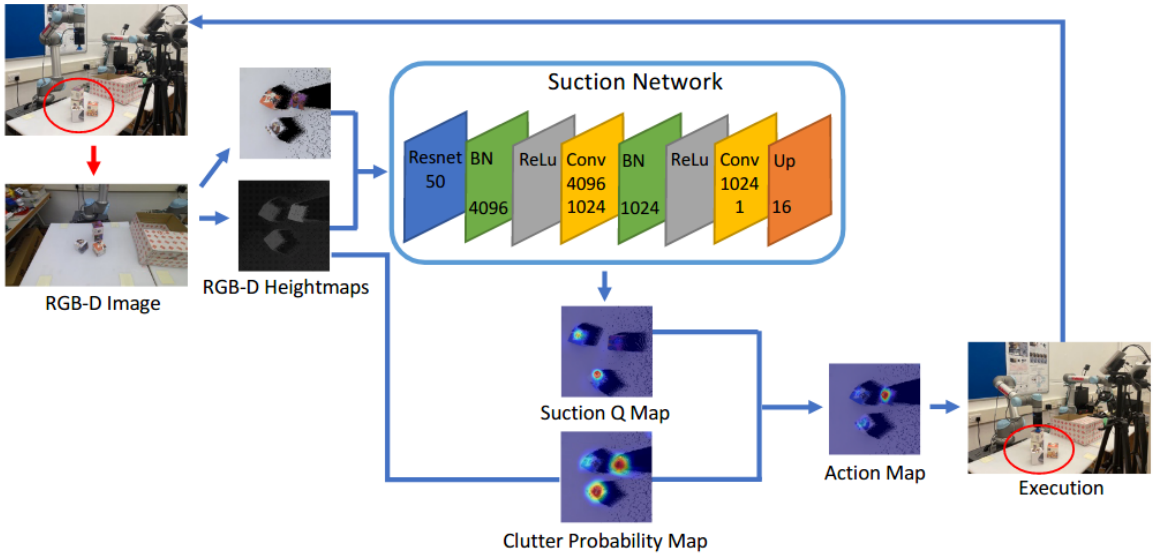


Figure 4.2: Overview of the proposed framework. BN stands for Batch Normalization. Conv represents convolution. Up stands for upsampling. The red circle denotes the pixel-wise best suction position. More details can be found from Algorithm 4.1.

4.1 The Proposed Method

The proposed method is completely trained under self-supervision through the interactions between the UR5 robot arm and the simulated environment. In this section, we present the system overview, reward space, NN structure, state space, and height-sensitive action policy of the proposed method.

4.1.1 System Overview

The overview of the proposed DRL framework is shown in Fig. 4.2. The Red Green Blue-Depth (RGB-D) image g_t, captured by a fixed camera, is orthographically projected in the gravity direction to construct the colour heightmap c_t and the depth heightmap d_t. Then both heightmaps are fed into the suction network to generate a suction Q map q_t. By detecting different heights from d_t, a clutter probability map l_t can be obtained. The position with the highest probability in the action map denotes the pixel-wise best suction position $[x_t, y_t]$. The suction height z_t is obtained from d_t.

The use of depth information alone can be inadequate in certain scenarios, such as those with low light or reflective surfaces, where the data obtained can be ambiguous or incomplete. By supplementing depth data with RGB images, the resulting data

can provide a more comprehensive and precise representation of the objects and environments being analyzed. Moreover, RGB images can contribute to object recognition and segmentation tasks by providing colour information. This integration of RGB and depth data can lead to improved performance and reliability in computer vision applications.

4.1.2 Reward Space

For each suctioned object, the centre distance ψ_t can be computed by

$$\psi_t = \sqrt{(x_t - \sigma_t)^2 + (y_t - \iota_t)^2} \tag{4.1}$$

where σ_t and ι_t denote x, y positions of the centre of the suctioned object.

The reward function can be defined as follows:

$$r = \frac{r_p}{(\psi_t + \delta)} r_g \tag{4.2}$$

where r_p is a positive constant reward, δ is a small positive number which prevents zero division, $r_g = 1$ if the object is successfully suctioned, otherwise $r_g = 0$. As a result, the reward in the simulated environment stimulates agents to suction the area close to the centre of the expected suctioned object, which increases the suction success rate.

4.1.3 Neural Network Structure

As shown in Fig. 4.2, the input of the suction network passes data through ResNet-50 [412] to extract features from both heightmaps. Then the aforementioned features are fed into a Batch Normalization layer [413] with 4096 input features, a ReLu layer [413], a Convolution layer [413] with 4096 input channels, and 1024 output channels. After passing data through another Batch Normalization layer [413], ReLu layer [413] and Convolution layer [413], data are processed by a bilinear upsample layer [413] with a scale factor of 16. The output of the suction network shares the same image size as the heightmap input, which is a dense pixel-wise Q map. The pixel with the highest probability in the action map denotes the best suction position.

Algorithm 4.1 Vision-based DRL for Pick-and-place

Initialize training parameter ϕ_t, learning rate α, RGB-D image g_t, discounted factor γ, training steps parameter T, replay buffer R_p.
while $t < T$ **do**
  Generate c_t and d_t from g_t.
  Generate l_t from d_t.
  if object number $b_t <$ empty threshold **then**
    Feed c_t and d_t into the suction network with the height-sensitive action policy to get action-value function $Q(\phi_t, s_t, a_t)$.
    if $t > 2$ **then**
      Use $Q(\phi_{t-1}, s_{t-1}, a_{t-1})$ to generate r_t.
      Minimize the temporal difference error ξ_{t-1}:

$$y_{t-1} = r_t + \gamma \max_a (Q(\phi^-_{t-1}, s_t, a)).$$
$$\xi_{t-1} = Q(\phi_{t-1}, s_{t-1}, a_{t-1}) - y_{t-1}.$$

      Sample a minibatch from R_p for experience replay.
    end if
    Suction objects.
    Store (c_t, d_t, a_t) in R_p.
  else
    Reposition objects.
  end if
end while

4.1.4 State Space

The state space s_t contains the colour heightmap c_t and the depth heightmap d_t.

4.1.5 Height-sensitive Action Policy

To effectively suction in a challenging environment, a height-sensitive action policy is proposed. As can be seen from Fig. 4.2, q_t can be acquired from c_t and d_t. However, the information contained in q_t is not enough to make the proposed framework sensitive to the heights of the grasped objects. As a result, we introduce the clutter probability

map l_t. The depth heightmap is shifted along one axis for 60 pixels to generate a translated map. By comparing the depth difference between the translated and the original depth heightmap, the pixel with enough depth difference is counted as 1 otherwise 0, which builds the clutter probability map l_t. Therefore, the action space a_t can be computed as follows:

$$a_t = \arg\max_a (q_t l_t) \tag{4.3}$$

4.2 Experiments and Results

The feasibility of the proposed method is validated in both simulated and real environments. The proposed approach is implemented on a desktop with Nvidia GTX 2080 and Intel Core i9 CPU with 64 GB RAM. In this section, we demonstrate training details, evaluation metrics, baseline method, simulation evaluation and real-world evaluation of the proposed method.

4.2.1 Training Details

The proposed method is trained in Coppeliasim [404] using Python [414] and Pytorch [413]. The UR5 robot arm is connected with a suction gripper [415] to pick and place objects, as shown in Fig. 4.3. During each training iteration, a vision sensor captures RGB-D images of the UR5 robot arm in a 0.448×0.448 m^2 workspace. The resolution of the RGB-D images is 640×480. The UR5 robot arm motion planning task can be accomplished using Coppeliasim [404] internal IK. The suctioned objects are $5 \times 5 \times 5$ cm^3 cubes.

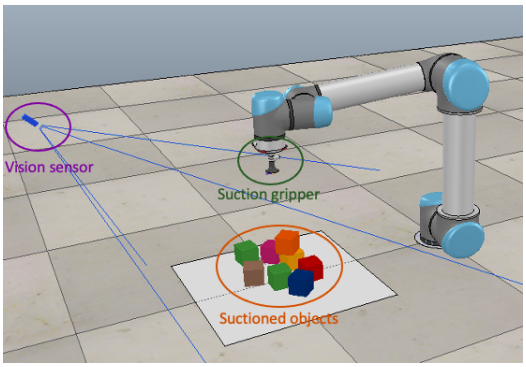


Figure 4.3: The training environment in simulation

Depending on the size of the suction gripper and the intrinsic of the vision sensor, we set $r_p = 15$ and $\delta = 0.00001$ in (4.2). For other robotic platforms, these values can also be reconfigured. In Algorithm 4.1, the learning rate α is set to 0.0001. The discounted factor γ has a fixed value of 0.5. The training steps parameter T is set to 400. The training is considered to be successful if the UR5 robot arm is able to pick and place target objects which are randomly dropped into the workspace.

4.2.2 Evaluation Metrics

We design two metrics to evaluate the suction performance of the UR5 robot arm. For all these metrics, a larger value leads to better performance.

The suction success rate S_r is given by

$$S_r = \frac{N_s}{N_i} \times 100\% \tag{4.4}$$

where N_s stands for the number of successful suctions, N_i represents the number of training steps.

The distance rate D_r is defined as follows

$$D_r = \frac{N_d}{N_i} \times 100\% \tag{4.5}$$

where N_d is the number of times when $\psi_t < 0.015$ m.

4.2.3 Baseline Method

The performance of our system is compared with the following baseline approach:

Visual Grasping method shares the same input as our proposed method to generate the probability maps for best suction positions. However, it takes binary classification for the reward space design in which 1 stands for successful grasp and 0 otherwise. This baseline method is analogous to the Visual Pushing Grasping (VPG) method [126]. Nevertheless, we extend this method to our suction framework for a fair comparison.

4.2.4 Simulation Evaluation

To confirm the validity of our design, we train both methods in simulation for 400 steps. Overall the proposed method outperforms the Visual Grasping method in terms

of both suction success rate and distance rate by large margins. It can be obtained from Fig. 4.4 that the proposed method arrives at around 97% suction success rate at 150 training steps, while the Visual Grasping method shows only 52%. When the height-sensitive action policy is removed from both methods, it takes longer for both methods to achieve the same suction success rate. As can be seen from Fig. 4.5, the distance rate of the proposed method reaches around 80% at 400 training steps, whereas the Visual Grasping method shows only 58%. When the height-sensitive action policy is separated from both methods, the distance rates are reduced by 20% and 33%, respectively. A pick-and-place demonstration with the proposed method is shown in Fig. 4.6. The simulation results confirm the validity of our reward space design which stimulates robots to suction the area close to the centre of the expected suctioned object, thereby improving the suction success rate.

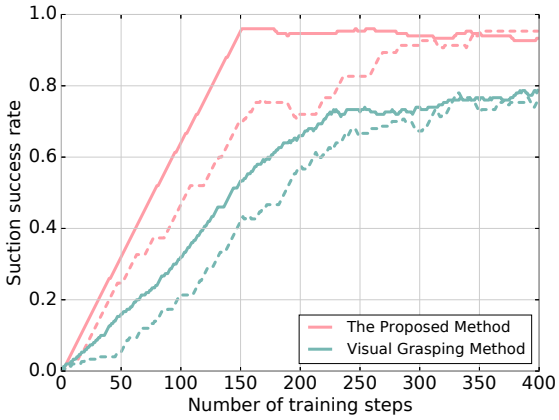


Figure 4.4: Suction success rates of the proposed method and the Visual Grasping method. The dotted lines represent methods without the height-sensitive action policy.

4.2.5 Real-world Evaluation

We evaluate both methods in the real environment using the models trained in Fig. 4.4. Each model is executed 10 times, and the maximum number of actions is 20 for each run. The UR5 robot arm is connected to a Robotiq EPick vacuum gripper for real-world evaluation. A fixed Azure Kinect camera is used to capture RGB-D images with a resolution of 1280×720. The suctioned objects are $7 \times 7 \times 7$ cm^3 cubes.

Fig. 4.7 depicts the box plot of real-world evaluation with the proposed method and the

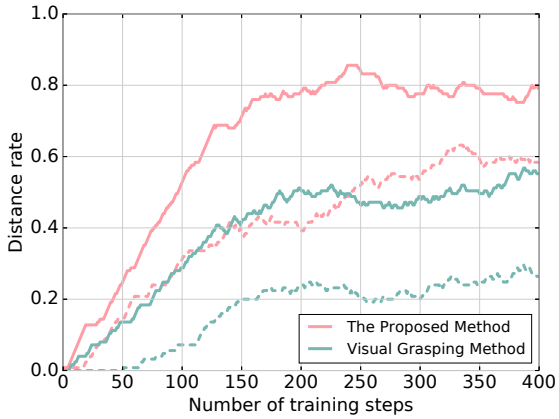


Figure 4.5: Distance rates of the proposed method and the Visual Grasping method. The dotted lines stand for methods without the height-sensitive action policy.

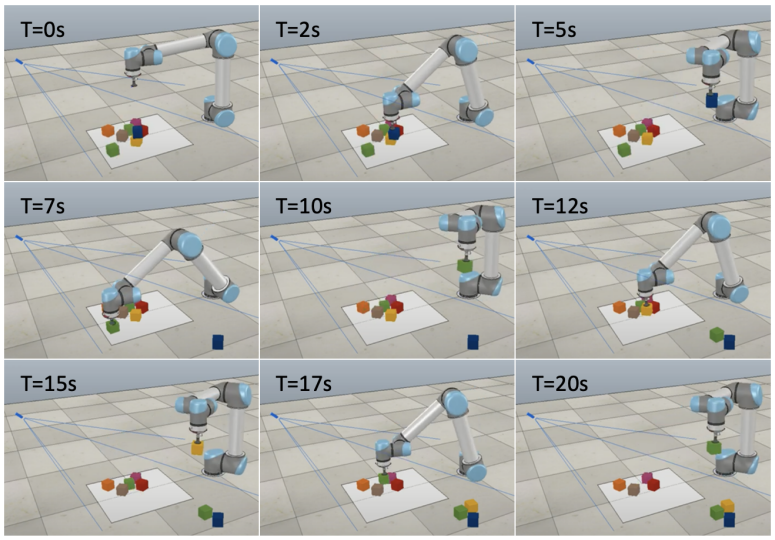


Figure 4.6: Pick-and-place demonstration in simulation with the proposed approach. The proposed method encourages the UR5 robot arm to suction the area close to the centre of the target objects with the height-sensitive action policy.

Visual Grasping method. The proposed method achieves a 90% suction success rate at 200 training steps in real-world evaluation, while the Visual Grasping method shows only 40%. When the height-sensitive action policy is separated from both methods, the suction success rates drop to 65% and 30%, respectively. By implementing the proposed method, the suction success rate gap between the simulation and the real environment is only 6%, much smaller than the gap using the Visual Grasping method (26%). Although the suctions are considered successful in Fig. 4.8, they will result in real-world failures because of edge suctioning, which enlarges the gap between the

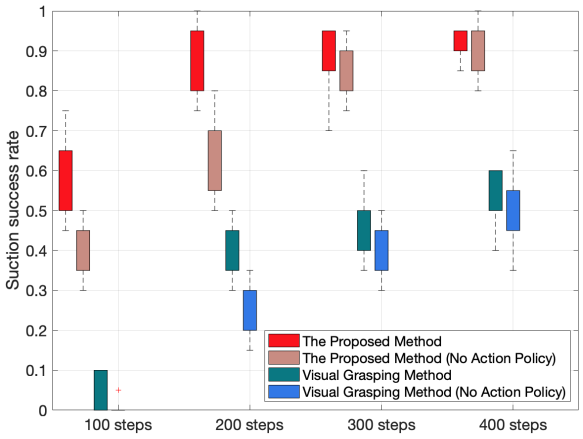
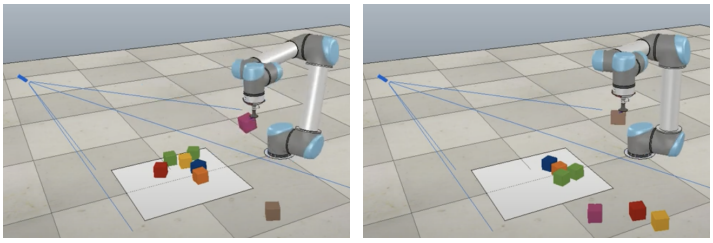


Figure 4.7: Real-world evaluation of the proposed method and the Visual Grasping method



(a) (b)

Figure 4.8: Edge suctioning with the Visual Grasping method. Although the suctions are considered successful in simulation, they will cause real-world failures.

simulation and the real world. The demonstration of real-world evaluation with the proposed method is shown in Fig. 4.9. By encouraging the UR5 robot arm to suction the area close to the centre of the target objects, the proposed method has a higher suction success rate than the Visual Grasping method.

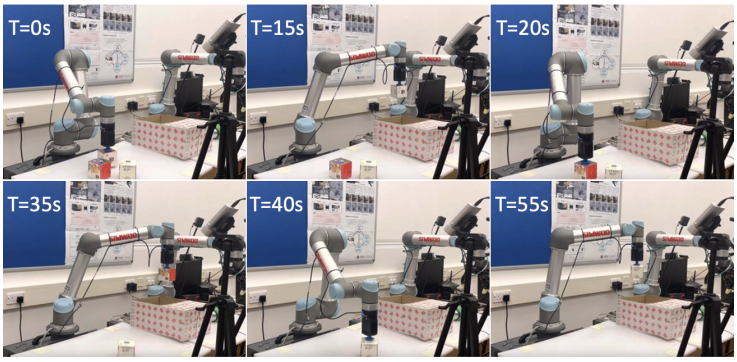


Figure 4.9: The demonstration of real-world evaluation with the proposed approach. The training model with the proposed method can be implemented directly to a real suction task without any fine-tuning from the real world.

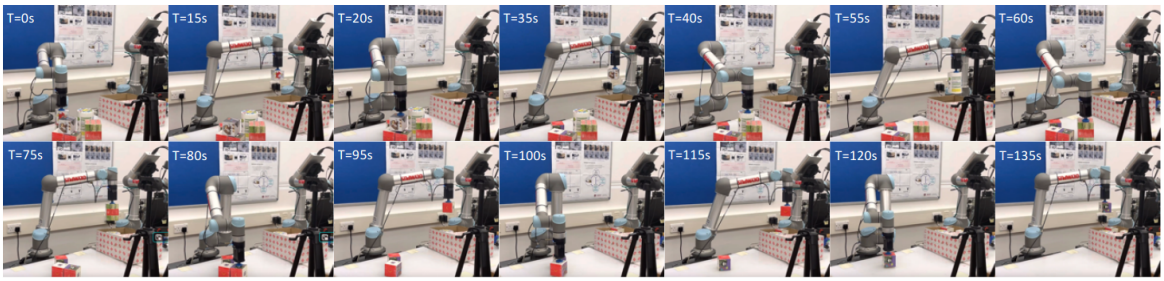


Figure 4.10: The demonstration of suctioning in challenging Environment 3

Table 4.1: Average performance in challenging environments

Collision rate (%)	Env 1	Env 2	Env 3
The proposed method	0	0	5
The proposed method (No Policy)	0	45	50
Visual Grasping method	0	1	9
Visual Grasping method (No Policy)	5	55.5	60

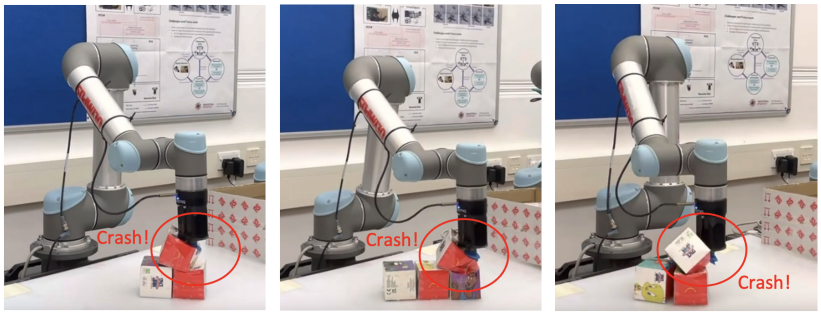
* No Policy means without the height-sensitive action policy.

4.3 Suction in Challenging Environments

The performance of the height-sensitive action policy in our proposed method is validated in this section. When humans try to pick and place crowded or stacked objects, they tend to grasp top objects first and then bottom objects since it's safer to do so. Inspired by this, we design a height-sensitive action policy that makes the UR5 robot arm take the heights of objects into consideration in order to avoid potential collisions when applying the presented approach. The testing environments are elucidated in Fig. 4.11. Environment 1 contains fully stacked objects. Environment 2 consists of half stacked objects, which is more challenging. Environment 3 is the most challenging environment which contains both half stacked objects and novel objects. As shown in Fig. 4.10, the proposed method is able to handle crowded and stacked objects in a safe manner. It can be obtained from Table 4.1 that the more challenging the environment is, the more effective the height-sensitive action policy is. If the height-sensitive action policy is removed from both methods in Environment 3, the collision probability will increase by 45% and 51%, respectively. Some failed examples are shown in Fig. 4.12, which are due to the fact that the UR5 robot arm tries to suction the object below first rather than the object above, thus colliding with the object above. This confirms the necessity of the proposed height-sensitive action policy which ensures safety during the entire movement of the UR5 robot arm.

(a) (b) (c)

Figure 4.11: Suctioning in challenging environments: (a) Environment 1; (b) Environment 2; (c) Environment 3



(a) (b) (c)

Figure 4.12: Examples of collisions without the height-sensitive action policy

4.4 Real-world Unseen Objects Challenge

In this section, we validate the generalisation capability of our proposed vision-based DRL method. As shown in Fig. 4.13, novel objects contain cylinders of different heights as well as irregularly shaped objects. The proposed method can generalise to novel objects with a suction success rate of 90% without any real-world fine-tuning. More details can be seen in Fig. 4.14.

(a) (b)

Figure 4.13: Novel objects for real-world suctioning: (a) Environment 1; (b) Environment 2

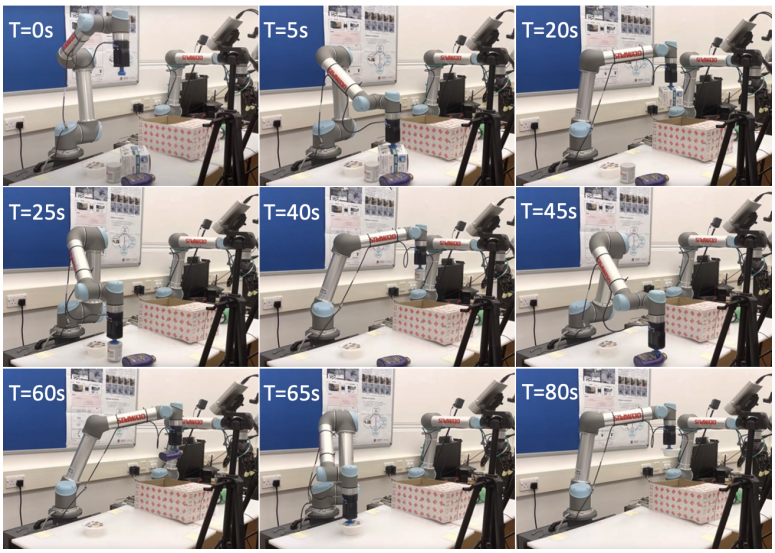


Figure 4.14: Pick-and-place novel objects with the proposed method

4.5 Summary

In this chapter, we propose a self-supervised vision-based DRL method to bridge the gap between simulated and real environments. Overall the proposed approach outperforms the Visual Grasping method in terms of both suction success rate and distance rate by large margins. By implementing the proposed approach, the suction success rate of the real robot achieves 90% at 200 training steps, while the Visual Grasping method shows only 40%. By implementing the height-sensitive action policy, the proposed method is able to pick and place crowded and stacked objects safely in challenging environments. Not only can our model be applied to the real experiment directly, but also it is capable of generalising to novel objects with a success rate of 90% without any fine-tuning from the real world. In the future, an optimisation of the proposed method will be explored to handle more complicated scenarios.

Chapter 5

Distributed Neural Networks Training for Robotic Manipulation with Consensus Algorithm

In Chapter 3 and Chapter 4, we pay attention to simulations and real experiments of single-agent DRL. In this chapter, we aim to propose an off-policy algorithm that combines multi-agent actor-critic based DRL with consensus-based distributed training. Compared to the policy gradient [416] method, the actor-critic based off-policy algorithm can perform single-step updates which lead to faster update speed. Additionally, the actor-critic based off-policy algorithm can select actions on a continuous action space whereas the Q learning [417] algorithm must have a discrete action space.

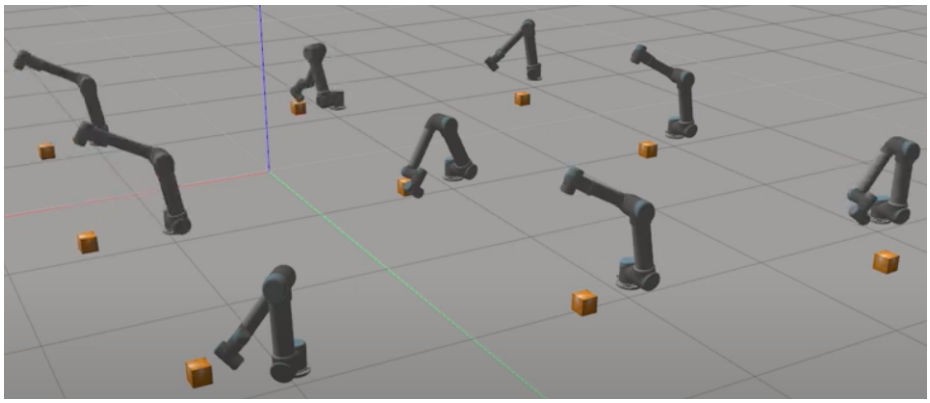


Figure 5.1: Training a group of nine UR5 robot arms to reach the random target positions. The targets are represented by wooden boxes.

Our work builds on the algorithm presented in [2], but our novel algorithm allows us to update both actor and critic parameters at each time step. As a result, the system is no longer a single excitation system and the consensus error of the actor network with our proposed method converges 50 times faster than the algorithm proposed in [2]. Furthermore, even if the consensus-based distributed training on the critic value training parameter is removed from our proposed algorithm, the consensus error of the actor network is still 25 times faster than the algorithm proposed in [2]. To guarantee convergence of the novel algorithm, a convergence analysis of a consensus algorithm for a type of nonlinear systems with a Lyapunov method is developed. We use this result to analyse the convergence properties of both actor and critic training parameters simultaneously in our algorithm.

To sum up, a multi-agent training algorithm with actor-critic based off-policy DRL and consensus-based distributed training is developed. The convergence of this algorithm is verified in the presence of the actor training parameter and the critic training parameter. Additionally, a multi-agent training framework is proposed to support the implementation of our algorithm. Compared with centralized training, the proposed multi-agent training framework has better scalability with a limited communication bandwidth when dealing with a large number of agents and protects the privacy of each agent. The feasibility and efficiency of the proposed algorithm are validated by experiments using several groups of UR5 robot arms, as shown in Fig. 5.1.

5.1 Problem Formulation

We develop an algorithm that combines multi-agent DRL with consensus-based distributed training. During the process of training, each agent performs training parameter sharing with other agents, which guarantees privacy protection.

5.1.1 Consensus-based Distributed Training

The interaction topology of the J agents is described by an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ which consists of a vertex set $\mathcal{V} = \{1, 2, \cdots, J\}$ and an edge set $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$. The edge

$(k, j) \in \mathcal{E}$ represents that the k^{th} and j^{th} agents are connected with each other. For an undirected graph $\mathcal{G}$, we suppose that τ_k denotes the training parameter vector for agent k in graph and $\tilde{\tau}_k$ denotes the updated training parameter of τ_k after a single consensus step. Therefore, the consensus training step of each agent can be written as

$$\tilde{\tau}_k = M\tau_k + u_k \tag{5.1}$$

$$u_k = \iota T \sum_{l=1}^{J} \hat{a}_{kl}(\tau_l - \tau_k) \tag{5.2}$$

where $M \in \mathbb{R}^{n\times n}$ and M is Schur, u_k is the input of the agent k, ι is the scalar coupling gain, T stands for the feedback gain, $\hat{a}_{kl}$ represents the elements of the graph adjacency matrix generated by the undirected graph $\mathcal{G}$.

The connectivity of the undirected graph $\mathcal{G}$ can be formalized in a weighted connectivity matrix $W = [w_{kl}] \in \mathbb{R}^{J\times J}$, where the value of the elements w_{kl} represent the strength of the connection between these agent k and agent l [418; 400]. So we have

$$w_{kl} \begin{cases} > 0 & \text{if agent k and agent l are connected.} \\ = 0 & \text{if agent k and agent l are disconnected.} \end{cases} \tag{5.3}$$

With the aim of reaching multi-agent consensus for any undirected graph, the weighted connectivity matrix W should meet the following requirements: 1) $w_{kl} \in [0, 1],\ \forall(k, l)$; 2) $w_{kl} = w_{lk},\ \forall(k, l)$; 3)$\sum_{l=1}^{J} w_{kl} \leq 1,\ \forall k$.

We can use the consensus algorithm to update an agent k in the following scheme by putting (5.2) into (5.1):

$$\begin{aligned} \tilde{\tau}_k &= M\tau_k + \iota T \sum_{l=1}^{J} \hat{a}_{kl}(\tau_l - \tau_k) \\ &= \mathcal{C}_k\left(\tau_l, w_{kl}\right) = \sum_{l=1}^{J} w_{kl}\tau_l \end{aligned} \tag{5.4}$$

where $\mathcal{C}_k$ denotes the consensus protocol of the k^{th} agent.

Let the training parameter matrices $\tilde{\tau}$ and τ represent the concatenation of training parameter vectors of all the J agents before and after the consensus process, respectively. We can show the update of training parameters of all the J agents with a single

consensus step as

$$\begin{aligned}\tilde{\tau} &= (I_J \otimes M - \iota L \otimes T)\tau \\ &= \mathcal{C}(\tau, W) = W\tau\end{aligned} \tag{5.5}$$

where $\mathcal{C}$ denotes the consensus protocol for all agents, L is the Laplacian matrix generated by the given undirected graph, and I_J is a $J \times J$ identity matrix. By repetitively computing (5.5), this algorithm allows all agents to converge to their average [418].

5.1.2 Actor-critic Based Off-policy Deep Reinforcement Learning

We consider single-agent training in this scenario. The state of the agent at the current time step t is represented by s_t and the action of the agent at the current time step t is denoted by a_t. Both the state s_t and the action a_t are continuous. The local reward of the agent at current time step t is defined by $r(s_t, a_t)$ and the state transition function of agent is denoted by $s_{t+1} = T(s_t, a_t, \sigma_t)$, where σ_t represents the noise in the state dynamics at time step t [2; 419].

The state value function $V^\pi(s)$ and the state action value function $Q^\pi(s, a)$ can be defined as follows [2]:

$$V^\pi(s) = \mathbb{E}[\textstyle\sum_{t=0}^{\infty} \rho^t r(s_t, a_t) | s_0 = s, \pi] \tag{5.6}$$

$$Q^\pi(s, a) = \mathbb{E}[\textstyle\sum_{t=0}^{\infty} \rho^t r(s_t, a_t) | s_0 = s, a_0 = a, \pi] \tag{5.7}$$

where $\pi(s) : s \longrightarrow a$ denotes the global deterministic policy function which is followed starting from this state and $\rho \in (0, 1)$ is the discounted factor.

The relationship of the state value function $V^\pi(s)$ and the state action value function $Q^\pi(s, a)$ is given by

$$V^\pi(s) = \mathbb{E}[Q^\pi(s, a), a = \pi(s)] \tag{5.8}$$

For all s, the goal of the actor-critic method is to find the optimal policy $\hat{\pi}(s) =$

$\arg\max_{\pi} f(\pi, s)$ of each agent to maximize infinite-time discounted-reward value function $f(\pi, s) = V^{\pi}(s)$. As stated in [338], if the parameterization is linear, the policy function $\pi_\eta(s)$ and the value function $g_\xi(s, a)$ can be described as:

$$\pi_\eta(s) = \eta^T \phi(s) \tag{5.9}$$

$$g_\xi(s, a) = \xi^T \psi(s, a) \tag{5.10}$$

where $\eta \in \mathbb{R}^k$ denotes the actor training parameter, $\xi \in \mathbb{R}^k$ represents the critic training parameter. $\phi(s)$ and $\psi(s, a)$ denote the feature functions that respond to the policy function and the value function, respectively.

Under (5.9), the optimization problem can be converted to the parameterized policy optimization problem

$$\max_{\eta} f(\pi_\eta(s), s) \tag{5.11}$$

which can be solved by gradient descent methods. According to [420], $\nabla_\eta f(\pi_\eta(s), s) = \mathbb{E}[\nabla_\eta \pi(s) \nabla_a Q^{\pi}(s, a) | a = \pi_\eta(s)]$.

Since the function $Q^{\pi}(s, a)$ is unknown, according to [421; 348], $g_\xi(s, a)$ can be used to represent the closest approximation of $Q^{\pi}(s, a)$. Hence, $g_\xi(s, a) \approx Q^{\pi}(s, a)$. The expression of $\nabla_\eta f(\pi_\eta(s), s)$ can be rewritten as $\nabla_\eta f(\pi_\eta(s), s) = \mathbb{E}[\nabla_\eta \pi(s) \nabla_a g_\xi(s, a)|_{a=\pi_\eta(s)}]$.

As stated in [303; 422], the process of updating the critic training parameter and the actor training parameter are given as:

$$\kappa_t = r_{t+1} + \rho g_\xi(s_{t+1}, a_{t+1}) - g_\xi(s_t, a_t) \tag{5.12}$$

$$\xi_{t+1} = \xi_t + \alpha \kappa_t \nabla_\xi g(s_t, a_t) \tag{5.13}$$

$$\eta_{t+1} = \eta_t + \beta \nabla_\eta \pi(s_t) \nabla_a g_\xi(s_t, a)|_{a=\pi_\eta(s)} \tag{5.14}$$

where κ_t stands for the temporal difference error, ξ_{t+1} and η_{t+1} represent the updated critic training parameter and the actor training parameter, respectively. α and β denote the critic learning rate and the actor learning rate, respectively. From (5.10), we have $\nabla_\xi g(s_t, a_t) = [\psi(s, a) | \xi = \xi_t, s = s_t, a = a_t]$.

As discussed in [420], we make the following assumptions:

Assumption 1. *For any deterministic policy function $\pi_\eta(s)$, there always exists a compatible function approximator of the form*

$$g_\xi(s,a) = (a - \pi_\eta(s))^T \nabla_\eta \pi(s)^T \xi + V^v(s) \tag{5.15}$$

where $g_\xi(s,a)$ can be used to represent the closest approximation of $Q^\pi(s,a)$. $V^v(s)$ can be any differentiable baseline function that is independent of the action a. For instance, $V^v(s)$ can be $v(s)^T\tilde{\xi}$, where $v(s)$ represents the state feature function.

5.1.3 Actor-critic Based Off-policy Deep Reinforcement Learning with Consensus-based Distributed Training

Before we state our main result with J agents, we make the following assumptions:

Assumption 2. *All agents share the same training task and scenario.*

Assumption 3. *Each agent is able to accomplish the training independently.*

This algorithm combines multi-agent actor-critic based off-policy DRL with consensus-based distributed training. Each agent has its own actor training parameter η^j and critic training parameter ξ^j. The graph connection between different agents performs a decentralized consensus-based distributed training.

We consider a network with J agents. In this training scenario, the state of the system at time t is $s_t = [s_t^1, s_t^2, \cdots, s_t^J] \in \mathcal{S}$, where $s_t^j \in \mathcal{S}$ denotes the j^{th} agent at current time step t. The executed joint action at time t can be rewritten as $a_t = [a_t^1, a_t^2, \cdots, a_t^J] \in \mathcal{A}$, where $a_t^j \in \mathcal{A}_j$ denotes the action of the j^{th} agent at current time step t. The state space $\mathcal{S}_j$ and the action space $\mathcal{A}_j$ are continuous. The local reward received by agent j at current time step t is defined by $r(s_t^j, a_t^j)$ and the state transition function of agent j is denoted by $s_{t+1}^j = T(s_t^j, a_t^j, \sigma_t^j)$, where σ_t^j represents the noise in the state dynamics at time step t for agent j [2; 419].

Assumption 4. *Any applied exploration policy retains some ultimate bounded stability for each agent, i.e. $\mathcal{S}$ and $\mathcal{A}$ are compact sets.*

Similar to (5.6) and (5.7), the state value function $V^{\pi}(s)$ and the state action value function $Q^{\pi}(s,a)$ of J agents can be defined as follows [2]:

$$V^{\pi}(s) = \mathbb{E}[\sum_{t=0}^{\infty} \rho^t \sum_{j=1}^{J} r(s_t^j, a_t^j)|s_0 = s, \pi] \tag{5.16}$$

$$Q^{\pi}(s,a) = \mathbb{E}[\sum_{t=0}^{\infty} \rho^t \sum_{j=1}^{J} r(s_t^j, a_t^j)|s_0 = s, a_0 = a, \pi] \tag{5.17}$$

where $\pi(s) : \mathcal{S} \rightarrow \mathcal{A}$ denotes the global deterministic policy function and $\rho \in (0,1)$ is the discounted factor.

The goal of this algorithm is to find the optimal policy $\hat{\pi}(s) = \arg\max_{\pi} f(\pi, s), \forall s \in \mathcal{S}$ of each agent to maximize infinite-time discounted-reward value function $f(\pi, s) = V^{\pi}(s)$.

The definition of the policy function $\pi_{\eta}(s)$ and the value function $g_{\xi}(s,a)$ of J agents can be also represented by (5.9) and (5.10). In this J agent training scenario, $\eta = [(\eta^1)^T, (\eta^2)^T, \cdots, (\eta^J)^T]^T \in \mathbb{R}^{n_{\eta}}$ which denotes the actor training parameter, $\xi = [(\xi^1)^T, (\xi^2)^T, \cdots, (\xi^J)^T]^T \in \mathbb{R}^{n_{\xi}}$ which represents the critic training parameter.

Similar to the single-agent training scenario, the optimization problem of J agents can be also converted to the parameterized policy optimization problem as stated in (5.11). Correspondingly, $g_{\xi}(s,a)$ can be also used to represent the closest approximation of $Q^{\pi}(s,a)$ according to [421; 348]. Hence, $g_{\xi}(s,a) \approx Q^{\pi}(s,a)$. The expression of $\nabla_{\eta} f(\pi_{\eta}(s), s)$ can be rewritten as $\nabla_{\eta} f(\pi_{\eta}(s), s) = \mathbb{E}[\nabla_{\eta}\pi(s) \nabla_a g_{\xi}(s,a)|_{a=\pi_{\eta}(s)}]$.

Under Assumption 2 and Assumption 3, for J agents, the process of updating the critic training parameter and the actor training parameter with consensus-based distributed training can be described as:

$$\kappa_t^j = r_{t+1}^j + \rho g_{\xi}(s_{t+1}^j, a_{t+1}^j) - g_{\xi}(s_t^j, a_t^j) \tag{5.18}$$

$$\xi_{t+1}^j = \tilde{\xi}_t^j + \alpha \kappa_t^j \nabla_{\xi} g(s_t^j, a_t^j) \tag{5.19}$$

$$\eta_{t+1}^j = \tilde{\eta}_t^j + \beta \nabla_{\eta}\pi(s_t^j) \nabla_a g_{\xi}(s_t^j, a^j)|_{a^j = \pi_{\eta}(s^j)} \tag{5.20}$$

$$\tilde{\xi}_{t+1}^j = \mathcal{C}_j\left(\xi_{t+1}^k, w_{kj}\right) \tag{5.21}$$

$$\tilde{\eta}_{t+1}^j = \mathcal{C}_j\left(\eta_{t+1}^k, w_{kj}\right) \tag{5.22}$$

where $\nabla_{\xi} g(s_t^j, a_t^j) = [\psi(s,a)|\xi = \xi_t^j, s = s_t^j, a = a_t^j]$ is the gradient of the j^{th} agent, ξ_{t+1}^j

represents the updated critic training parameter of the j^{th} agent at the next time step, $\tilde{\xi}_{t+1}$ stands for the updated critic training parameter of J agents after consensus-based distributed training at the next time step, $\tilde{\xi}_t$ is the updated critic training parameter of J agents after consensus-based distributed training, κ_t^j stands for temporal difference error of the j^{th} agent, $\tilde{\eta}_t^j$ stands for the updated actor training parameter of the j^{th} agent after consensus-based distributed training, $\tilde{\eta}_{t+1}^j$ stands for the updated actor training parameter of the j^{th} agent after consensus-based distributed training at the next time step, η_{t+1}^j represents the actor training parameter of the j^{th} agent at the next time step.

Algorithm 5.1 summarizes the proposed actor-critic based off-policy DRL with consensus-based distributed training. Compare with the on-policy method, the off-policy method is more powerful due to the fact that it ensures the comprehensiveness of the data and can cover all behaviours.

5.2 Stability Analysis

In this section, we state at first some auxiliary results: The first result considers the stability analysis of a linear, exponentially stable system impacted by some Lipschitz-continuous non-linearity. The second result investigates the Lipschitz continuity of the non-linearities of the overall consensus-based learning algorithms. We use these results to analyse the convergence properties of the actor and the critic training parameters for the system described in (5.18)-(5.22).

5.2.1 Convergence Analysis of a Type of Nonlinear Discrete Systems

Let us consider a system defined as

$$e_{t+1} = \mathbf{A}e_t + \hat{f}(e_t, t), \tag{5.23}$$

where $e \in \mathbb{R}^n$, $\mathbf{A} \in \mathbb{R}^{n\times n}$, and $\hat{f} : \mathbb{R}^n \times \mathbb{R}^+ \to \mathbb{R}^n$ is a Lipschitz continuous function. Moreover $\hat{f}(0, t) = 0$ for all $t \in \mathbb{R}^+$, hence $e_t = 0$ is an equilibrium point of the system.

The matrix $\mathbf{A}$ is said to be Schur if the absolute value of all its eigenvalues are within the unit disk. The notation $\|\cdot\|$ represents the Euclidean norm of a vector or spectral norm of a matrix [383; 423].

Theorem 3. *Let* $\mathbf{A}$ *be Schur. Then there exists* $\hat{\Lambda} > 0$ *such that the system defined in* (5.23) *is exponentially stable if* $\|\hat{f}(e_t, t)\| \leq \hat{\Lambda}\|e_t\|$ *for all* $t \in \mathbb{R}^+$.

Proof. If $\mathbf{A}$ is Schur, there exists a matrix $Q > 0$ such that $\mathbf{A}^T Q\mathbf{A} - Q < -\Gamma I_n$ for some $\Gamma > 0$. Take $\hat{\Lambda}$ such that $\Gamma = \hat{\Lambda}\|\mathbf{A}^T Q\| + \hat{\Lambda}\|Q\mathbf{A}\| + \hat{\Lambda}^2\|Q\|$.
We consider a Lyapunov function

$$V_t = \frac{1}{2} e_t^T Q e_t, \tag{5.24}$$

then it follows

$$\begin{aligned} V_{t+1} - V_t &= \frac{1}{2} e_{t+1}^T Q e_{t+1} - \frac{1}{2} e_t^T Q e_t \\ &= \frac{1}{2}(\mathbf{A}e_t + \hat{f}(e_t, t))^T Q(\mathbf{A}e_t + \hat{f}(e_t, t)) - \frac{1}{2} e_t^T Q e_t \\ &= \frac{1}{2} e_t^T \mathbf{A}^T Q\mathbf{A} e_t - \frac{1}{2} e_t^T Q e_t + \frac{1}{2} e_t^T \mathbf{A}^T Q \hat{f}(e_t, t) + \\ &\frac{1}{2}\hat{f}(e_t, t)^T Q\mathbf{A} e_t + \frac{1}{2}\hat{f}(e_t, t)^T Q \hat{f}(e_t, t). \end{aligned} \tag{5.25}$$

Since $\|\hat{f}(e_t, t)\| \leq \hat{\Lambda}\|e_t\|$ for all $t \in \mathbb{R}^+$, yields

$$\begin{aligned} V_{t+1} - V_t \leq \frac{1}{2} e_t^T \mathbf{A}^T Q\mathbf{A} e_t + \frac{1}{2}\hat{\Lambda}\|\mathbf{A}^T Q\| e_t^T e_t + \frac{1}{2}\hat{\Lambda} \\ \|Q\mathbf{A}\| e_t^T e_t + \frac{1}{2}\hat{\Lambda}^2\|Q\| e_t^T e_t - \frac{1}{2} e_t^T Q e_t \\ = \frac{1}{2} e_t^T (\mathbf{A}^T Q\mathbf{A} + \Gamma I_n - Q) e_t. \end{aligned} \tag{5.26}$$

Let $\hat{R} = -(\mathbf{A}^T Q\mathbf{A} + \Gamma I_n - Q) > 0$, we have

$$\begin{aligned} V_{t+1} - V_t &= -\frac{1}{2} e_t^T \hat{R} e_t \\ &\leq -\frac{\lambda_{min}(\hat{R})}{2} e_t^T e_t \\ &\leq -\frac{\lambda_{min}(\hat{R})}{2\lambda_{max}(Q)} e_t^T Q e_t \\ &= -\hat{a} V_t. \end{aligned} \tag{5.27}$$

where $\hat{a} = \frac{\lambda_{min}(\hat{R})}{\lambda_{max}(Q)} > 0$. By using the Lyapunov Theorem [424], for any initial condition $e_0 \in \mathbb{R}^n$, i.e. the system is exponentially stable.

□

Lemma 2. *Let $x^j \in \mathbb{R}^k$. If $\bar{f}(x^j)$ is Lipschitz continuous with Lipschitz constant Λ, and we apply the change of variable $e^j = x^j - \bar{x}$, where $\bar{x} = \frac{1}{J}\sum_{n=1}^{J} x^n$. Define $e = [(e^1)^T, (e^2)^T, \cdots, (e^J)^T]^T \in \mathbb{R}^{Jk}$ and $\hat{f}(e^j, \bar{x}) = \bar{f}(x^j) - \frac{1}{J}\sum_{n=1}^{J} \bar{f}(x^n)$, then the function $\hat{F}(e, \bar{x}) = [\hat{f}(e^1, \bar{x})^T, \hat{f}(e^2, \bar{x})^T, \cdots, \hat{f}(e^J, \bar{x})^T]^T$ satisfies*

$$\left\|\hat{F}(e, \bar{x})\right\| \leq \hat{\Lambda}\|e\| \qquad \forall e \in \mathbb{R}^{Jk}, \tag{5.28}$$

where $\hat{\Lambda} = 2\Lambda\sqrt{J}$.

Proof. Since $\bar{f}(x_t)$ is Lipschitz continuous with a Lipschitz constant Λ, it follows

$$\begin{aligned}
\left\|\hat{f}(e^j, \bar{x})\right\| &= \left\|\frac{1}{J}\sum_{n=1}^{J} \bar{f}(x^j) - \frac{1}{J}\sum_{n=1}^{J} \bar{f}(x^n)\right\| \\
&= \left\|\frac{1}{J}\sum_{n=1}^{J} (\bar{f}(x^j) - \bar{f}(x^n))\right\| \\
&\leq \frac{\Lambda}{J}\sum_{n=1}^{J} \|x^j - x^n\| \\
&= \frac{\Lambda}{J}\sum_{n=1}^{J} \|(x^j - \bar{x}) - (x^n - \bar{x})\| \\
&= \frac{\Lambda}{J}\sum_{n=1}^{J} \|e^j - e^n\| \\
&\leq \frac{\Lambda}{J}\sum_{n=1}^{J} (\|e^j\| + \|e^n\|) \\
&= \Lambda\|e^j\| + \frac{\Lambda}{J}\sum_{n=1}^{J} \|e^n\| .
\end{aligned} \tag{5.29}$$

Then,

$$\begin{aligned}
\left\|\hat{F}(e, \bar{x})\right\|^2 &= \sum_{n=1}^{J} \left\|\hat{f}(e^n, \bar{x})\right\|^2 \\
&\leq \left(\sum_{n=1}^{J} \left\|\hat{f}(e^n, \bar{x})\right\|\right)^2 \\
&\leq (2\Lambda \sum_{n=1}^{J} \|e^n\|)^2 \\
&\leq 4\Lambda^2 J \sum_{n=1}^{J} \|e^n\|^2 \\
&= 4\Lambda^2 J \|e\|^2 ,
\end{aligned} \tag{5.30}$$

where we have used the generalized mean. So we complete the proof if $\hat{\Lambda} = 2\Lambda\sqrt{J}$.

□

5.2.2 Convergence Analysis of the Critic Training Parameter

In this section, we analyse the critic training parameter dynamics as defined in (5.19) and (5.21).

Substituting $\nabla_\xi g(s_t^j, a_t^j)$ in (5.19) with (5.10), it follows

$$\begin{aligned}\xi_{t+1}^j &= \tilde{\xi}_t^j + \alpha\kappa_t^j\psi(s_t^j, a_t^j) \\ &= \tilde{\xi}_t^j + \alpha(r_{t+1}^j + \rho g_\xi(s_{t+1}^j, a_{t+1}^j) \\ &- g_\xi(s_t^j, a_t^j))\psi(s_t^j, a_t^j) \\ &= \tilde{\xi}_t^j + \alpha(r_{t+1}^j + \rho(\xi_t^j)^T\psi(s_{t+1}^j, a_{t+1}^j) \\ &- (\xi_t^j)^T\psi(s_t^j, a_t^j))\psi(s_t^j, a_t^j).\end{aligned} \tag{5.31}$$

We show that the nonlinear term in the equation above is Lipschitz continuous:

Lemma 3. *Let* $\Xi(\xi_t, r_{t+1}, s_t, s_{t+1}, a_t, a_{t+1}) = \alpha(r_{t+1}+\rho(\xi_t)^T\psi(s_{t+1}, a_{t+1})-\xi_t^T\psi(s_t, a_t))\psi(s_t, a_t)$, *where* $s_t, s_{t+1} \in \mathcal{S}$ *and* $a_t, a_{t+1} \in \mathcal{A}$, *for all* t. *Then,* Ξ *is Lipschitz continuous with respect to the variable* ξ_t.

Proof. With some abuse of notation, henceforth we use $\Xi(\xi_t) = \Xi(\xi_t, r_{t+1}, s_t, s_{t+1}, a_t, a_{t+1})$. For any ξ_t and $\xi_{t'}$, we have

$$\begin{aligned}\left\|\Xi(\xi_{t'}) - \Xi(\xi_t)\right\| &= \alpha\|(r_{t+1} + (\xi_{t'})^T(\rho\psi(s_{t+1}, a_{t+1})- \\ &\psi(s_t, a_t)))\psi(s_t, a_t) - (r_{t+1} + (\xi_t)^T(\rho \\ &\psi(s_{t+1}, a_{t+1}) - \psi(s_t, a_t)))\psi(s_t, a_t)\| \\ &= \alpha\|(\xi_{t'} - \xi_t)^T(\rho\psi(s_{t+1}, a_{t+1})- \\ &\psi(s_t, a_t))\psi(s_t, a_t)\| \\ &\leq \alpha\|\rho\psi(s_{t+1}, a_{t+1}) - \psi(s_t, a_t)\| \\ &\|\psi(s_t, a_t)\|\|\xi_{t'} - \xi_t\|.\end{aligned} \tag{5.32}$$

By Assumption 4, s_t and s_{t+1} belong to a compact set $\mathcal{S}$, a_t and a_{t+1} belong to a compact set $\mathcal{A}$, $\forall t$, there exists $\Lambda > 0$ that satisfies $\alpha\|\rho\psi(s_{t+1}, a_{t+1})-\psi(s_t, a_t)\|\|\psi(s_t, a_t)\| < \Lambda$, $\forall t$, $s_t \in \mathcal{S}, a_t \in \mathcal{A}$ as ψ is a bounded function. As a result, Ξ is Lipschitz continuous with respect to ξ_t, i.e.

$$\left\|\Xi(\xi_{t'}) - \Xi(\xi_t)\right\| < \Lambda\|\xi_{t'} - \xi_t\|. \tag{5.33}$$

□

α will allow to tune the Lipschitz constant of the non-linearity which will be important for analysis later.

Now, the critic training process of each agent is given by

$$\xi_{t+1}^{j} = \tilde{\xi}_{t}^{j} + \Xi(\xi_{t}^{j}) \tag{5.34}$$

$$\tilde{\xi}_{t}^{j} = M\xi_{t}^{j} + \iota T \sum_{l=1}^{J} \hat{a}_{jl}(\xi_{t}^{l} - \xi_{t}^{j}) \tag{5.35}$$

where $M = \hat{m}I_k$ and $\hat{m} \leq 1$, ι is the scalar coupling gain, T stands for the feedback gain, $\hat{a}_{jl}$ represents the elements of the graph adjacency matrix generated by the undirected graph $\mathcal{G}$.

For J agents, the training process can be rewritten as

$$\begin{aligned} \xi_{t+1} &= (I_J \otimes M)\xi_t - (\iota L \otimes T)\xi_t + \Omega(\xi_t) \\ &= (I_J \otimes M - \iota L \otimes T)\xi_t + \Omega(\xi_t) \\ &= \mathbf{A}'\xi_t + \Omega(\xi_t), \end{aligned} \tag{5.36}$$

where $\mathbf{A}' = I_J \otimes M - \iota L \otimes T$ and $\Omega(\xi_t) = [\Xi(\xi_t^1)^T, \Xi(\xi_t^2)^T, ..., \Xi(\xi_t^J)^T]^T$.

Proposition 1. *Given an undirected graph $\mathcal{G}$, under Assumptions 2, 3 and 4, if $\mathbf{A}$ is Schur, the critic training parameter ξ of each agent in* (5.19) *and* (5.21) *will exponentially converge to a neighbourhood of the optimal value ξ^* for some small enough α. The size of the neighbourhood approaches 0 as M approaches I_k.*

Proof. Let $\bar{\xi} = \frac{1}{J}\sum_{n=1}^{J} \xi^n$, $e^j = \xi^j - \bar{\xi}$, for J agents, the error e_t is given by

$$e_t = ((I_J - \frac{1}{J}\mathbf{1}\mathbf{1}^T) \otimes I_k)\xi_t. \tag{5.37}$$

Since $(I_J - \frac{1}{J}\mathbf{1}\mathbf{1}^T) \otimes I_k$ and $I_J \otimes M - \iota L \otimes T$ are commute, combining (5.36) and (5.37), yields

$$\begin{aligned} e_{t+1} = ((I_J - \frac{1}{J}\mathbf{1}\mathbf{1}^T) \otimes I_k)((I_J \otimes M - \iota L \otimes T)e_t + \Omega(\xi_t)) \\ = \mathbf{A}e_t + \mathbf{B}\Omega(\xi_t), \end{aligned} \tag{5.38}$$

where $\mathbf{A} = \mathbf{B}\mathbf{A}'$, $\mathbf{B} = (I_J - \frac{1}{J}\mathbf{1}\mathbf{1}^T) \otimes I_k$.

Algorithm 5.1 Actor-critic based off-policy DRL with consensus-based distributed training

1: Initialize each agent actor training parameter η_0^j, critic training parameter ξ_0^j, critic learning rate α, actor learning rate β, initial state s_0^j, fixed behaviour policy μ, discounted factor ρ, maximum time limit $\hat{T}$, maximum episode limit $\hat{N}$, exploration noise O_t.

2: **for** episode = 1, $\hat{N}$ **do**

3: Reset the environment.

4: Receive initial state s_1^j for each agent.

5: **for** t = 1, $\hat{T}$ **do**

6: Choose action $a_t^j = \mu(s_t^j) + O_t$ for action exploration according to the fixed behaviour policy and the exploration noise.

7: Execute a_t^j, calculate reward r_{t+1}^j and get new state s_{t+1}^j.

8: Store transition $(s_t^j, a_t^j, r_t^j, s_{t+1}^j)$ in the replay buffer R_p.

9: Sample a minibatch $(s_t^j, a_t^j, r_t^j, s_{t+1}^j)$ from transitions in R_p.

10: Compute temporal difference error:
$\kappa_t^j = r_{t+1}^j + \rho g_\xi(s_{t+1}^j, a_{t+1}^j) - g_\xi(s_t^j, a_t^j)$.

11: Update the local critic value training parameter:
$\xi_{t+1}^j = \tilde{\xi}_t^j + \alpha\kappa_t^j \nabla_\xi g(s_t^j, a_t^j)$.

12: Update the local actor policy training parameter:
$\eta_{t+1}^j = \tilde{\eta}_t^j + \beta \nabla_\eta \pi(s_t^j) \nabla_a g_\xi(s_t^j, a^j)|_{a^j=\pi_\eta(s^j)}$.

13: Perform consensus-based distributed training on the critic value training parameter and the actor policy training parameter of each agent:
$\tilde{\xi}_{t+1}^j = \mathcal{C}_j\left(\xi_{t+1}^k, w_{kj}\right)$.
$\tilde{\eta}_{t+1}^j = \mathcal{C}_j\left(\eta_{t+1}^k, w_{kj}\right)$.

14: **end for**

15: **end for**

By Lemma 3, Ξ is Lipschitz continuous with constant Λ. Let $\hat{\Xi}(\xi^i) = \Xi(\xi^i) - \frac{1}{J}\sum_{n=1}^{J}\Xi(\xi^n)$. Then, by Lemma 2, the nonlinear function

$$\mathbf{B}\Omega(\xi) = \left[\hat{\Xi}(\xi^1)^T \quad \hat{\Xi}(\xi^2)^T \quad \cdots \quad \hat{\Xi}(\xi^J)^T\right]^T$$

satisfies $\|\mathbf{B}\Omega(\xi)\| \leq 2\Lambda\sqrt{J}\|e\|$ for all $\bar{\xi}$.

As $\mathbf{A}$ is Schur and Λ is proportional to α, by applying Theorem 3, there exists a small enough selection of α such that $\lim_{t\to\infty} e_t = 0$. Let $\mathbf{B}' = \frac{1}{J}\mathbf{1}\mathbf{1}^T \otimes I_k$, from (5.36), we can get

$$\bar{\xi}_{t+1} = (I_J \otimes M)\bar{\xi}_t + \mathbf{B}'\Omega(\xi_t). \tag{5.39}$$

When $\hat{m} = 1$, $M = I_k$, we can obtain that the critic training parameter ξ will converge to ξ^* from Assumption 2.

When $\hat{m} < 1$, under Assumption 2, it can be obtained that $\Omega(\xi^*) = 0$. Therefore,

$$\begin{aligned}\bar{\xi}_{t+1} - \xi^* &= (I_J \otimes M)\bar{\xi}_t - \xi^* + \mathbf{B}'\Omega(\xi_t) - \mathbf{B}'\Omega(\xi^*) \\ &= (I_J \otimes M)(\bar{\xi}_t - \xi^*) + \mathbf{B}'\Omega(\bar{\xi}_t) - \mathbf{B}'\Omega(\xi^*) \\ &+ \mathbf{B}'\Omega(\xi_t) - \mathbf{B}'\Omega(\bar{\xi}_t) + (I_J \otimes M)\xi^* - \xi^*.\end{aligned} \tag{5.40}$$

Consider a Lyapunov function $V_n = Q^n\|\bar{\xi}_t - \xi^*\|^2$, where $Q^n > 0$, and $Q^n\|I_J \otimes M\|^2 - Q^n < 0$. From the average inequality, it follows that

$$\begin{aligned}V_n(t+1) - V_n(t) &= Q^n\|(I_J \otimes M)(\bar{\xi}_t - \xi^*) + \mathbf{B}'\Omega(\bar{\xi}_t) - \mathbf{B}'\Omega(\xi^*) \\ &+ \mathbf{B}'\Omega(\xi_t) - \mathbf{B}'\Omega(\bar{\xi}_t) + (I_J \otimes M)\xi^* - \xi^*\|^2 - Q^n\|\bar{\xi}_t - \xi^*\|^2 \\ &\le ((1+3\hat{k})Q^n\|I_J \otimes M\|^2 - Q^n + (\frac{1}{\hat{k}} + 3)Q^n\Lambda^2 J)\|\bar{\xi}_t - \xi^*\|^2 \\ &+ (\frac{1}{\hat{k}} + 3)Q^n\Lambda^2 J\|\xi_t - \bar{\xi}_t\|^2 + (\frac{1}{\hat{k}} + 3)Q^n\|(I_J \otimes M)\xi^* - \xi^*\|^2.\end{aligned} \tag{5.41}$$

There exists $\hat{k} > 0$ so that $k_n > 0$, $k_n^* > 0$, and $0 < \bar{k}_n < \frac{\hat{a}\lambda_{min}(Q)}{2}$, where

$$-k_n = (1+3\hat{k})Q^n\|I_J \otimes M\|^2 - Q^n + (\frac{1}{\hat{k}} + 3)Q^n\Lambda^2 J \tag{5.42}$$

and

$$\bar{k}_n = (\frac{1}{\hat{k}} + 3)Q^n\Lambda^2 J, \;\; k_n^* = (\frac{1}{\hat{k}} + 3)Q^n. \tag{5.43}$$

Hence,

$$\begin{aligned}V_n(t+1) - V_n(t) &\le -k_n\|\bar{\xi}_t - \xi^*\|^2 + \bar{k}_n\|e_t\|^2 + \\ &\qquad k_n^*\|(I_J \otimes M)\xi^* - \xi^*\|^2 \\ &\le -\frac{k_n}{Q_n}V_n + \frac{2\bar{k}_n}{\lambda_{min}(Q)}V_t + k_n^*\|(I_J \otimes M)\xi^* - \xi^*\|^2.\end{aligned} \tag{5.44}$$

Then, we can construct a new Lyapunov function

$$V_a(t) = V_n(t) + V_t \tag{5.45}$$

From (5.27) and (5.44), we have

$$\begin{aligned} V_a(t+1) - V_a(t) &= V_n(t+1) - V_n(t) + V_{t+1} - V_t \\ &\leq -\frac{k_n}{Q_n} V_n(t) - \bar{a} V_t + k_n^* \|(I_J \otimes M)\xi^* - \xi^*\|^2 \\ &\leq -g_a V_a(t) + k_n^* \|(I_J \otimes M)\xi^* - \xi^*\|^2. \end{aligned} \tag{5.46}$$

where $g_a = min\{\frac{k_n}{Q_n}, \bar{a}, g^*\}$, $\bar{a} = \hat{a} - \frac{2\bar{k}_n}{\lambda_{min}(Q)} > 0$, and $g^* \in (0,1)$ is a constant. Hence, $0 < g_a < 1$, we can get

$$V_a(t+1) \leq (1 - g_a) V_a(t) + k_n^* \|(I_J \otimes M)\xi^* - \xi^*\|^2 \tag{5.47}$$

When $t \to +\infty$,

$$V_a(t) \leq \frac{k_n^*}{g_a} \|(1 - \hat{m})\xi^*\|^2 \tag{5.48}$$

As a result, the critic training parameter ξ will exponentially converge to a neighbourhood of the optimal value ξ^* for some small enough α. The size of the neighbourhood approaches 0 as M approaches I_k. This completes the proof. □

5.2.3 Convergence Analysis of the Actor Training Parameter

In this section, we analyse the actor training parameter dynamics as defined in (5.20) and (5.22).

Under Assumption 1 and (5.9), (5.20) can be rewritten as

$$\begin{aligned} \eta_{t+1}^j &= \tilde{\eta}_t^j + \beta \nabla_\eta \pi(s_t^j)(\nabla_\eta \pi(s_t^j))^T \xi_t^j \\ &= \tilde{\eta}_t^j + \beta \phi(s_t^j)\phi(s_t^j)^T \xi_t^j. \end{aligned} \tag{5.49}$$

Following [425; 426], $\xi^j = \chi(\eta^j)$ where χ is a Lipschitz continuous function. Therefore, (5.49) can be rewritten as

$$\eta_{t+1}^j = \tilde{\eta}_t^j + \beta \phi(s_t^j)\phi(s_t^j)^T \chi(\eta_t^j). \tag{5.50}$$

We show that the nonlinear term in the equation above is Lipschitz continuous:

Lemma 4. *Let $H(\eta_t^j, s_t) = \beta\phi(s_t^j)\phi(s_t^j)^T\chi(\eta_t^j)$, if $s_t \in \mathcal{S}$, $\forall t$. Then H is Lipschitz continuous in η_t^j.*

Proof. With some abuse of notation, henceforth we use $H(\eta_t^j, s_t) = H(\eta_t^j)$. Since $\chi(\eta)$ is Lipschitz continuous, for any η_t and $\eta_{t'}$, we have

$$\|\chi(\eta_{t'}) - \chi(\eta_t)\| < \lambda^*\|\eta_{t'} - \eta_t\|, \tag{5.51}$$

where $\lambda^* > 0$ is a Lipschitz constant. From (5.51), it follows

$$\begin{aligned}\|H(\eta_{t'}) - H(\eta_t)\| &= \|\beta\phi(s_t^j)\phi(s_t^j)^T\chi(\eta_{t'}^j) - \beta\phi(s_t^j) \\ &\quad \phi(s_t^j)^T\chi(\eta_t^j)\| \\ &\leq \beta\|\phi(s_t^j)\phi(s_t^j)^T\|\|\chi(\eta_t^j) - \chi(\eta_t^j)\| \\ &< \beta\|\phi(s_t^j)\phi(s_t^j)^T\|\lambda^*\|\eta_{t'} - \eta_t\|.\end{aligned} \tag{5.52}$$

By Assumption 4, s_t belongs to a compact set $\mathcal{S}$, $\forall t$, there exists a λ that satisfies $\beta\|\phi(s_t^j)\phi(s_t^j)^T\|\lambda^* < \lambda$, $\forall t$, $s_t \in \mathcal{S}$ as ϕ is a bounded function. Therefore, H is Lipschitz continuous in η_t, i.e.

$$\|H(\eta_{t'}) - H(\eta_t)\| < \lambda\|\eta_{t'} - \eta_t\| \tag{5.53}$$

for all $\eta_{t'}$ and η_t.

□

β will allow tuning the Lipschitz constant of the non-linearity which will be important for analysis later.

Now, the actor training process of each agent is obtained by

$$\eta_{t+1}^j = \tilde{\eta}_t^j + H(\eta_t^j) \tag{5.54}$$

$$\tilde{\eta}_t^j = \hat{M}\eta_t^j + \epsilon K\sum\nolimits_{l=1}^{J}\hat{a}_{jl}(\eta_t^l - \eta_t^j) \tag{5.55}$$

where $\hat{M} = \hat{m}I_k$ and $\hat{m} \leq 1$, ϵ is the scalar coupling gain, K stands for the feedback gain, $\hat{a}_{jl}$ represents the elements of the graph adjacency matrix generated by the undirected graph $\mathcal{G}$.

For J agents, the training process can be rewritten as

$$\begin{aligned}\eta_{t+1} &= (I_J \otimes \hat{M})\eta_t - (\epsilon L \otimes K)\eta_t + G(\eta_t) \\ &= (I_J \otimes \hat{M} - \epsilon L \otimes K)\eta_t + G(\eta_t) \\ &= \mathbf{A}'\eta_t + G(\eta_t),\end{aligned} \tag{5.56}$$

where $\mathbf{A}' = I_J \otimes \hat{M} - \epsilon L \otimes K$ and $G(\eta_t) = [H(\eta_t^1)^T, H(\eta_t^2)^T, ..., H(\eta_t^J)^T]^T$.

Proposition 2. *Given an undirected graph $\mathcal{G}$, under Assumptions 1, 2, 3 and 4, if $\mathbf{A}$ is Schur, the actor training parameter η of each agent in* (5.20) *and* (5.22) *will exponentially converge to a neighbourhood of the optimal value η^* for small enough β. The size of the neighbourhood approaches 0 as $\hat{M}$ approaches I_k.*

Proof. Let $\bar{\eta} = \frac{1}{J}\sum_{n=1}^{J}\eta^n$, $e^j = \eta^j - \bar{\eta}$, for J agents, the error e_t is given by

$$e_t = ((I_J - \frac{1}{J}\mathbf{1}\mathbf{1}^T) \otimes I_k)\eta_t. \tag{5.57}$$

Since $(I_J - \frac{1}{J}\mathbf{1}\mathbf{1}^T) \otimes I_k$ and $I_J \otimes \hat{M} - \epsilon L \otimes K$ are commute, combining (5.56) and (5.57), yields

$$\begin{aligned}e_{t+1} = ((I_J - \frac{1}{J}\mathbf{1}\mathbf{1}^T) \otimes I_k)((I_J \otimes \hat{M} - \epsilon L \otimes K)e_t + G(\eta_t)) \\ = \mathbf{A}e_t + \mathbf{B}G(\eta_t),\end{aligned} \tag{5.58}$$

where $\mathbf{A} = \mathbf{B}\mathbf{A}'$, $\mathbf{B} = (I_J - \frac{1}{J}\mathbf{1}\mathbf{1}^T) \otimes I_k$.
By Lemma 4, H is Lipschitz continuous with constant λ. Let $\hat{H}(\eta^i) = H(\eta^i) - \frac{1}{J}\sum_{n=1}^{J}H(\eta^n)$. Then, by Lemma 2, the nonlinear function

$$\mathbf{B}G(\eta) = \begin{bmatrix}\hat{H}(\eta^1)^T & \hat{H}(\eta^2)^T & \cdots & \hat{H}(\eta^J)^T\end{bmatrix}^T$$

satisfies $\|\mathbf{B}G(\eta)\| \leq 2\lambda\sqrt{J}\|e\|$ for all $\bar{\eta}$.

As $\mathbf{A}$ is Schur and λ is proportional to β, by applying Theorem 3, there exists a small enough selection of β such that $\lim_{t\to\infty} e_t = 0$.

Similarly, we can choose the Lyapunov function as

$$V_b(t) = Q^n\|\bar{\eta}_t - \eta^*\|^2 + V_t \tag{5.59}$$

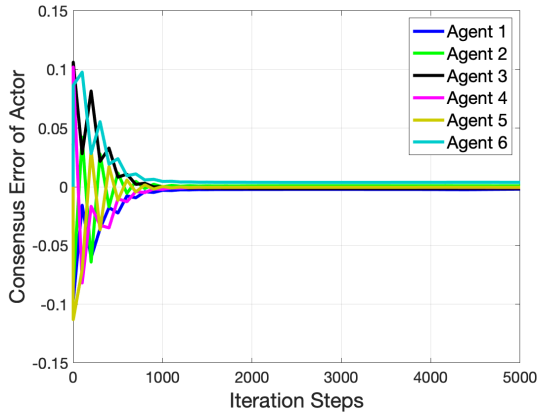
By using the same analysis in Proposition 1, it follows that the actor training parameter η will exponentially converge to a neighbourhood of the optimal value η^* for small enough β. The size of the neighbourhood approaches 0 as $\hat{M}$ approaches I_k. This completes the proof. □

5.3 Experiments and Results

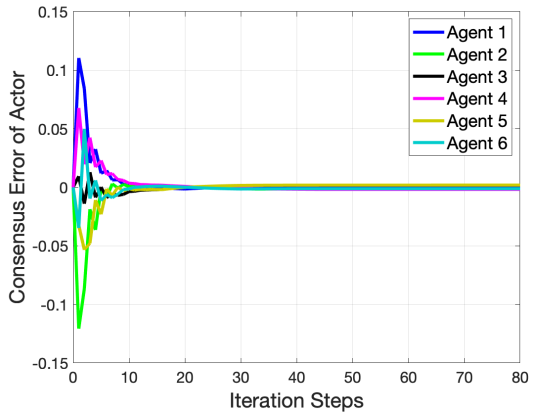
The feasibility of the proposed multi-agent training framework and algorithm is validated in this section. Section 5.3.1 compares the proposed method with the existing consensus-based RL method and Section 5.3.2 introduces the setup of our proposed algorithm. The training details are depicted in Section 5.3.3 and Section 5.3.4 demonstrates the simulation results. A comparison with an existing multi-agent algorithm is conducted in Section 5.3.5. Section 5.3.6 details the discussion on bandwidth and privacy protection.

5.3.1 Comparison with Existing Consensus-based RL Method

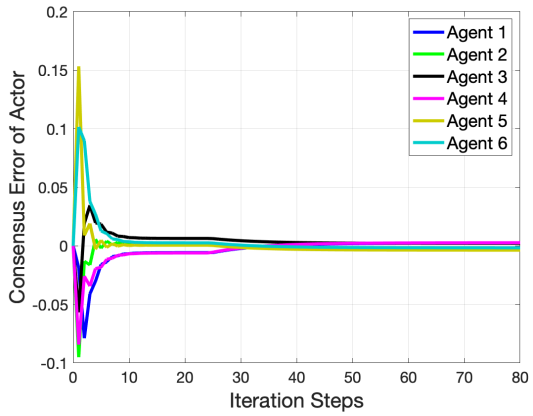
Perhaps the most relevant work to our proposed algorithm is shown in [2]. The key idea in [2] is to perform consensus-based distributed training with RL in a two-time scale technique, which means each local policy training parameter is fixed during the critic update and every local critic training parameter has already reached the correct value during the actor update. Additionally, consensus-based distributed training is only applied to each local policy training parameter. Compared with [2], our algorithm is able to update both actor and critic training parameters simultaneously. As a result, the system is no longer a single excitation system and the consensus error of the actor network with our proposed method (as shown in Fig. 5.2 (b)) converges 50 times faster than the algorithm proposed in [2] (as shown in Fig. 5.2 (a)). Furthermore, even if the consensus-based distributed training on the critic value training parameter is removed from our proposed algorithm, the consensus error of the actor network (as shown in Fig. 5.2 (c)) is still 25 times faster than the algorithm proposed in [2].



(a)



(b)



(c)

Figure 5.2: Consensus error of the actor network with 6 agents. The setup for both methods are described in [2]. (a) With the algorithm proposed in [2] (b) With our algorithm (c) With our algorithm except consensus-based distributed training on the critic value training parameter

5.3.2 Deep Reinforcement Learning Setup

As mentioned in the Section above, the aim of the experiment is to train each UR5 robot arm to reach the random target position in a smooth trajectory without jerk.

Reward Space

Each UR5 robot arm is designed to reach its target position. The reward function for each robot arm is defined as

$$r_j = r_{dj} + r_{aj} + r_{kj} \tag{5.60}$$

where r_j stands for accumulated reward of the j^{th} UR5 robot arm, r_{dj} represents the distance reward of the j^{th} UR5 robot arm, r_{aj} describes the arrive reward of the j^{th} UR5 robot arm, r_{kj} is the smoothness reward of the j^{th} UR5 robot arm.
The distance reward r_{dj} can be computed by

$$r_{dj} = r_{pj} - r_{cj} \tag{5.61}$$

where r_{pj} denotes the distance between the end effector position and the target point with previous action of the j^{th} UR5 robot arm, r_{cj} stands for the distance between the end effector position and the target point with current action of the j^{th} UR5 robot arm. If r_{pj} is greater than r_{cj} the robot will receive a positive distance reward since it gets closer to the target.

The arrive reward r_{aj} is given by

$$r_{aj} = \begin{cases} r_{ap} & \text{if } |r_{cj}| < r_{dh} \\ 0 & \text{otherwise} \end{cases} \tag{5.62}$$

where r_{ap} equals to a positive peak reward, r_{dh} represents the distance threshold.

The smoothness reward r_{kj} is computed by

$$r_{kj} = \begin{cases} -r_{kp} & \text{if } |r_{dj}| > r_{kh} \\ 0 & \text{otherwise} \end{cases} \tag{5.63}$$

where r_{kp} denotes a positive smoothness peak reward, r_{kh} represents the smoothness threshold, and $|\cdot|$ represents the Euclidean norm of a vector. As a result, r_{kj} will

receive punishment if $|r_{dj}|$ is above the smoothness threshold. By introducing r_{kj} in the accumulated reward, the smooth trajectory of the UR5 robot arm can be guaranteed.

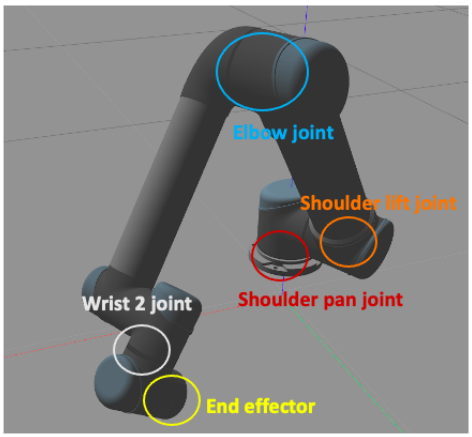


Figure 5.3: The joint positions of a UR5 robot arm.

Action Space and State Space

The base joint of each UR5 robot arm is actuated in this scenario, together with the second joint and the third joint from the base. Consequently, for each UR5 robot arm, the action space is a vector of dimension 3 that consists of three different joint values, which are the shoulder pan joint, the shoulder lift joint and the elbow joint, as shown in Fig. 5.3.

The state space is a vector of dimension 16 which includes the action space, the location of the elbow joint and the wrist 2 joint, the distance between the elbow joint and the goal, the distance between the wrist 2 joint and the goal, the distance between the end effector and the goal.

Neural Network Structure

The input of the actor network passes the data through three dense layers whose activation functions are ReLu [403] with either 500 or 1000 nodes, as shown in Fig. 5.4. It contains 16 elements, which are the joint angle of the shoulder pan joint, the shoulder lift joint, the elbow joint, the location of the elbow joint and the wrist 2 joint, the distance between the elbow joint and the goal, the distance between the wrist 2 joint

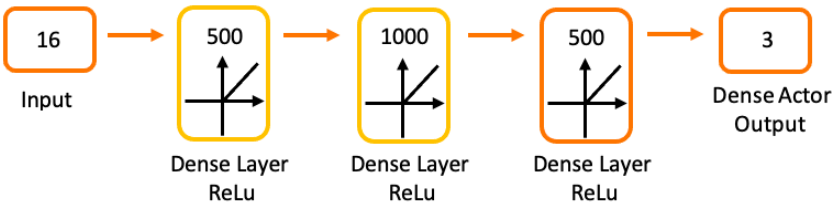


Figure 5.4: The structure of the actor network. The fixed layers are marked with yellow boxes.

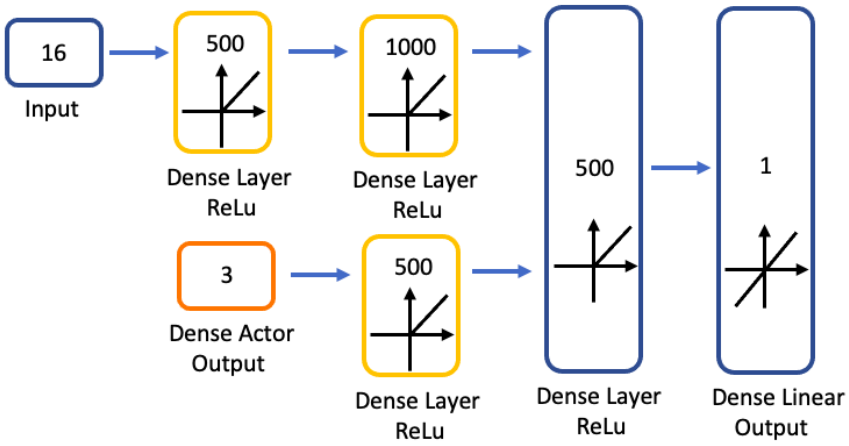


Figure 5.5: The structure of the critic network. The fixed layers are marked with yellow boxes.

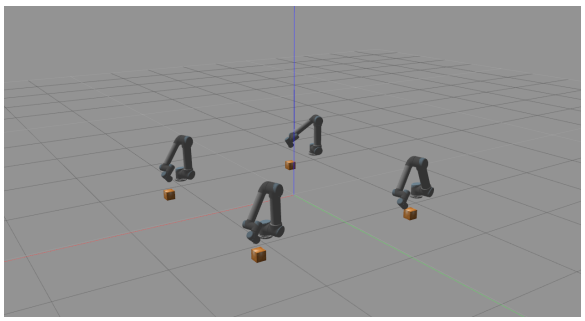
and the goal, the distance between the end effector and the goal. The output of the actor network consists of 3 elements, which are the joint angle of the shoulder pan joint, the shoulder lift joint and the elbow joint. This is also used in the input of the critic network, as illustrated in Fig. 5.5. The other input of the critic network shares the same elements as the input of the actor network. The output of the critic network depicts how good the actor performance is, which is a Q value generated by a linear activation function.

During each iteration, each UR5 robot arm tries to explore the environment with random noise. The accumulated reward can be computed by r_j. For each training, the state space, the action space, the next state space and the accumulated reward are stored in the replay buffer, and the training model can learn from the replay buffer randomly when it is full.

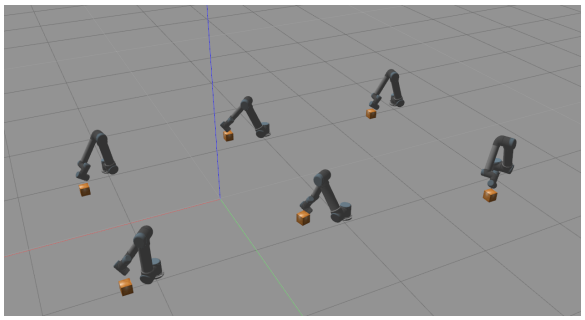
In order to satisfy the condition listed in (5.9) and (5.10), the first two layers of the

actor network and the first three layers of the critic network are fixed with the optimal weight. The fixed layers are marked with yellow boxes.

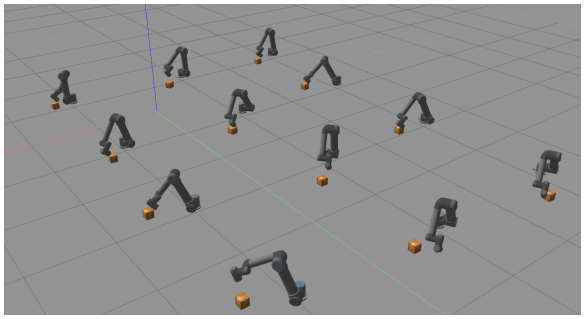
5.3.3 Training Details



(a)



(b)



(c)

Figure 5.6: Actor-critic based off-policy DRL and consensus-based distributed training with different numbers of UR5 robot arms. (a) With 4 UR5 robot arms. (b) With 6 UR5 robot arms. (c) With 12 UR5 robot arms.

Our system was trained in ROS [405] and Gazebo [427] for efficiency. Experiments were carried out with different numbers of UR5 robot arms, as demonstrated in Fig. 5.6. Single UR5 robot arm training with actor-critic based DRL is applied in Fig. 5.8 as a baseline to compare with our consensus-based distributed DRL algorithm. Other three

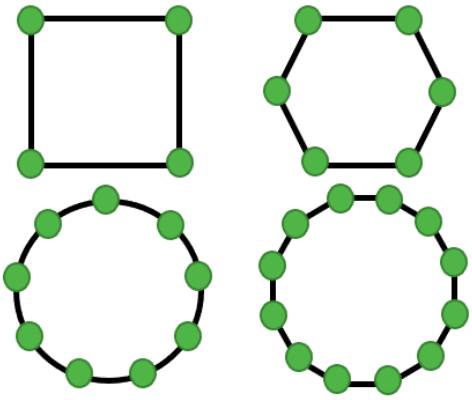


Figure 5.7: Interaction topology of 4, 6, 9 and 12 agents.

experiments are implementing our consensus-based distributed training with four UR5 robot arms, as shown in Fig. 5.8. Fig. 5.8 (a) and Fig. 5.8 (b) compare the performance of four UR5 robot arms with different feedback gains. Fig. 5.8 (a) and Fig. 5.8 (c) contrast the performance of four UR5 robot arms with higher connectivity. The experiments in Fig. 5.9 are implementing our algorithm with six, nine and twelve UR5 robot arms.

The interaction topology of different numbers of UR5 robot arms is shown in Fig. 5.7. The scalar coupling gain τ for the critic training parameter ξ and the scalar coupling gain ϵ for the actor training parameter η are both set to 0.3. The feedback gain T for the critic training parameter ξ is set to 0.9, so does the feedback gain K for the actor training parameter η. It can be verified that $\mathbf{A}$ is Schur for both actor and critic training scenarios.

For each UR5 robot arm, the actor learning rate β in (5.19) and the critic learning rate α in (5.20) are set to 0.001. The discounted factor ρ [303] listed in (5.18) has a fixed value of 0.9. The action exploration mentioned in Algorithm 5.1 satisfies ϵ-greedy exploration strategy [303] with ϵ initialized at 0.9 and a decay rate $\hat{\rho} = 0.99995$. The maximum time limit $\hat{T}$ is set to 200 and the maximum episode limit $\hat{N}$ is initialized at 255.

5.3.4 Simulation Results

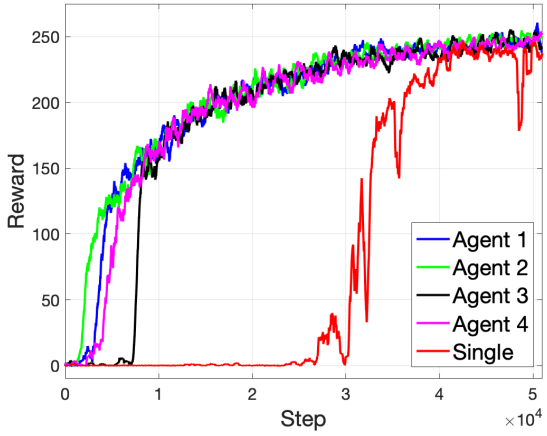
In this section, we evaluate the performance of the proposed method in various ways.

Evaluation of Decentralized Consensus Training

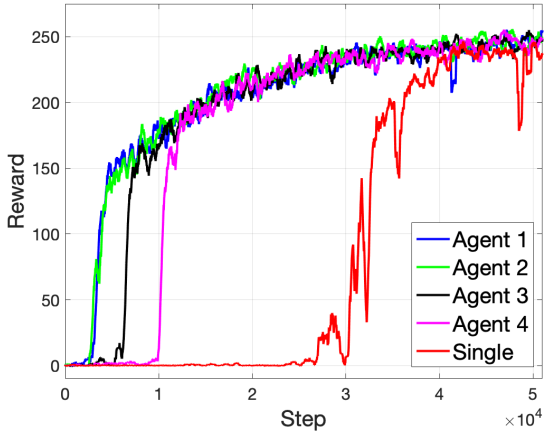
The average reward graph of implementing our algorithm with four UR5 robot arms is elucidated in Fig. 5.8 (a), which confirms the validity of the theoretical analysis. The interaction topology used in Fig. 5.8 (a) is shown in Fig. 5.7. The average reward of single UR5 robot arm training is taken as the base of the comparison. The average reward is computed by averaging the reward within a fixed batch size. Compared with single UR5 robot arm DRL, the training speed of our algorithm is faster. Besides, the average reward curve of each UR5 robot arm is more stable. All UR5 robot arms training with our algorithm have reached consensus at about 10000 steps while the training reward of a single UR5 robot arm did not increase until reaching around 29000 steps. The proposed algorithm guarantees a better performance compared with single UR5 robot arm DRL.

Fig. 5.8 (b) depicts the average reward graph of implementing our algorithm with four UR5 robot arms with the feedback gain set to 0.1. Compared with Fig. 5.8 (a), all UR5 robot arms reach consensus at around 12000 steps. This shows that a smaller feedback gain can slow down the training. If the interaction topology of four UR5 robot arms is set to fully connected, the required convergence steps decrease to around 5000 steps, as shown in Fig. 5.8 (c). In addition, the reward curve of each UR5 robot arm looks tighter because the communication between each UR5 robot arm is strengthened. Therefore, larger feedback gain and fully connected interaction topology result in faster convergence to the optimal result.

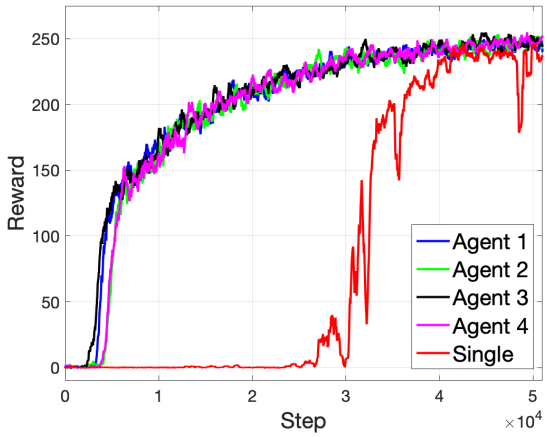
Fig. 5.9 describes the average reward performance of three groups of different numbers of UR5 robot arms. The interaction topology used in Fig. 5.9 is shown in Fig. 5.7. The average reward is computed by averaging the reward within a fixed batch size. As stated in Fig. 5.9 (a), all six UR5 robot arms converge to the optimal value at around 8000 steps, which is smaller than the consensus steps with four UR5 robot arms shown



(a)

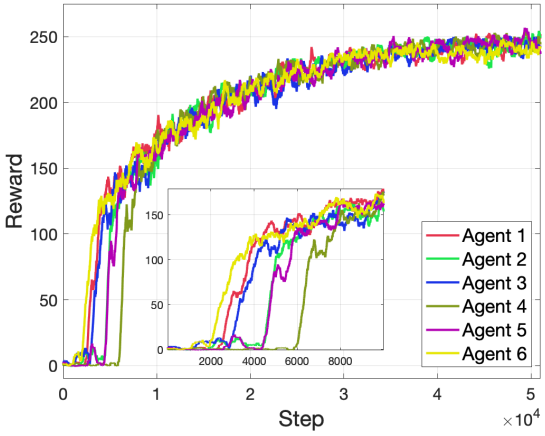


(b)

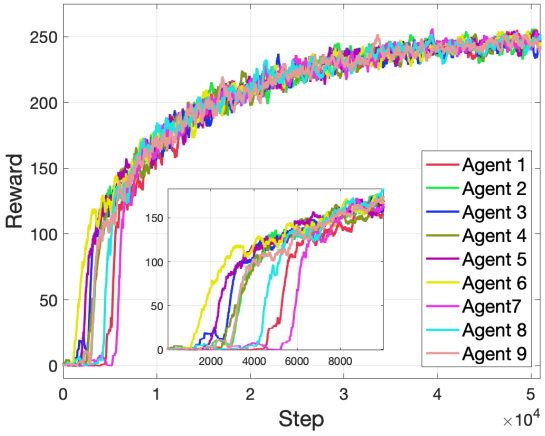


(c)

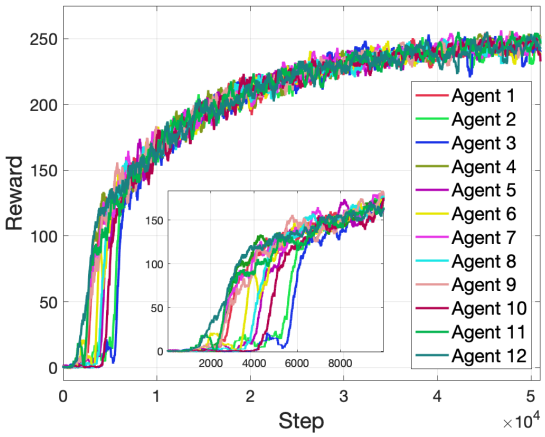
Figure 5.8: Average reward graph of the algorithm with 4 UR5 robot arms compared with single-agent DRL. (a) The feedback gain is set to 0.9. (b) The feedback gain is set to 0.1. (c) The interaction topology is completely connected with feedback gain set to 0.9.



(a)



(b)



(c)

Figure 5.9: Average reward graph of the algorithm with three groups of different numbers of UR5 robot arms. (a) Performing consensus-based distributed training with 6 UR5 robot arms at every 1 step. (b) Performing consensus-based distributed training with 9 UR5 robot arms at every 1 step. (c) Performing consensus-based distributed training with 12 UR5 robot arms at every 1 step.

in Fig. 5.8 (a). In the case of nine UR5 robot arms with the same algorithm, the required convergence steps descend to around 6900 steps, as shown in Fig. 5.9 (b). In Fig. 5.9 (c), only around 6300 steps are required to converge to the optimal value if twelve UR5 robot arms participate in training. As a result, a larger number of UR5 robot arms participating in this algorithm leads to faster convergence to the optimal result.

Fig. 5.11 and Fig. 5.12 reveal the variation of the consensus error of the actor and critic networks correspond to the interaction topology as shown in Fig. 5.10. We can see that the consensus errors of both actor and critic converge to zero and the convergence speed will increase if the communication between the agents is strengthened, as shown in Fig. 5.13.

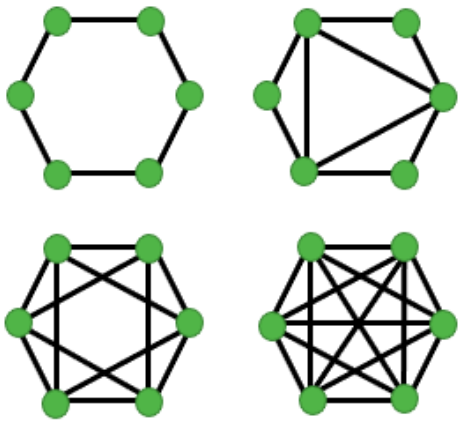
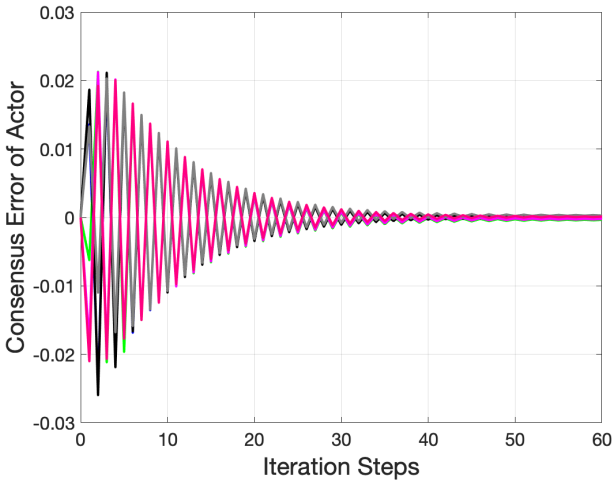


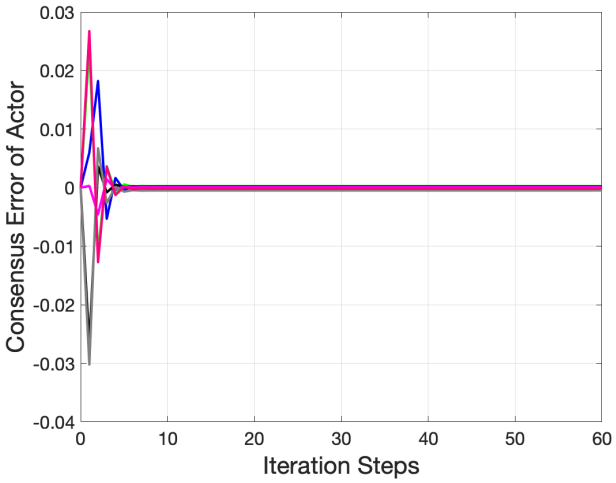
Figure 5.10: Interaction topology of 6 agents, with the Fiedler eigenvalues equal to 1, 1.7, 4, and 6.

Evaluation of Performance Tests

In order to compare and contrast the performance of the proposed algorithm, 50 trials were tested, varying the number of UR5 robot arms with models at different training steps. Fig. 5.14 depicts the number of times that each UR5 robot arm reaches the target point subjected to different scenarios. It can be seen that a larger number of UR5 robot arms participating in the training process results in a larger number of times that each UR5 robot arm reaches the target point in all cases, which can be



(a)

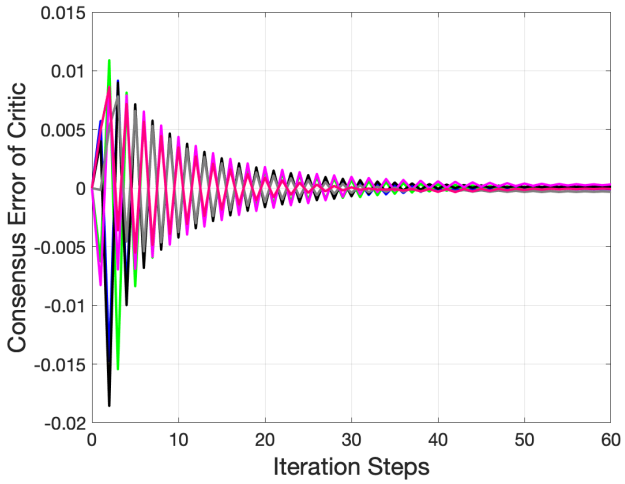


(b)

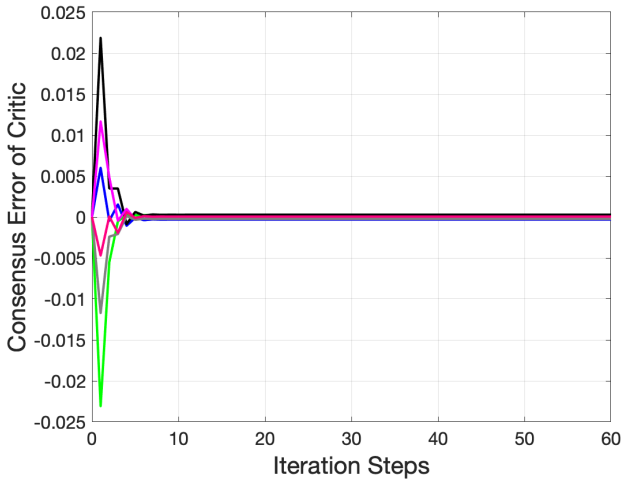
Figure 5.11: Consensus error of the actor network with 6 UR5 robot arms. (a) With the Fiedler value equal to 1 (b) With the Fiedler value equal to 6

viewed as a significant factor that controls the consensus speed of the proposed algorithm. Fig. 5.15 illustrates how a group of four UR5 robot arms reach the target point. The random target point is represented by the location of the wooden box in Gazebo. Each UR5 robot arm has the same default initial position. If any UR5 robot arm reaches the target point, its position will be reset to the default initial position for the next training and the target point will be reset to another random position.

It can be inferred from the experiments that this proposed algorithm makes all UR5 robot arms converge to the optimal model. In addition, the performance of all UR5



(a)



(b)

Figure 5.12: Consensus error of the critic network with 6 UR5 robot arms. (a) With the Fiedler value equal to 1 (b) With the Fiedler value equal to 6

robot arms with this proposed algorithm is better than the performance of single UR5 robot arm training. The consensus speed of all UR5 robot arms depends on the connectivity of interaction topology and the number of training agents. The consensus speed of this proposed algorithm can be refined by either increasing the connectivity of interaction topology in consensus-based distributed training or letting more agents participate in training.

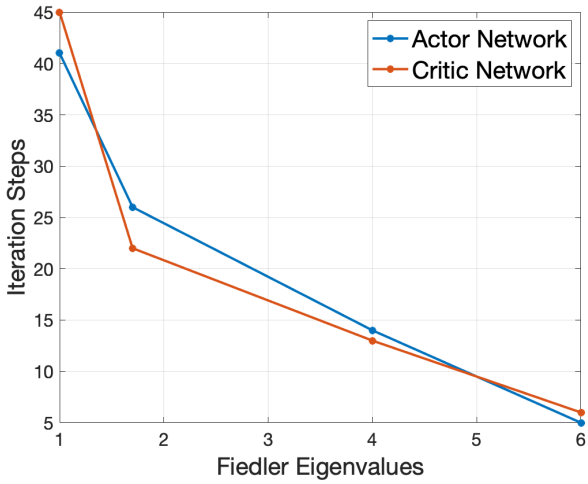


Figure 5.13: Iteration steps versus Fiedler eigenvalues of the actor network and the critic network with 6 UR5 robot arms. The Fiedler eigenvalues are calculated by the interaction topology shown in Fig. 5.10.

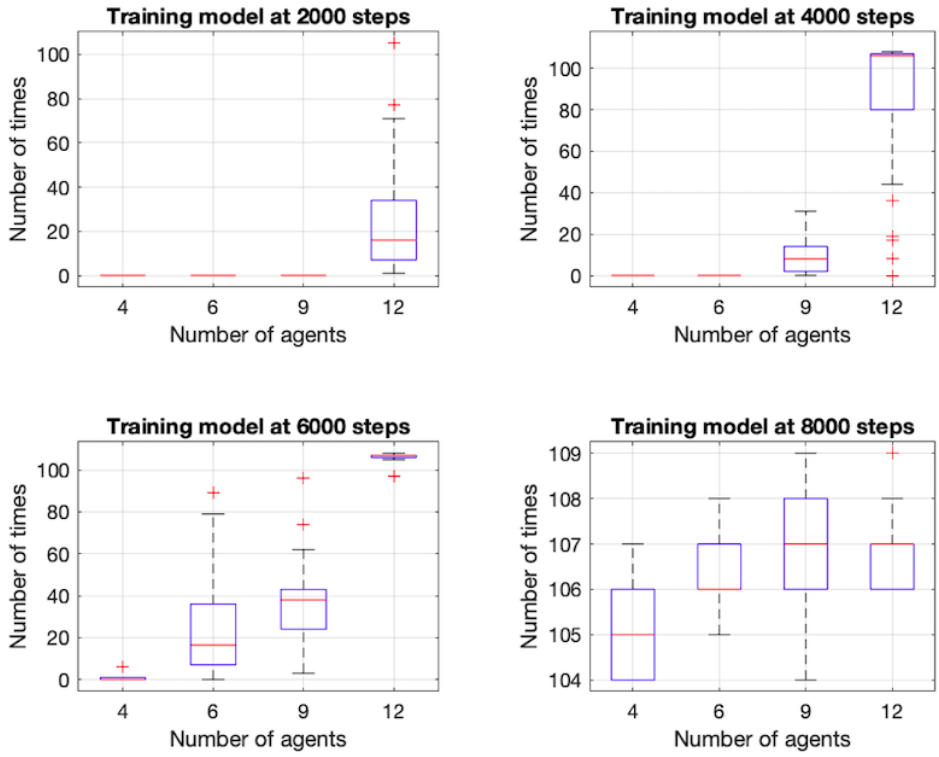


Figure 5.14: Box plot of the number of times that each UR5 robot arm reaches the target point in 100 s with models at different training steps. The results in each case are collected from 50 trials with random target positions.

Evaluation of Trajectory Learning

As mentioned in (5.60), the reward space consists of a distance reward, an arrive reward and a smoothness reward. By introducing a smoothness reward in the reward

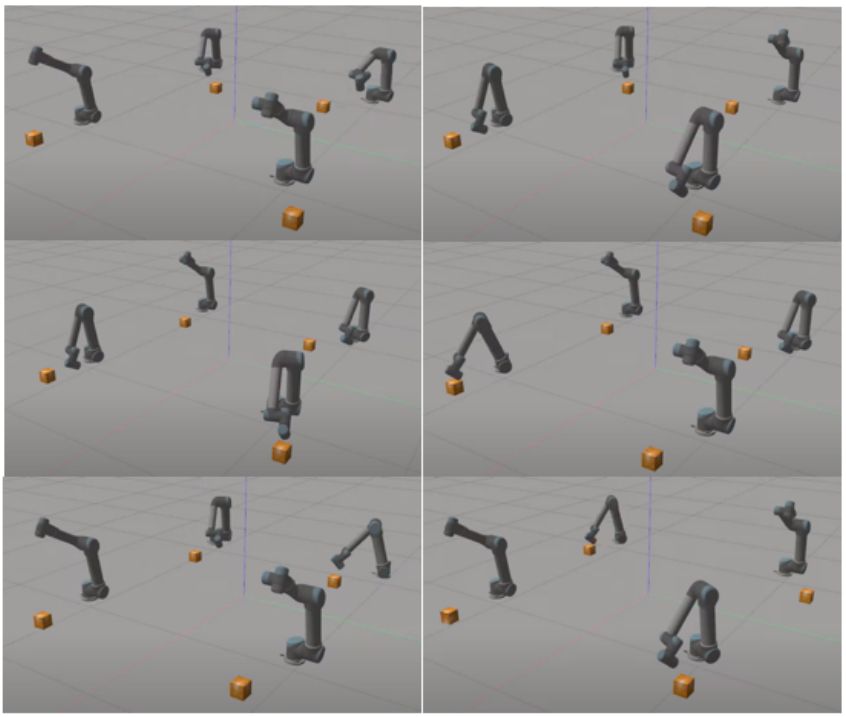
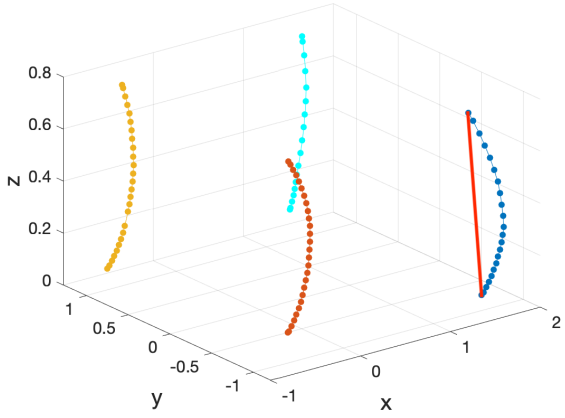


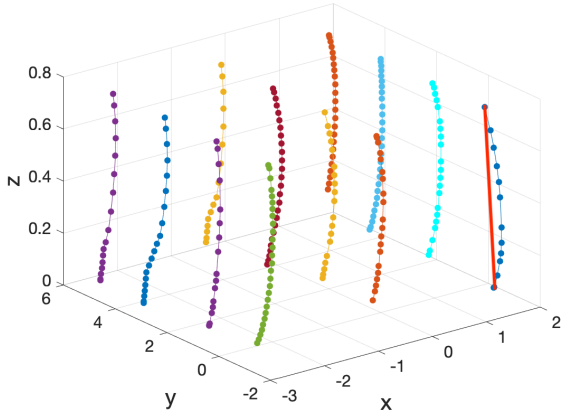
Figure 5.15: Performance tests of consensus-based distributed training with 4 UR5 robot arms.

space, each UR5 robot arm is able to reach the random target position in a smooth trajectory without jerk. Different from standard path planning methods, this algorithm can not only ensure that all angles of the UR5 robot arm are within the allowable range when the end effector of the UR5 robot arm reaches the target point, but also guarantee that all angles of the UR5 robot arm are within the allowable range during the entire process of reaching the random target point.

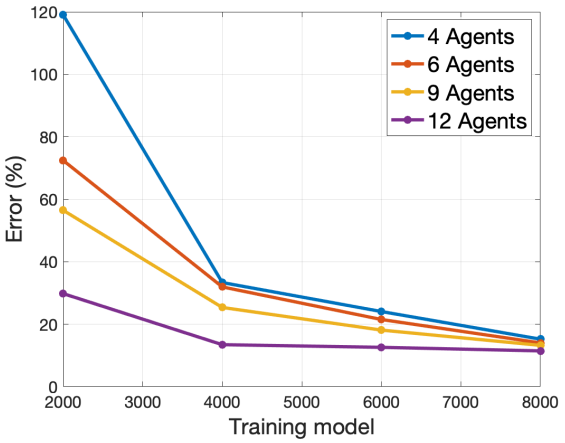
The performance of smooth path planning learning of each UR5 robot arm is validated in Fig. 5.16. Fig. 5.16 (a) and Fig. 5.16 (b) demonstrate the trajectories of the end effector of different numbers of UR5 robot arms with different training models at one step. The dotted line stands for the trajectories of the end effector of each UR5 robot arm, which is called the actual line. Fig. 5.16 (a) represents the trajectories of the end effector of four UR5 robot arms with the training models at 6000 steps. Fig. 5.16 (b) depicts the trajectories of the end effector of twelve UR5 robot arms with the training models at 8000 steps. The red straight line that connects the start and the end point in both Fig. 5.16 (a) and Fig. 5.16 (b) represents the shortest distance of each UR5 robot



(a)



(b)



(c)

Figure 5.16: (a) Trajectories of the end effector of 4 UR5 robot arms with the training models at 6000 steps. (b) Trajectories of the end effector of 12 UR5 robot arms with the training models at 8000 steps. (c) Error rate of the trajectories of the end effector of different numbers of UR5 robot arms with different training models at one step. The result in each case is collected from 50 trials with random target positions.

arm, which is called the reference line. Compared with Fig. 5.16 (a), the trajectories of the end effector of UR5 robot arms in Fig. 5.16 (b) are closer to the reference line. The error rate of each UR5 robot arm in Fig. 5.16 (c) is computed as follows:

$$e_r = \left| \frac{l_a - l_r}{l_r} \right| \tag{5.64}$$

where e_r represents the error rate of each UR5 robot arm, l_a is the length of the actual line of each UR5 robot arm and l_r stands for the length of the reference line of each UR5 robot arm.

The error rate of the end effector trajectories of different numbers of UR5 robot arms with different training models is shown in Fig. 5.16 (c). As can be seen from the graph, the error rate of each UR5 robot arm descends as the number of training steps ascends. Moreover, more agents involved in the proposed algorithm leads to less error rate and more stable performance.

It can be inferred from the experiment that after doing consensus to the actor training parameter and the critic training parameter, for each UR5 robot arm, the critic is more likely to give bigger reward to the action that makes the UR5 robot arm closer to the target position. Because of the off-policy algorithm, the actor is more likely to repeat the joint angles that give high rewards, which are closer to the target position. Therefore, the UR5 robot arm is able to learn a smooth path planning trajectory to the random target point and learn how to reach the random target point with the shortest allowable trajectory as the number of training steps increases.

The first two layers of the actor network and the first three layers of the critic network are fixed with the optimal weight in order to satisfy the condition listed in (5.9) and (5.10). If all the previously fixed layers are randomly initialized and subjected to the consensus algorithm, our algorithm is still applicable. This will be a non-convex optimization problem that we could expand on in future work.

5.3.5 Comparison with Existing Multi-agent Algorithm

We confirm the validity of our multi-agent DRL framework using the algorithm proposed in [3]. The key idea in [3] is to update both actor and critic training parameters during each training iteration and only apply consensus-like training to each local policy training parameter. Different from [3], our algorithm is able to update both actor and critic training parameters simultaneously with consensus-based distributed training. Consequently, our proposed method is completely implemented under a distributed training protocol with DRL. The average reward graph of four UR5 robot arms shows that our proposed method (as shown in Fig. 5.8 (a)) converges faster than the algorithm proposed in [3] (as shown in Fig. 5.17), which validates the efficiency of our proposed algorithm.

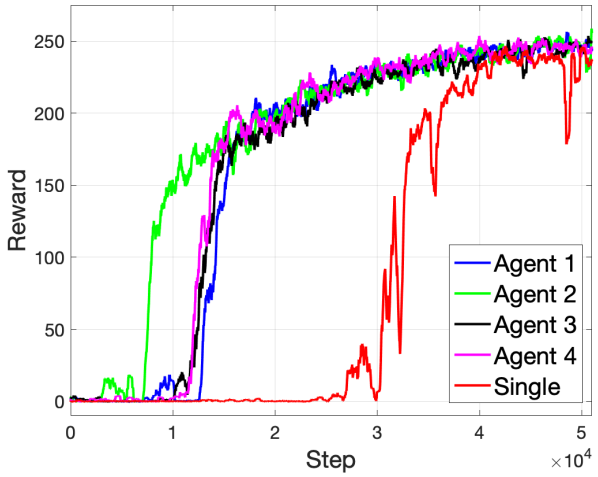


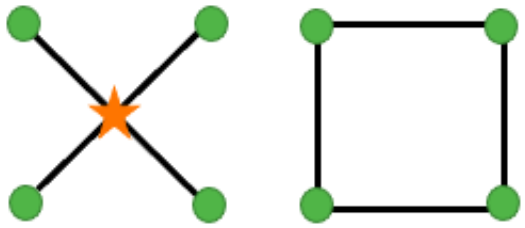
Figure 5.17: Average reward graph of 4 UR5 robot arms compared to single-agent DRL with the algorithm proposed in [3].

5.3.6 Discussion on Bandwidth and Privacy Protection

Discussion on Bandwidth

Existing methods [175; 176] are mostly designed for a centralized graph, in which there is a central agent connected with multiple agents to share information and parameters, as demonstrated in Fig. 5.18 (a). Our method focuses on a decentralized topology which has better scalability without increasing the communication bandwidth when

dealing with a large number of agents, as shown in Fig. 5.18 (b). The potential bottleneck of using a centralized topology is the communication traffic jam since the central agent needs to receive information from all other agents and send information back to them during each training iteration. If a communication traffic jam occurs on the central agent, it will cause severe data loss. With a limited communication bandwidth, the number of agents that the central agent can support each time is limited. Nevertheless, by implementing a decentralized topology, the number of agents that can be supported each time is much larger under the same communication bandwidth. Compared to the centralized topology, the decentralized topology avoids the communication traffic jam on the central agent, which makes it possible to tackle more agents training without increasing the internet bandwidth.



(a) (b)

Figure 5.18: Examples of graph topologies. The star denotes the central agent. (a) Centralized (b) Decentralized

Discussion on Privacy Protection

In practical applications, we cannot directly collect users' data to train a RL model due to the fact that these data contain users' privacy. For instance, if the heads of multiple hospitals want to jointly train a RL model for disease diagnosis, due to patient privacy issues, the hospital heads cannot directly share the patients' medical data with each other. To avoid the issue of user privacy leakage, training weights are shared between different agents with our proposed method instead of users' data since training weights are processed numbers. Compared with sharing users' data directly, sharing training weights between different agents are more secure and reliable.

5.4 Summary

A novel multi-agent training framework is proposed to support the training of multiple UR5 robot arms. In order to handle more agents training without increasing internet bandwidth and protect the privacy of each UR5 robot arm, a novel multi-agent training algorithm with actor-critic based off-policy DRL and consensus-based distributed training is proposed. The convergence analyses of the actor training parameter and the critic training parameter with Lyapunov method are provided to confirm the validity of the proposed training algorithm. By setting up appropriate reward space, action space and observation space, the experiments are provided to train each UR5 robot arm to reach the random target position smoothly. Compared with single UR5 robot arm DRL, the performance of all UR5 robot arms is better when this proposed algorithm is used. The consensus speed of this algorithm can be tuned by switching to a larger feedback gain, changing to an interaction topology with higher connectivity and adding more agents in the training process. The UR5 robot arm can learn how to reach the random target point with the shortest allowable trajectory as the number of training steps increases. In the future, an optimization strategy of our algorithm will be exploited to improve the training efficiency of all agents.

Chapter 6

Sim-and-Real Reinforcement Learning for Manipulation: A Consensus-based Approach

In the previous chapter, we focus on the theoretical development of multi-agent DRL with consensus-based training. As an essential component in robotic control, DRL has been widely used in various applications [428; 429; 430]. The training process of DRL [115] builds the bridge between the environment state and the action, thereby maximizing the cumulative reward. Learning from the simulation is safer, cheaper and faster while learning from the real world is more dangerous, expensive and slower. If the simulation shows high fidelity, the training model in the simulation can be transferred directly to the real world. However, in many circumstances, the simulation cannot mimic the real world very well, which limits robot performance in the real world. To overcome this difficulty, we develop a sim-and-real training method to balance the relationship between the simulation and the real world. We use concepts from control engineering, i.e. consensus [431; 233], to accomplish sim-and-real training.

In this chapter, we propose a CSAR algorithm that combines consensus-based training with DRL in a sim-and-real environment, as shown in Fig. 6.1. We apply CSAR to a group of simulated agents together with a real agent each learning to carry out a pick-and-place task with a suction robot device. Compared to conventional sim-to-real training method, the challenges of CSAR DRL are 1) information exchange between

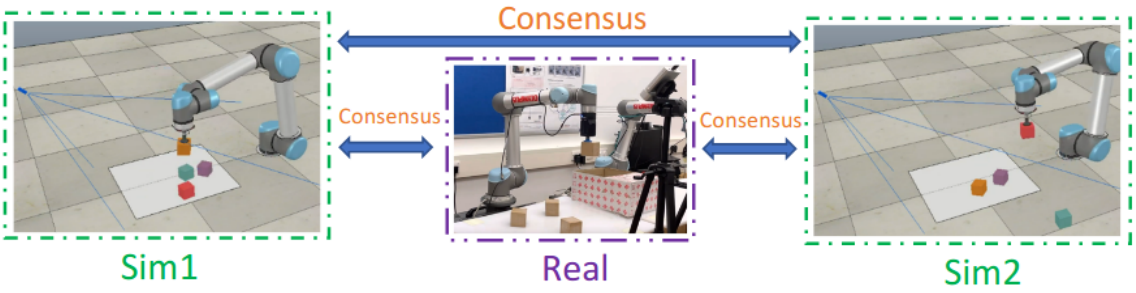


Figure 6.1: Pick-and-place objects with the CSAR approach

simulated and real robots, for instance, generating communication in a mixed environment, 2) data-efficient collection for training in a sim-and-real environment such as handling data from multiple robots simultaneously, 3) data pre-labelling for suctioning in a sim-and-real environment, for example, using aruco makers to locate suctioned objects.

To overcome these difficulties, a complete CSAR method is proposed for manipulators to learn pick-and-place tasks. By applying consensus-based training, the proposed method saves training time and reduces the number of required real robot training steps while maintaining a comparable suction success rate, which is cost-effective. Moreover, an end-to-end and lightweight NN is proposed to train the suction policy, which uses raw 3D visual data directly without pre-labelling. The effectiveness and feasibility of the CSAR method are validated through simulation and real-world experiments. We extend the consensus approach [224] from theory and simulations to a real-world pick-and-place problem and show the effectiveness of the proposed approach.

6.1 Methodology

We extend the consensus-based approach in [224], which only focuses on simulations, to sim-and-real scenarios. The proposed effective and efficient CSAR method can increase sim-and-real training speed as well as save real-world training costs with consensus-based training.

6.1.1 System Overview

Fig. 6.2 describes the overview of our proposed framework. The predefined workspace in the simulation is captured by a fixed simulated camera, which provides an ideal RGB-D image each time. Then the ideal RGB-D image is orthographically projected in the direction of gravity to construct the colour heightmap $\bar{c}_t$ and the depth heightmap $\bar{d}_t$, which are the inputs of our framework. Both heightmaps are fed into the Q-function NN to anticipate pixel-wise best suction position $[\bar{x}_t, \bar{y}_t]$. Given the specific use of these NNs modelling the Q-function for pick and place success through suction gripping, we may call these "suction networks". The suction height $\bar{z}_t$ can be found from $\bar{d}_t$.

When it comes to the real world, the predefined workspace is captured by a fixed azure kinect camera. Compared with the ideal RGB-D image which is obtained from the simulated camera, the real-world RGB-D image contains more camera distortion [432]. Similarly, the real-world RGB-D image is orthographically projected in the direction of gravity to construct the colour heightmap $\tilde{c}_t$ and the depth heightmap $\tilde{d}_t$ which are also fed into the suction network to predict real-world pixel-wise best suction position $[\tilde{x}_t, \tilde{y}_t]$. The suction height $\tilde{z}_t$ can be also acquired from $\tilde{d}_t$.

After performing predictions in both environments, consensus-based training is applied to the training parameters of each simulated or real agent. The suction process of each agent is carried out in parallel, which saves training time.

6.1.2 Deep Reinforcement Learning Setup

In this section, we illustrate action space, state space, reward space, NN structure and loss function of the CSAR method.

Action Space

The action space a_t is a Cartesian motion command that consists of pixel-wise best suction position. In the simulated environment, $\bar{a}_t = [\bar{x}_t, \bar{y}_t, \bar{z}_t]$. Correspondingly, $\tilde{a}_t = [\tilde{x}_t, \tilde{y}_t, \tilde{z}_t]$ in the real world. The suction height $\bar{z}_t$ and $\tilde{z}_t$ can be acquired from $\bar{d}_t$ and $\tilde{d}_t$.

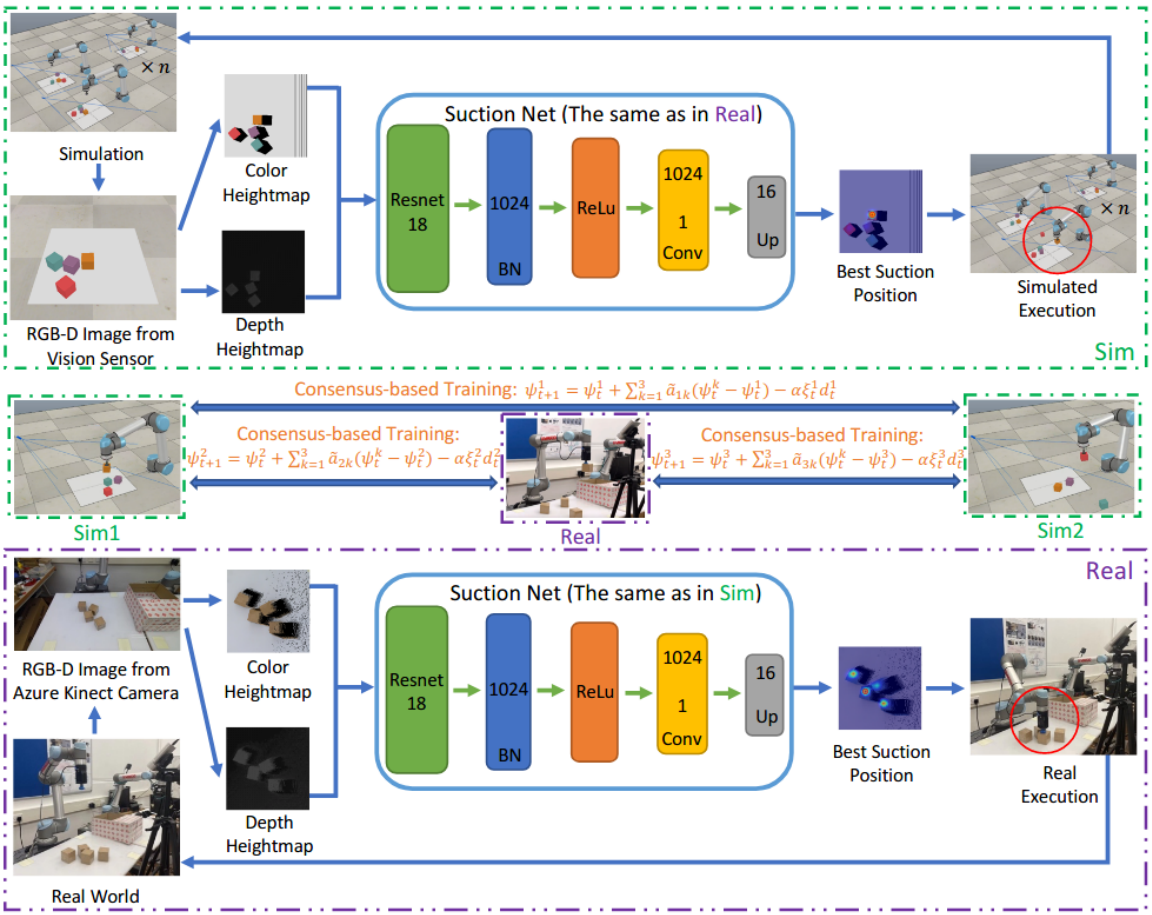


Figure 6.2: Overview of the proposed DRL framework with consensus-based training in the sim-and-real environment (substantiation of Figure. 6.1). During each iteration, consensus-based training is applied to the training parameters of every suction net (multi-layer NN modelling the Q-function for pick-and-place success through suction gripping). The suction executions occur simultaneously in both simulated and real environments. BN represents Batch Normalization. Conv stands for Convolution. Up represents Upsampling. More details can be found in Algorithm 6.1.

State Space

As shown in Fig. 6.2, the state space s_t denotes the colour heightmap and depth heightmap of the captured RGB-D image. In the simulated environment, $\bar{c}_t$ and $\bar{d}_t$ are acquired by the fixed simulated camera. In the real environment, $\tilde{c}_t$ and $\tilde{d}_t$ can be obtained from the fixed azure kinect camera.

Reward Space

The distance μ_m in the simulated environment can be computed by

$$\mu_m = \sqrt{(\bar{x}_m - \tau_m)^2 + (\bar{y}_m - \sigma_m)^2} \tag{6.1}$$

where τ_m and σ_m denote x, y positions of the centre of the expected suctioned object of the m^{th} agent, respectively.

We assign suction reward $r_s = 1$ if the target is successfully suctioned, otherwise $r_s = 0$. Thus, the DRL reward $\bar{r}_m$ for each agent in the simulation can be defined as

$$\bar{r}_m = \begin{cases} r_s r_0 & \text{if } \mu_m \leq \mu_{th} \\ r_s r_1 & \text{if } \mu_{th} < \mu_m \leq 2\mu_{th} \\ r_s r_2 & \text{if } 2\mu_{th} < \mu_m \leq 3\mu_{th} \\ r_s r_3 & \text{if } \mu_m > 3\mu_{th} \end{cases} \tag{6.2}$$

where $\bar{r}_m$ stands for the reward of the m^{th} agent in the simulated environment, μ_{th} represents distance threshold of the m^{th} agent, r_0, r_1, r_2 and r_3 are the positive reward when μ_m is within the corresponding range.

The DRL reward $\tilde{r}_m$ for each agent in the real environment is given by

$$\tilde{r}_m = r_s r_0 \tag{6.3}$$

Algorithm 6.1 CSAR: Consensus-based Sim-and-Real DRL

Initialize the m^{th} agent training parameter ψ_t^m, learning rate α, RGB-D image $\bar{g}_t^m$ from the simulation, initial RGB-D image $\tilde{g}_t^m$ from the real world, discounted factor γ, total training steps parameter T.
while $t < T$ **do**
  Generate $\bar{c}_t^m$ and $\bar{d}_t^m$ from $\bar{g}_t^m$.
  Generate $\tilde{c}_t^m$ and $\tilde{d}_t^m$ from $\tilde{g}_t^m$.
  if object count $O_t^m <$ empty threshold **then**
    Feed $\bar{c}_t^m$ and $\bar{d}_t^m$ into the m^{th} suction network to generate action-value function $Q(\psi_t^m, \bar{s}_t^m, \bar{a}_t^m)$.
    Feed $\tilde{c}_t^m$ and $\tilde{d}_t^m$ into the m^{th} suction network to generate action-value function $Q(\psi_t^m, \tilde{s}_t^m, \tilde{a}_t^m)$.
    if $t > 2$ **then**
      Generate r_t^m with $Q(\psi_{t-1}^m, \bar{s}_{t-1}^m, \bar{a}_{t-1}^m)$ and $Q(\psi_{t-1}^m, \tilde{s}_{t-1}^m, \tilde{a}_{t-1}^m)$.
      Compute ξ_{t-1}^m:

$$Y_{t-1}^m = r_t^m + \gamma \max_a (Q(\psi_{t-1}^-, s_t^m, a^m)).$$
$$\xi_{t-1}^m = Q(\psi_{t-1}^m, s_{t-1}^m, a_{t-1}^m) - Y_{t-1}^m.$$

      For M agents, update the training parameters ψ_t with consensus-based training:

$$\psi_t = \mathcal{C}\left(\psi_{t-1}, \mathcal{L}\right) - \alpha \Gamma_{t-1}.$$

      Sample a batch from the replay buffer R_p to implement experience replay.
    end if
    Perform suction execution in both simulated and real environments in parallel.
    Store $(\bar{c}_t^m, \bar{d}_t^m, \bar{a}_t^m)$ and $(\tilde{c}_t^m, \tilde{d}_t^m, \tilde{a}_t^m)$ in R_p.
  else
    Reposition objects.
  end if
end while

Neural Network Structure

As stated in Fig. 6.2, the input of the suction net passes data through ResNet-18 [412] to extract concatenated features from the colour heightmap and the depth heightmap. The aforementioned features are fed into a Batch Normalization layer [413] with 1024 input features, a ReLu layer [413], a Convolution layer [413] with 1024 input channels, and 1 output channel, then are processed by a bilinear upsample layer [413] with a scale factor of 16. The output of the suction net has the same image size as the heightmap input, which is a dense pixel-wise map of different Q values. The pixel which has the maximum Q value represents the best suction position.

Remark 1. *It should be noted that the suction net can be substituted by any state-of-the-art NN. Since we use a standard laptop for training, we purposely design a lightweight version of the suction net inspired by [126].*

During each training iteration t, the training objective is to minimize the temporal difference error ξ_t [347]:

$$\xi_t = Q(\psi_t, s_t, a_t) - Y_t \tag{6.4}$$

where $Y_t = r_{t+1} + \gamma \max_a(Q(\psi_t^-, s_{t+1}, a))$ and a represents all available actions, γ stands for the discount factor, Q represents the action-value function, r is the reward, ψ_t stands for the training parameters of the suction network at time t, ψ_t^- denotes the target training parameters.

Loss function

Inspired by [126], we use the Huber loss function [433] to train our proposed suction network in both simulated and real environments. The loss function Ω at the t^{th} iteration can be computed as follows:

$$\Omega_t = \begin{cases} \frac{1}{2}(\xi_t)^2 & \text{if } |\xi_t| < 1 \\ |\xi_t| - \frac{1}{2} & \text{otherwise} \end{cases} \tag{6.5}$$

Gradients are only passed through the single pixel on which the action is executed during each iteration t. All other pixels propagate with 0 loss [126].

6.1.3 Consensus-based Training

The Q-function of each simulated or real agent is trained through a consensus-based algorithm. Hence, we wish to introduce first the consensus network structure which facilitates that training process. The interaction topology of M agents can be depicted by an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ represents a vertex set $\mathcal{V} = \{1, 2, \cdots, M\}$ and $\mathcal{E}$ stands for an edge set $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$. The edge $(j, m) \in \mathcal{E}$ if the j^{th} and m^{th} agents are connected with one another [409]. The adjacency matrix $\mathcal{A}$ of $\mathcal{G}$ can be described as $\mathcal{A} = [a_{jm}] \in \mathbb{R}^{M \times M}$, where $a_{jm} > 0$ if $(j, m) \in \mathcal{E}$, otherwise $a_{jm} = 0$. Hence, the Laplacian matrix $\mathcal{L}$ of $\mathcal{G}$ is defined as $\mathcal{L} = \mathcal{D} - \mathcal{A}$, where $\mathcal{D} = diag\{d_{11}, \cdots, d_{MM}\} \in \mathbb{R}^{M \times M}$ and $d_{jj} = \sum_{j \neq m} a_{jm}$ [434]. For an undirected topology, $\mathcal{L}$ is positive semi-definite. $\mathcal{L}\mathbf{1}_M = 0$, where $\mathbf{1}_M = [1, \cdots, 1]^\top$. If the graph $\mathcal{G}$ has a spanning tree, the rank of $\mathcal{L}$ should be $M - 1$ [434].

For an undirected graph $\mathcal{G}$, if $\hat{\chi}_m \in \mathbb{R}^n$ represents the updated training parameter of $\chi_m \in \mathbb{R}^n$ after a single consensus step and χ_m stands for the row vector of the training parameter for agent m in the graph, the consensus training step of each agent m can be described as

$$\hat{\chi}_m = \chi_m + u_m \tag{6.6}$$

$$u_m = \sum_{k=1}^{M} a_{mk}(\chi_k - \chi_m) \tag{6.7}$$

where a_{mk}, the element of the graph adjacency matrix, is engendered by the undirected graph $\mathcal{G}$ and u_m stands for the input of the agent m.

By integrating (6.7) and (6.6), the consensus algorithm can be used to update an agent m in the following scheme:

$$\begin{aligned} \hat{\chi}_m &= \chi_m + \sum_{k=1}^{M} a_{mk}(\chi_k - \chi_m) \\ &= \chi_m - \sum_{k=1}^{M} l_{mk}\chi_k \\ &= \mathcal{C}_m\left(\chi_k, l_{mk}\right) \end{aligned} \tag{6.8}$$

where l_{mk} is the element of the Laplacian matrix $\mathcal{L}$ and $\mathcal{C}_m$ represents the consensus protocol of the m^{th} agent. The training parameter update of all the M agents with a

single consensus step can be summarised as

$$\begin{aligned}\hat{\chi} &= ((I_M - \mathcal{L}) \otimes I_n)\chi \\ &= \mathcal{C}(\chi, \mathcal{L})\end{aligned} \tag{6.9}$$

where $\mathcal{C}$ stands for the consensus protocol for all agents, I_M and I_n denote the $M \times M$ and $n \times n$ identity matrix, $\mathcal{L}$ represents the Laplacian matrix. By repetitively computing (6.9), this consensus algorithm makes all agents converge to their weighted average [418].

6.1.4 Consensus-based Training with Deep Reinforcement Learning

Given the consensus network structure in the previous sub-section, the training algorithm for the training parameters ψ_t in (6.4) in the DRL is now introduced. As stated in [435], the process of updating ψ_t for the m^{th} agent is given as:

$$\psi_{t+1}^m = \psi_t^m - \alpha \xi_t^m \frac{dQ(\psi_t^m, s_t^m, a_t^m)}{d\psi_t^m} \tag{6.10}$$

where α represents the learning rate.

By applying (6.8), the training process of the CSAR algorithm can be summarised as:

$$\hat{\psi}_t^m = \psi_t^m + \sum\nolimits_{k=1}^{M} \tilde{a}_{mk}(\psi_t^k - \psi_t^m) \tag{6.11}$$

$$\psi_{t+1}^m = \hat{\psi}_t^m - \alpha \xi_t^m d_t^m \tag{6.12}$$

where $d_t^m = \frac{dQ(\psi_t^m, s_t^m, a_t^m)}{d\psi_t^m}$.

Substituting (6.11) into (6.12), we can get

$$\psi_{t+1}^m = \psi_t^m + \sum\nolimits_{k=1}^{M} \tilde{a}_{mk}(\psi_t^k - \psi_t^m) - \alpha \xi_t^m d_t^m \tag{6.13}$$

Let $\Gamma_t = [\xi_t^1 d_t^1, \xi_t^2 d_t^2, \cdots, \xi_t^M d_t^M]^T$, for M agents, the update of the training parameters in our suction network in the t^{th} iteration can be illustrated as

$$\begin{aligned}\psi_{t+1} &= ((I_M - \mathcal{L}) \otimes I_n)\psi_t - \alpha\Gamma_t \\ &= \mathcal{C}(\psi_t, \mathcal{L}) - \alpha\Gamma_t\end{aligned} \tag{6.14}$$

Algorithm 6.1 summarizes our CSAR algorithm.

6.2 Experiments and Results

The feasibility of the CSAR algorithm is validated in this section. The system is implemented on a standard laptop with Nvidia GTX 2070 super and Intel Core i7 CPU (2.6 GHz) with 16 GB RAM. The experimental video is available at: https://youtu.be/mcHJtNIsTEQ.

6.2.1 Experiment Setup

Simulation

Our system in the simulated environment is trained in Coppeliasim [404] with Bullet Physics 2.78 for dynamics, as demonstrated in Fig. 6.1. The simulation setup for each agent consists of a UR5 robot arm with a suction gripper [415]. The suctioned objects in the simulated environment are cubes with a side length of 5 cm. The motion planning task for each UR5 robot arm is accomplished by Coppeliasim [404] internal IK. Simulated cameras are used to capture RGB-D images of each agent in a 0.448×0.448 m^2 workspace. The resolution of the simulated RGB-D images is 640×480.

Real World

The setup for each agent in the real environment is composed of a UR5 robot arm with a Robotiq EPick vacuum gripper. The suctioned objects are cubes with a side length of 6.5 cm. To pick and place objects successfully with the suction gripper in the sim-and-real environment, the objects should have a flat surface and no overlap between objects placed in the workspace. We use a fixed Azure Kinect camera to acquire real-world RGB-D images with a resolution of 1280×720. The location of the Azure Kinect camera is shown in Fig. 6.1, which can generate a top-down view in a 0.448×0.448 m^2 workspace.

Reward

Depending on the intrinsic and distortion of the Azure Kinect camera and the size of our suction gripper, we assign $r_0 = 2000$, $r_1 = 1000$, $r_2 = 100$, $r_3 = 1$ and $\mu_{th} = 0.005$ m in (6.2). These values can also be reconfigured for other robotic platforms.

Neural Network

The proposed framework is fully trained under self-supervision through the interactions between the UR5 robot arms and the sim-and-real environment. The learning rate α in (6.10) has a fixed value of 0.0001. The discounted factor γ listed in (6.4) is set to 0.5. The future reward discount is fixed at 0.5. The total training steps parameter T is initialized at 270. Algorithm 6.1 satisfies ϵ-greedy exploration strategy with ϵ initialized at 0.5 and annealed to 0.1 over training. The simulated camera and the Azure Kinect camera capture RGB-D images to generate colour and depth heightmaps, which are fed into the suction nets to predict pixel-wise best suction positions.

Evaluation Metric

The suction performance of the m^{th} agent can be evaluated using the suction success rate S_r^m, which is defined as follows:

$$S_r^m = \frac{N_s^m}{N_i^m} \times 100\% \tag{6.15}$$

where N_s^m represents the number of successful target suctions of the m^{th} agent, N_i^m represents the number of iterations of the m^{th} agent.

We explore various training strategies to discover the most suitable training conditions for robots:

Sim-and-Real: Only simulation samples are used to train and optimise the model initially. When the suction success rate in the simulation reaches 0.5, we switch to the CSAR method with 3 **sim**ulated robots and 1 **real** robot.

Sim-to-Real: Only simulation samples are used to train and optimise the model at the beginning. When the suction success rate in the simulation reaches 0.5, we switch to real-world training with 1 real robot.

6.2.2 Sim-and-Real is Better Than Sim-to-Real

Fig. 6.3 demonstrates the suction success rate of the real robot using two different training strategies. The interaction topology of Sim-and-Real is shown in Fig. 6.4 (c). When applying the Sim-and-Real strategy, the suction success rate of the real

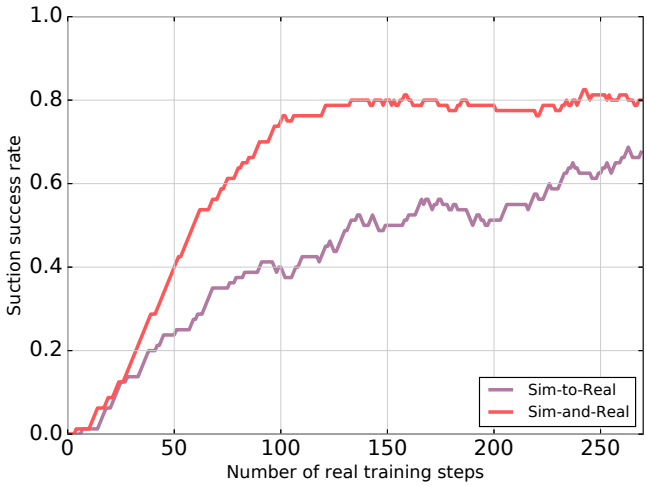
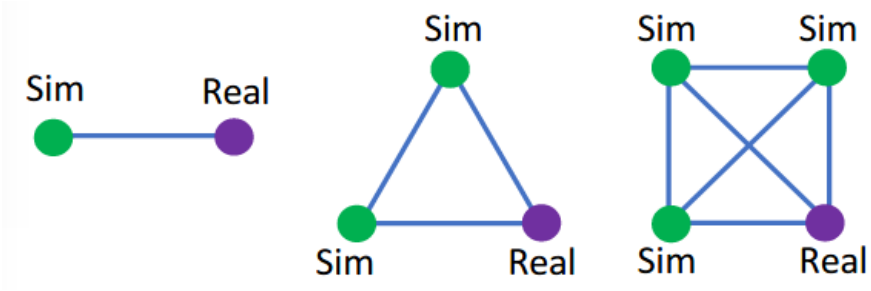


Figure 6.3: Suction success rates of the real robot between "Sim-to-Real" and "Sim-and-Real" strategies



(a) (b) (c)

Figure 6.4: Topology of the interaction of simulation and the real world: (a) 1 **sim**ulated robot and 1 **real** robot; (b) 2 **sim**ulated robots and 1 **real** robot; (c) 3 **sim**ulated robots and 1 **real** robot

robot reaches 80% at around 140 training steps, which outperforms the Sim-to-Real strategy. Since our policy for each robot is greedy deterministic, a robot may execute the same action repetitively if there is no environment change when using the Sim-to-Real training strategy. However, by applying consensus-based training, the simulated agent can be used to introduce noise indirectly into the sim-and-real environment, which prevents robots from getting stuck in the same action. In summary, applying the Sim-and-Real strategy leads to a faster training speed, which saves real-world training costs.

6.2.3 Best Policy in Simulation is Not the Best for Sim-and-Real Training

A striking observation from our experiment is that the best-obtained policy trained in simulation is not the best pre-trained model to start the co-training between simulated and real robots, as shown Fig. 6.5. When the suction success rate of the pre-trained simulation model is 0.5, the Sim-and-Real strategy achieves the best performance. When the suction success rate drops to 0.3, it takes longer for the real robot to solve the task. Surprisingly, when the suction success rate of the pre-trained simulation model is too high (0.7, 0.9), the performance deteriorates.

This is counterintuitive, as shown in the Sim-to-Real experiment, that the best policy obtained in the simulation is typically the one to be deployed. This observation suggests that the "mediocre" policy is the best for co-training. When the success rate of the pre-trained simulation model is too high, the sim-and-real framework will be initialised at a value that is close to the optimal simulation value. This will take longer to converge to the mixed optimality in a sim-and-real environment. As a result, applying the "mediocre" policy can reduce real robot training costs and save the pre-training time in the simulation.

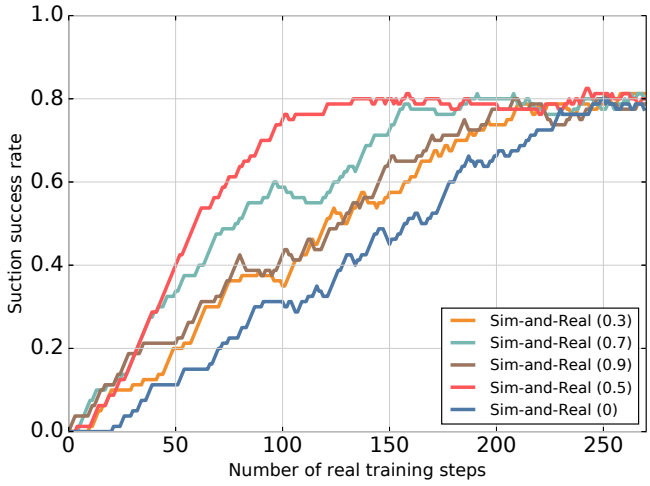


Figure 6.5: Suction success rates of the real robot with different initial weights when applying the Sim-and-Real strategy. The number in brackets denotes the suction success rate from the pre-trained simulation model.

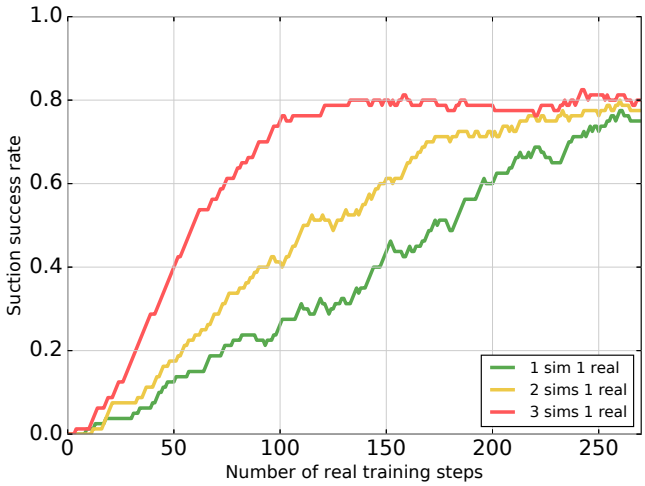


Figure 6.6: Suction success rates of the **real** robot with different number of **sim**ulated robots using Sim-and-Real strategy

6.2.4 The More Agents in Simulation, the Better for Sim-and-Real Training

Readers may wonder why we use 3 **sim**ulated robots and 1 **real** robot during training. Therefore, we vary the number of simulated robots when using the Sim-and-Real strategy. Fig. 6.6 describes the suction success rate when using the Sim-and-Real strategy with different numbers of simulated robots. The interaction topology used in Fig. 6.6 is shown in Fig. 6.4. It takes around 260 steps to make the real robot arrive at 80% suction success rate when using 1 **sim**ulated robot and 1 **real** robot strategy. In the case of 2 **sim**ulated robots 1 **real** robot, the required training steps descend to around 240. Only around 140 steps are required to maintain the same suction success rate when using the 3 **sim**ulated robots 1 **real** robot strategy. More simulated robots participating in the proposed framework can accelerate the training speed and exhibit good robustness in the sim-and-real environment, thus decreasing the number of required real robot training steps while maintaining a comparable suction success rate.

6.2.5 Generalisation of Real-world Unseen Objects

The Sim-and-Real strategy is capable of generalising to novel objects (Fig. 6.7) with a suction success rate of 80%. After training on cubes in both simulated and real environments, the CSAR training model can also be applied to pick and place novel

objects such as cylinders and irregularly shaped objects with different heights, as shown in Fig. 6.8.

(a) (b) (c)

Figure 6.7: Novel objects for validation: (a) Environment 1; (b) Environment 2; (c) Environment 3

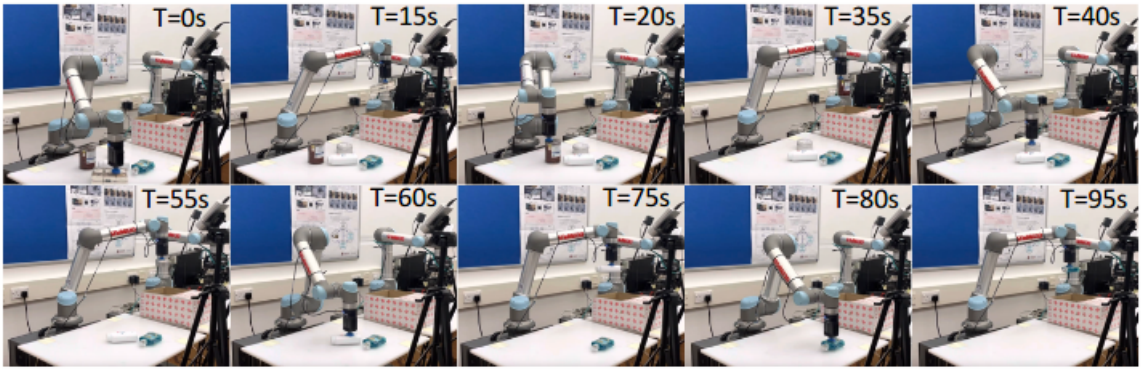


Figure 6.8: The demonstration of picking novel objects. More details can be seen in the video.

6.3 Summary

In this chapter, we propose a CSAR approach which is able to improve sim-and-real training speed and reduce real-world training costs. By implementing the Sim-and-Real strategy, the suction success rate of the real robot attains 80% at around 140 training steps, which outperforms the Sim-to-Real strategy. Applying the "mediocre" policy can not only reduce the number of required real robot training steps but also save the pre-training time in the simulation. More simulated robots participating in the CSAR method increase the training speed, thereby reducing real-world training expenses. The Sim-and-Real strategy is also capable of generalising to novel objects. The CSAR method is a straightforward generalization and practical verification of

the team's recently developed theory of a consensus-based RL approach [224]. In the future, an optimisation of the CSAR approach will be exploited to tackle more complicated scenarios.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

Manipulators have been widely used in various applications such as building blocks assembling [436], picking and placing [246; 126], goal reaching [428], path planning [437], human-robot interaction [410], etc. One ultimate goal of research about manipulators is to explore algorithms that allow manipulators to interact with the environment autonomously, thereby reducing the burden on human operations. In this thesis, we develop four algorithms that enable manipulators to learn autonomously and cooperatively in both simulations and real environments.

First of all, we illustrate a new off-policy DRL method to deal with the problem of path planning of the UR5 robot arm. Different from standard path planning methods, this method is able to guarantee a smooth movement of the UR5 robot arm, and all joint angles of the UR5 robot arm lie within the allowable range during each movement. Moreover, a standard path planning method has been implemented on the real UR5 robot arm as a baseline to compare and contrast the benefits and drawbacks of both methods.

Furthermore, we pay attention to real-world experiments. A complete self-supervised vision-based DRL method is developed for manipulators to learn to pick and place objects. By encouraging the UR5 robot arm to suction the area close to the centre of target objects, the training model with the proposed method is able to accomplish

pick-and-place tasks in the real world with a suction success rate of 90% without any real-world fine-tuning. Specially, a height-sensitive action policy is developed for the proposed self-supervised vision-based DRL method to suction in a challenging environment, i.e., crowded and stacked objects. The performance of the proposed method is validated in both simulated and real environments. The presented approach can also be applied to novel objects with a suction success rate of 90% without any fine-tuning from the real world.

Additionally, we are also interested in extending research from single-agent DRL to multi-agent DRL. A multi-agent training algorithm with actor-critic based off-policy DRL and consensus-based distributed training is developed. The convergence of this algorithm is verified in the presence of the actor training parameter and the critic training parameter. A multi-agent training framework is proposed to support the implementation of the algorithm. Compared with centralized training, the proposed multi-agent training framework has better scalability with a limited communication bandwidth when dealing with a large number of agents and protects the privacy of each agent. The feasibility and efficiency of the proposed algorithm are validated by experiments using several groups of UR5 robot arms.

Last but not least, we focus our attention on multi-agent real-world applications. A complete CSAR method is proposed for manipulators to learn pick-and-place tasks. By applying consensus-based training, the proposed method saves training time and reduces the number of required real robot training steps while maintaining a comparable suction success rate, which is cost-effective. An end-to-end and lightweight NN is proposed to train the suction policy, which uses raw 3D visual data directly without pre-labelling. The effectiveness and feasibility of the CSAR method are validated through simulation and real-world experiments.

In summary, this thesis provides an extensive exploration of the research on DRL with consensus for manipulators. Some potential future work directions which are worth being considered are provided in the next section.

7.2 Future Work

Some potential future work directions are listed as follows:

1. Designing more complex vision algorithms for moving objects. In Chapter 6, RGB-D images are used for DRL vision inputs and all the target objects are static. How to use visual information to deal with dynamic objects should be further investigated.

2. Using different types of grippers to handle various tasks. In this thesis, suction grippers are applied for pick-and-place cubes in the real world. However, two-finger grippers are more suitable for grasping arbitrarily shaped objects such as cups, scissors, etc. Thus, using two-finger grippers to pick and place arbitrarily shaped objects will be conducted in the future.

3. Combining adaptive control with DRL. With the complexity and diversification of industrial field tasks, a single robotic arm can no longer meet the needs of actual working conditions. If adaptive control and DRL are combined to handle nonlinear tasks in MASs, the complexity and cost of the system can be greatly reduced. Due to the huge application prospects of the manipulator system in practice, the study of combining adaptive control and DRL with robotic arms has great potential value for theoretical research and practical application.

4. Making an automatic sorting system with manipulators using DRL. In the traditional industrial production progress, the controller directs the robotic arm to complete production tasks. Since every action of the robotic arm is planned, the robot arm lacks flexibility if the tasks change suddenly. Therefore, an interesting direction will be designing an automatic sorting system with manipulators using DRL, which will improve the intelligence and adaptability of the traditional industrial production progress significantly.

5. Developing an algorithm that allows heterogeneous training. An important assumption in Chapter 5 is that all agents share the same training task and scenario, which may be regarded as a limitation. A potential future working direction could be exploring algorithms which enable heterogeneous training.

Bibliography

[1] R. Munoz-Salinas, "Aruco: a minimal library for augmented reality applications based on opencv," *Universidad de Córdoba*, vol. 386, 2012.

[2] Y. Zhang and M. M. Zavlanos, "Distributed off-policy actor-critic reinforcement learning with policy consensus," in *2019 IEEE 58th Conference on Decision and Control (CDC)*, pp. 4674–4679, IEEE, 2019.

[3] P. Pennesi and I. C. Paschalidis, "A distributed actor-critic algorithm and applications to mobile sensor network coordination problems," *IEEE Transactions on Automatic Control*, vol. 55, no. 2, pp. 492–497, 2010.

[4] J. Bruce and M. M. Veloso, "Real-time randomized path planning for robot navigation," in *Robot soccer world cup*, pp. 288–295, Springer, 2002.

[5] J. Barraquand, B. Langlois, and J.-C. Latombe, "Numerical potential field techniques for robot path planning," *IEEE transactions on systems, man, and cybernetics*, vol. 22, no. 2, pp. 224–241, 1992.

[6] J. Borenstein and Y. Koren, "Real-time obstacle avoidance for fast mobile robots," *IEEE Transactions on systems, Man, and Cybernetics*, vol. 19, no. 5, pp. 1179–1187, 1989.

[7] J. Borenstein, Y. Koren, *et al.*, "Histogramic in-motion mapping for mobile robot obstacle avoidance," *IEEE Transactions on robotics and automation*, vol. 7, no. 4, pp. 535–539, 1991.

[8] J. Mo, Z.-F. Shao, L. Guan, F. Xie, and X. Tang, "Dynamic performance analysis of the x4 high-speed pick-and-place parallel robot," *Robotics and Computer-Integrated Manufacturing*, vol. 46, pp. 48–57, 2017.

[9] F. Nagata, K. Miki, A. Otsuka, K. Yoshida, K. Watanabe, and M. K. Habib, "Pick and place robot using visual feedback control and transfer learning-based cnn," in *2020 IEEE International Conference on Mechatronics and Automation (ICMA)*, pp. 850–855, IEEE, 2020.

[10] F. Belkhouche, B. Belkhouche, and P. Rastgoufard, "Parallel navigation for reaching a moving goal by a mobile robot," *Robotica*, vol. 25, no. 1, pp. 63–74, 2007.

[11] L. Jamone, L. Natale, F. Nori, G. Metta, and G. Sandini, "Autonomous online learning of reaching behavior in a humanoid robot," *International Journal of Humanoid Robotics*, vol. 9, no. 03, p. 1250017, 2012.

[12] C.-H. Chen, T.-K. Liu, and J.-H. Chou, "A novel crowding genetic algorithm and its applications to manufacturing robots," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 3, pp. 1705–1716, 2014.

[13] H. Robinson, B. MacDonald, and E. Broadbent, "The role of healthcare robots for older people at home: A review," *International Journal of Social Robotics*, vol. 6, pp. 575–591, 2014.

[14] F. Amigoni, "Experimental evaluation of some exploration strategies for mobile robots," in *2008 IEEE International Conference on Robotics and Automation*, pp. 2818–2823, IEEE, 2008.

[15] K. M. Wurm, C. Stachniss, and W. Burgard, "Coordinated multi-robot exploration using a segmentation of the environment," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1160–1165, IEEE, 2008.

[16] H. Geering, L. Guzzella, S. Hepner, and C. Onder, "Time-optimal motions of robots in assembly tasks," *IEEE transactions on automatic control*, vol. 31, no. 6, pp. 512–518, 1986.

[17] L. Sciavicco and B. Siciliano, *Modelling and control of robot manipulators*. Springer Science & Business Media, 2001.

[18] M. W. Spong, "On the robust control of robot manipulators," *IEEE Transactions on automatic control*, vol. 37, no. 11, pp. 1782–1786, 1992.

[19] J.-J. E. Slotine and W. Li, "On the adaptive control of robot manipulators," *The international journal of robotics research*, vol. 6, no. 3, pp. 49–59, 1987.

[20] P. J. Alhama Blanco, F. J. Abu-Dakka, and M. Abderrahim, "Practical use of robot manipulators as intelligent manufacturing systems," *Sensors*, vol. 18, no. 9, p. 2877, 2018.

[21] B. Hamner, S. Koterba, J. Shi, R. Simmons, and S. Singh, "An autonomous mobile manipulator for assembly tasks," *Autonomous Robots*, vol. 28, pp. 131–149, 2010.

[22] W. Khalil, M. Gautier, and P. Lemoine, "Identification of the payload inertial parameters of industrial manipulators," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pp. 4943–4948, IEEE, 2007.

[23] S. S. Perumaal and N. Jawahar, "Automated trajectory planner of industrial robot for pick-and-place task," *International Journal of Advanced Robotic Systems*, vol. 10, no. 2, p. 100, 2013.

[24] K. Ghadge, S. More, P. Gaikwad, and S. Chillal, "Robotic arm for pick and place application," *International Journal of Mechanical Engineering and Technology*, vol. 9, no. 1, pp. 125–133, 2018.

[25] A. W. Schell, G. Kewes, T. Schröder, J. Wolters, T. Aichele, and O. Benson, "A scanning probe-based pick-and-place procedure for assembly of integrated quantum optical hybrid devices," *Review of Scientific Instruments*, vol. 82, no. 7, p. 073709, 2011.

[26] P. Dzitac and A. M. Mazid, "A depth sensor to control pick-and-place robots for fruit packaging," in *2012 12th International Conference on Control Automation Robotics & Vision (ICARCV)*, pp. 949–954, IEEE, 2012.

[27] A. Björnsson, M. Jonsson, and K. Johansen, "Automated material handling in composite manufacturing using pick-and-place systems–a review," *Robotics and Computer-Integrated Manufacturing*, vol. 51, pp. 222–229, 2018.

[28] R. Mattone, M. Divona, and A. Wolf, "Sorting of items on a moving conveyor belt. part 2: performance evaluation and optimization of pick-and-place operations," *Robotics and Computer-Integrated Manufacturing*, vol. 16, no. 2-3, pp. 81–90, 2000.

[29] A. Smirnov, A. Kashevnik, N. Teslya, S. Mikhailov, and A. Shabaev, "Smart-m3-based robots self-organization in pick-and-place system," in *2015 17th Conference of Open Innovations Association (FRUCT)*, pp. 210–215, IEEE, 2015.

[30] D. W. Pearson, N. C. Steele, R. F. Albrecht, E. Cervera, and A. P. del Pobil, "Self-organizing maps for supervision in robot pick-and-place operations," in *Artificial Neural Nets and Genetic Algorithms: Proceedings of the International Conference in Alès, France, 1995*, pp. 372–375, Springer, 1995.

[31] R. Kumar, S. Lal, S. Kumar, and P. Chand, "Object detection and recognition for a pick and place robot," in *Asia-Pacific world congress on computer science and engineering*, pp. 1–7, IEEE, 2014.

[32] A. Zeng, S. Song, K.-T. Yu, E. Donlon, F. R. Hogan, M. Bauza, D. Ma, O. Taylor, M. Liu, E. Romo, *et al.*, "Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching," *The International Journal of Robotics Research*, vol. 41, no. 7, pp. 690–705, 2022.

[33] G. Garimella and M. Kobilarov, "Towards model-predictive control for aerial pick-and-place," in *2015 IEEE international conference on robotics and automation (ICRA)*, pp. 4692–4697, IEEE, 2015.

[34] J.-P. Saut, M. Gharbi, J. Cortés, D. Sidobre, and T. Siméon, "Planning pick-and-place tasks with two-hand regrasping," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4528–4533, IEEE, 2010.

[35] P. Bellandi, F. Docchio, and G. Sansoni, "Roboscan: a combined 2d and 3d vision system for improved speed and flexibility in pick-and-place operation," *The International Journal of Advanced Manufacturing Technology*, vol. 69, pp. 1873–1886, 2013.

[36] A. A. Maciejewski and C. A. Klein, "Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments," *The international journal of robotics research*, vol. 4, no. 3, pp. 109–117, 1985.

[37] T. Greville, "The pseudoinverse of a rectangular or singular matrix and its application to the solution of systems of linear equations," *SIAM review*, vol. 1, no. 1, pp. 38–43, 1959.

[38] B. Dasgupta and T. Mruthyunjaya, "Singularity-free path planning for the stewart platform manipulator," *Mechanism and Machine Theory*, vol. 33, no. 6, pp. 711–725, 1998.

[39] S. R. Buss, "Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods," *IEEE Journal of Robotics and Automation*, vol. 17, no. 1-19, p. 16, 2004.

[40] H. Kaneko, T. Arai, K. Inoue, and Y. Mae, "Real-time obstacle avoidance for robot arm using collision jacobian," in *Proceedings 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human and Environment Friendly Robots with High Intelligence and Emotional Quotients (Cat. No. 99CH36289)*, vol. 2, pp. 617–622, IEEE, 1999.

[41] A. Ata and T. Myo, "Collision-free trajectory planning for manipulators using generalized pattern search," *International journal of simulation modelling*, vol. 5, no. 4, pp. 145–154, 2006.

[42] K.-K. Lee, Y. Komoguchi, and M. Buss, "Multiple obstacles avoidance for kinematically redundant manipulators using jacobian transpose method," in *SICE Annual Conference 2007*, pp. 1070–1076, IEEE, 2007.

[43] H.-I. Lin, Y.-Y. Chen, and Y.-Y. Chen, "Robot vision to recognize both object and rotation for robot pick-and-place operation," in *2015 international conference on advanced robotics and intelligent systems (aris)*, pp. 1–6, IEEE, 2015.

[44] T. Kotthauser and G. F. Mauer, "Vision-based autonomous robot control for pick and place operations," in *2009 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pp. 1851–1855, IEEE, 2009.

[45] P.-C. Huang and A. K. Mok, "A case study of cyber-physical system design: Autonomous pick-and-place robot," in *2018 IEEE 24th international conference on embedded and real-time computing systems and applications (RTCSA)*, pp. 22–31, IEEE, 2018.

[46] T. Gecks and D. Henrich, "Human-robot cooperation: Safe pick-and-place operations," in *ROMAN 2005. IEEE International Workshop on Robot and Human Interactive Communication, 2005.*, pp. 549–554, IEEE, 2005.

[47] H. M. Qul'am, T. Dewi, P. Risma, Y. Oktarina, and D. Permatasari, "Edge detection for online image processing of a vision guide pick and place robot," in *2019 International Conference on Electrical Engineering and Computer Science (ICECOS)*, pp. 102–106, IEEE, 2019.

[48] A. Abdulkareem, O. Ladenegan, A. Agbetuyi, and C. Awosope, "Design and implementation of a prototype remote-controlled pick and place robot," *International Journal of Mechanical Engineering and Technology*, vol. 10, no. 2, 2019.

[49] R. V. Sharan and G. C. Onwubolu, "Client-server control architecture for a vision-based pick-and-place robot," *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, vol. 226, no. 8, pp. 1369–1378, 2012.

[50] K. Harish, D. Megha, M. Shuklambari, K. Amit, and K. J. Chaitanya, "Pick and place robotic arm using arduino," *International Journal of Science, Engineering and Technology Research (IJSETR) Volume*, vol. 6, pp. 1568–73, 2017.

[51] S. Smys and G. Ranganathan, "Robot assisted sensing control and manufacture in automobile industry," *Journal of ISMAC*, vol. 1, no. 03, pp. 180–187, 2019.

[52] K. Mølhave, T. Wich, A. Kortschack, and P. Bøggild, "Pick-and-place nanomanipulation using microfabricated grippers," *Nanotechnology*, vol. 17, no. 10, p. 2434, 2006.

[53] A. Cowley, B. Cohen, W. Marshall, C. J. Taylor, and M. Likhachev, "Perception and motion planning for pick-and-place of dynamic objects," in *2013 IEEE/RSJ*

International Conference on Intelligent Robots and Systems, pp. 816–823, IEEE, 2013.

[54] A. Iriondo, E. Lazkano, L. Susperregi, J. Urain, A. Fernandez, and J. Molina, "Pick and place operations in logistics using a mobile manipulator controlled with deep reinforcement learning," *Applied Sciences*, vol. 9, no. 2, p. 348, 2019.

[55] S. D. Han, S. W. Feng, and J. Yu, "Toward fast and optimal robotic pick-and-place on a moving conveyor," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 446–453, 2019.

[56] Z. Zhang, J. Liu, X. Wang, Q. Zhao, C. Zhou, M. Tan, H. Pu, S. Xie, and Y. Sun, "Robotic pick-and-place of multiple embryos for vitrification," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 570–576, 2016.

[57] H. Mnyusiwalla, P. Triantafyllou, P. Sotiropoulos, M. A. Roa, W. Friedl, A. M. Sundaram, D. Russell, and G. Deacon, "A bin-picking benchmark for systematic evaluation of robotic pick-and-place systems," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1389–1396, 2020.

[58] C.-Y. Tsai, C.-C. Wong, C.-J. Yu, C.-C. Liu, and T.-Y. Liu, "A hybrid switched reactive-based visual servo control of 5-dof robot manipulators for pick-and-place tasks," *IEEE Systems Journal*, vol. 9, no. 1, pp. 119–130, 2014.

[59] M. Ghadiri Nejad, S. M. Shavarani, H. Güden, and R. V. Barenji, "Process sequencing for a pick-and-place robot in a real-life flexible robotic cell," *The International Journal of Advanced Manufacturing Technology*, vol. 103, pp. 3613–3627, 2019.

[60] J. T. Schwartz and M. Sharir, "A survey of motion planning and related geometric algorithms," *Artificial Intelligence*, vol. 37, no. 1-3, pp. 157–169, 1988.

[61] A. Bhatia, M. R. Maly, L. E. Kavraki, and M. Y. Vardi, "Motion planning with complex goals," *IEEE Robotics & Automation Magazine*, vol. 18, no. 3, pp. 55–64, 2011.

[62] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "Stomp: Stochastic trajectory optimization for motion planning," in *2011 IEEE international conference on robotics and automation*, pp. 4569–4574, IEEE, 2011.

[63] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Ffrob: An efficient heuristic for task and motion planning," in *Algorithmic Foundations of Robotics XI: Selected Contributions of the Eleventh International Workshop on the Algorithmic Foundations of Robotics*, pp. 179–195, Springer, 2015.

[64] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[65] J.-P. Laumond *et al.*, *Robot motion planning and control*, vol. 229. Springer, 1998.

[66] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel, "Motion planning with sequential convex optimization and convex collision checking," *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1251–1270, 2014.

[67] A. Israr, Z. A. Ali, E. H. Alkhammash, and J. J. Jussila, "Optimization methods applied to motion planning of unmanned aerial vehicles: A review," *Drones*, vol. 6, no. 5, p. 126, 2022.

[68] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2997–3004, IEEE, 2014.

[69] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," in *2009 IEEE international conference on robotics and automation*, pp. 489–494, IEEE, 2009.

[70] R. Bonalli, A. Cauligi, A. Bylard, and M. Pavone, "Gusto: Guaranteed sequential trajectory optimization via sequential convex programming," in *2019 International conference on robotics and automation (ICRA)*, pp. 6741–6747, IEEE, 2019.

[71] S. Banerjee, T. Lew, R. Bonalli, A. Alfaadhel, I. A. Alomar, H. M. Shageer, and M. Pavone, "Learning-based warm-starting for fast sequential convex programming and trajectory optimization," in *2020 IEEE Aerospace Conference*, pp. 1–8, IEEE, 2020.

[72] T. Lew, R. Bonalli, and M. Pavone, "Chance-constrained sequential convex programming for robust trajectory optimization," in *2020 European Control Conference (ECC)*, pp. 1871–1878, IEEE, 2020.

[73] C. Yuan, W. Zhang, G. Liu, X. Pan, and X. Liu, "A heuristic rapidly-exploring random trees method for manipulator motion planning," *IEEE Access*, vol. 8, pp. 900–910, 2019.

[74] K. P. Ferentinos, K. G. Arvanitis, and N. Sigrimis, "Heuristic optimization methods for motion planning of autonomous agricultural vehicles," *Journal of Global Optimization*, vol. 23, no. 2, p. 155, 2002.

[75] J. Wen, X. Zhang, H. Gao, J. Yuan, and Y. Fang, "E 3 mop: Efficient motion planning based on heuristic-guided motion primitives pruning and path optimization with sparse-banded structure," *IEEE Transactions on Automation Science and Engineering*, vol. 19, no. 4, pp. 2762–2775, 2021.

[76] F. Duchoň, A. Babinec, M. Kajan, P. Beňo, M. Florek, T. Fico, and L. Jurišica, "Path planning with modified a star algorithm for a mobile robot," *Procedia engineering*, vol. 96, pp. 59–69, 2014.

[77] A. Stentz *et al.*, "The focussed dˆ* algorithm for real-time replanning," in *IJCAI*, vol. 95, pp. 1652–1659, 1995.

[78] J. Lander and G. CONTENT, "Making kine more flexible," *Game Developer Magazine*, vol. 1, no. 15-22, p. 2, 1998.

[79] R. Mukundan, "A robust inverse kinematics algorithm for animating a joint chain," *International Journal of Computer Applications in Technology*, vol. 34, no. 4, pp. 303–308, 2009.

[80] M. Elbanhawi and M. Simic, "Sampling-based robot motion planning: A review," *Ieee access*, vol. 2, pp. 56–77, 2014.

[81] S. R. Lindemann and S. M. LaValle, "Current issues in sampling-based motion planning," in *Robotics Research. The Eleventh International Symposium: With 303 Figures*, pp. 36–54, Springer, 2005.

[82] Z. Kingston, M. Moll, and L. E. Kavraki, "Sampling-based methods for motion planning with constraints," *Annual review of control, robotics, and autonomous systems*, vol. 1, pp. 159–185, 2018.

[83] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[84] A. Bhatia, L. E. Kavraki, and M. Y. Vardi, "Sampling-based motion planning with temporal goals," in *2010 IEEE International Conference on Robotics and Automation*, pp. 2689–2696, IEEE, 2010.

[85] E. Plaku, K. E. Bekris, B. Y. Chen, A. M. Ladd, and L. E. Kavraki, "Sampling-based roadmap of trees for parallel motion planning," *IEEE Transactions on Robotics*, vol. 21, no. 4, pp. 597–608, 2005.

[86] R. S. Sutton, A. G. Barto, *et al.*, "Reinforcement learning," *Journal of Cognitive Neuroscience*, vol. 11, no. 1, pp. 126–134, 1999.

[87] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[88] O. Arslan and P. Tsiotras, "Use of relaxation methods in sampling-based algorithms for optimal motion planning," in *2013 IEEE International Conference on Robotics and Automation*, pp. 2421–2428, IEEE, 2013.

[89] A.-A. Agha-Mohammadi, S. Chakravorty, and N. M. Amato, "Firm: Sampling-based feedback motion-planning under motion uncertainty and imperfect measurements," *The International Journal of Robotics Research*, vol. 33, no. 2, pp. 268–304, 2014.

[90] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, J. Pineau, *et al.*, "An introduction to deep reinforcement learning," *Foundations and Trends® in Machine Learning*, vol. 11, no. 3-4, pp. 219–354, 2018.

[91] P. Dayan and Y. Niv, "Reinforcement learning: the good, the bad and the ugly," *Current opinion in neurobiology*, vol. 18, no. 2, pp. 185–196, 2008.

[92] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of research and development*, vol. 3, no. 3, pp. 210–229, 1959.

[93] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine learning*, vol. 3, no. 1, pp. 9–44, 1988.

[94] P. Kormushev, S. Calinon, and D. G. Caldwell, "Reinforcement learning in robotics: Applications and real-world challenges," *Robotics*, vol. 2, no. 3, pp. 122–148, 2013.

[95] J. Peters, S. Vijayakumar, and S. Schaal, "Reinforcement learning for humanoid robotics," in *Proceedings of the third IEEE-RAS international conference on humanoid robots*, pp. 1–20, 2003.

[96] A. Jeerige, D. Bein, and A. Verma, "Comparison of deep reinforcement learning approaches for intelligent game playing," in *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 0366–0371, IEEE, 2019.

[97] A. R. Sharma and P. Kaushik, "Literature survey of statistical, deep and reinforcement learning in natural language processing," in *2017 International Conference on Computing, Communication and Automation (ICCCA)*, pp. 350–354, IEEE, 2017.

[98] S. L. Marie-Sainte, N. Alalyani, S. Alotaibi, S. Ghouzali, and I. Abunadi, "Arabic natural language processing and machine learning-based systems," *IEEE Access*, vol. 7, pp. 7011–7020, 2018.

[99] P. N. Kolm and G. Ritter, "Modern perspectives on reinforcement learning in finance," *Modern Perspectives on Reinforcement Learning in Finance (September 6, 2019). The Journal of Machine Learning in Finance*, vol. 1, no. 1, 2020.

[100] A. Charpentier, R. Elie, and C. Remlinger, "Reinforcement learning in economics and finance," *Computational Economics*, pp. 1–38, 2021.

[101] Y. Wang, C. Tang, S. Wang, L. Cheng, R. Wang, M. Tan, and Z. Hou, "Target tracking control of a biomimetic underwater vehicle through deep reinforcement learning," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[102] J. Hu, H. Niu, J. Carrasco, B. Lennox, and F. Arvin, "Voronoi-based multi-robot autonomous exploration in unknown environments via deep reinforcement learning," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 12, pp. 14413–14423, 2020.

[103] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[104] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[105] S. Lange, M. Riedmiller, and A. Voigtländer, "Autonomous reinforcement learning on raw visual input data in a real world application," in *The 2012 international joint conference on neural networks (IJCNN)*, pp. 1–8, IEEE, 2012.

[106] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016.

[107] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[108] P. Tynecki, A. Guziński, J. Kazimierczak, M. Jadczuk, J. Dastych, and A. Onisko, "Phageai-bacteriophage life cycle recognition with machine learning and natural language processing," *BioRxiv*, pp. 2020–07, 2020.

[109] R. Chai, A. Tsourdos, A. Savvaris, S. Chai, Y. Xia, and C. P. Chen, "Six-dof spacecraft optimal trajectory planning and real-time attitude control: a deep neural network-based approach," *IEEE transactions on neural networks and learning systems*, vol. 31, no. 11, pp. 5005–5013, 2019.

[110] X. Yang, Z. Ji, J. Wu, Y.-K. Lai, C. Wei, G. Liu, and R. Setchi, "Hierarchical reinforcement learning with universal policies for multistep robotic manipulation," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[111] Z. Rao, Y. Wu, Z. Yang, W. Zhang, S. Lu, W. Lu, and Z. Zha, "Visual navigation with multiple goals based on deep reinforcement learning," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[112] M. Everett, B. Lütjens, and J. P. How, "Certifiable robustness to adversarial state uncertainty in deep reinforcement learning," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[113] W. Meng, Q. Zheng, Y. Shi, and G. Pan, "An off-policy trust region policy optimization method with monotonic improvement guarantee for deep reinforcement learning," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[114] R. Mao, R. Cui, and C. P. Chen, "Broad learning with reinforcement learning signal feedback: Theory and applications," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[115] R. S. Sutton, "Introduction: The challenge of reinforcement learning," in *Reinforcement Learning*, pp. 1–3, Springer, 1992.

[116] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *Proceedings of the tenth international conference on machine learning*, pp. 330–337, 1993.

[117] W. Wang, X. Chen, H. Fu, and M. Wu, "Model-free distributed consensus control based on actor-critic framework for discrete-time nonlinear multiagent systems," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2018.

[118] A. Alhogail and A. Alsabih, "Applying machine learning and natural language processing to detect phishing email," *Computers & Security*, vol. 110, p. 102414, 2021.

[119] S. S. Mousavi, M. Schukat, and E. Howley, "Deep reinforcement learning: an overview," in *Proceedings of SAI Intelligent Systems Conference*, pp. 426–440, Springer, 2016.

[120] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3133–3174, 2019.

[121] W. Y. Wang, J. Li, and X. He, "Deep reinforcement learning for nlp," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics: Tutorial Abstracts*, pp. 19–21, 2018.

[122] E.-A. Costea *et al.*, "Machine learning-based natural language processing algorithms and electronic health records data," *Linguistic and Philosophical Investigations*, no. 19, pp. 93–99, 2020.

[123] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei, "Deep reinforcement learning from human preferences," *Advances in neural information processing systems*, vol. 30, 2017.

[124] G. Lample and D. S. Chaplot, "Playing fps games with deep reinforcement learning," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[125] S. Song, A. Zeng, J. Lee, and T. Funkhouser, "Grasping in the wild: Learning 6dof closed-loop grasping from low-cost demonstrations," *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 4978–4985, 2020.

[126] A. Zeng, S. Song, S. Welker, J. Lee, A. Rodriguez, and T. Funkhouser, "Learning synergies between pushing and grasping with self-supervised deep reinforcement learning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4238–4245, IEEE, 2018.

[127] A. Zeng, S. Song, J. Lee, A. Rodriguez, and T. Funkhouser, "Tossingbot: Learning to throw arbitrary objects with residual physics," *IEEE Transactions on Robotics*, vol. 36, no. 4, pp. 1307–1319, 2020.

[128] J. Kober, E. Oztop, and J. Peters, "Reinforcement learning to adjust robot movements to new situations," in *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[129] K. Zakka, A. Zeng, J. Lee, and S. Song, "Form2fit: Learning shape priors for generalizable assembly from disassembly," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9404–9410, IEEE, 2020.

[130] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *Advances in neural information processing systems*, pp. 5048–5058, 2017.

[131] J. Matas, S. James, and A. J. Davison, "Sim-to-real reinforcement learning for deformable object manipulation," in *Conference on Robot Learning*, pp. 734–743, PMLR, 2018.

[132] E. Salvato, G. Fenu, E. Medvet, and F. A. Pellegrino, "Crossing the reality gap: A survey on sim-to-real transferability of robot controllers in reinforcement learning," *IEEE Access*, vol. 9, pp. 153171–153187, 2021.

[133] S. James and E. Johns, "3d simulation for robot arm control with deep q-learning," *arXiv preprint arXiv:1609.03759*, 2016.

[134] J. Varley, J. Weisz, J. Weiss, and P. Allen, "Generating multi-fingered robotic grasps via deep learning," in *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 4415–4420, IEEE, 2015.

[135] M. Veres, M. Moussa, and G. W. Taylor, "Modeling grasp motor imagery through deep conditional generative models," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 757–764, 2017.

[136] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *2017 IEEE international conference on robotics and automation (ICRA)*, pp. 3357–3364, IEEE, 2017.

[137] N. Liu, Y. Cai, T. Lu, R. Wang, and S. Wang, "Real–sim–real transfer for real-world robot control policy learning with deep reinforcement learning," *Applied Sciences*, vol. 10, no. 5, p. 1555, 2020.

[138] B. Planche, Z. Wu, K. Ma, S. Sun, S. Kluckner, O. Lehmann, T. Chen, A. Hutter, S. Zakharov, H. Kosch, *et al.*, "Depthsynth: Real-time realistic synthetic data generation from cad models for 2.5 d recognition," in *2017 International Conference on 3D Vision (3DV)*, pp. 1–10, IEEE, 2017.

[139] S. R. Richter, V. Vineet, S. Roth, and V. Koltun, "Playing for data: Ground truth from computer games," in *European conference on computer vision*, pp. 102–118, Springer, 2016.

[140] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox, "Closing the sim-to-real loop: Adapting simulation randomization with real world experience," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 8973–8979, IEEE, 2019.

[141] M. Witman, D. Gidon, D. B. Graves, B. Smit, and A. Mesbah, "Sim-to-real transfer reinforcement learning for control of thermal effects of an atmospheric pressure plasma jet," *Plasma Sources Science and Technology*, vol. 28, no. 9, p. 095019, 2019.

[142] M. Breyer, F. Furrer, T. Novkovic, R. Siegwart, and J. Nieto, "Flexible robotic grasping with sim-to-real transfer based reinforcement learning," *arXiv preprint arXiv:1803.04996*, 2018.

[143] J. Tan, Z. Xie, B. Boots, and C. K. Liu, "Simulation-based design of dynamic controllers for humanoid balancing," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2729–2736, IEEE, 2016.

[144] S. Zhu, A. Kimmel, K. E. Bekris, and A. Boularias, "Fast model identification via physics engines for data-efficient policy search," *arXiv preprint arXiv:1710.08893*, 2017.

[145] J. P. Hanna and P. Stone, "Grounded action transformation for robot learning in simulation," in *Thirty-first AAAI conference on artificial intelligence*, 2017.

[146] A. Farchy, S. Barrett, P. MacAlpine, and P. Stone, "Humanoid robots learning to walk faster: From the real world to simulation and back," in *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pp. 39–46, 2013.

[147] P. M. Scheikl, E. Tagliabue, B. Gyenes, M. Wagner, D. Dall'Alba, P. Fiorini, and F. Mathis-Ullrich, "Sim-to-real transfer for visual reinforcement learning of deformable object manipulation for robot-assisted surgery," *IEEE Robotics and Automation Letters*, vol. 8, no. 2, pp. 560–567, 2022.

[148] Q. Vuong, S. Vikram, H. Su, S. Gao, and H. I. Christensen, "How to pick the domain randomization parameters for sim-to-real transfer of reinforcement learning policies?," *arXiv preprint arXiv:1903.11774*, 2019.

[149] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 3803–3810, IEEE, 2018.

[150] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 23–30, IEEE, 2017.

[151] S. James, A. J. Davison, and E. Johns, "Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task," in *Conference on Robot Learning*, pp. 334–343, PMLR, 2017.

[152] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, *et al.*, "Learning dexterous in-hand manipulation," *The International Journal of Robotics Research*, vol. 39, no. 1, pp. 3–20, 2020.

[153] F. Sadeghi and S. Levine, "Cad2rl: Real single-image flight without a single real image," *arXiv preprint arXiv:1611.04201*, 2016.

[154] S. W. Abeyruwan, L. Graesser, D. B. D'Ambrosio, A. Singh, A. Shankar, A. Bewley, D. Jain, K. M. Choromanski, and P. R. Sanketi, "i-sim2real: Reinforcement learning of robotic policies in tight human-robot interaction loops," in *Conference on Robot Learning*, pp. 212–224, PMLR, 2023.

[155] R. Kaushik, K. Arndt, and V. Kyrki, "Safeapt: Safe simulation-to-real robot learning using diverse policies learned in simulation," *IEEE Robotics and Automation Letters*, 2022.

[156] W. Chen, Y. Xu, Z. Chen, P. Zeng, R. Dang, R. Chen, and J. Xu, "Bidirectional sim-to-real transfer for gelsight tactile sensors with cyclegan," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 6187–6194, 2022.

[157] T. Bi, C. Sferrazza, and R. D'Andrea, "Zero-shot sim-to-real transfer of tactile control policies for aggressive swing-up manipulation," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5761–5768, 2021.

[158] A. Carlson, K. A. Skinner, R. Vasudevan, and M. Johnson-Roberson, "Sensor transfer: Learning optimal sensor effect image augmentation for sim-to-real domain adaptation," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2431–2438, 2019.

[159] U. Viereck, A. Pas, K. Saenko, and R. Platt, "Learning a visuomotor controller for real world robotic grasping using simulated depth images," in *Conference on robot learning*, pp. 291–300, PMLR, 2017.

[160] J. Mahler, J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, J. A. Ojea, and K. Goldberg, "Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics," *arXiv preprint arXiv:1703.09312*, 2017.

[161] S. James, P. Wohlhart, M. Kalakrishnan, D. Kalashnikov, A. Irpan, J. Ibarz, S. Levine, R. Hadsell, and K. Bousmalis, "Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12627–12637, 2019.

[162] K. Kang, S. Belkhale, G. Kahn, P. Abbeel, and S. Levine, "Generalization through simulation: Integrating simulated and real data into deep reinforcement learning for vision-based autonomous flight," in *2019 international conference on robotics and automation (ICRA)*, pp. 6008–6014, IEEE, 2019.

[163] T. Kim, M. Cha, H. Kim, J. K. Lee, and J. Kim, "Learning to discover cross-domain relations with generative adversarial networks," in *International conference on machine learning*, pp. 1857–1865, PMLR, 2017.

[164] E. Tzeng, C. Devin, J. Hoffman, C. Finn, X. Peng, S. Levine, K. Saenko, and T. Darrell, "Towards adapting deep visuomotor representations from simulated to real environments," *CoRR*, vol. abs/1511.07111, 2015.

[165] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige, *et al.*, "Using simulation and domain adaptation to improve efficiency of deep robotic grasping," in *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 4243–4250, IEEE, 2018.

[166] S. Di-Castro Shashua, D. Di Castro, and S. Mannor, "Sim and real: Better together," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[167] B. Sellner, F. W. Heger, L. M. Hiatt, R. Simmons, and S. Singh, "Coordinated multiagent teams and sliding autonomy for large-scale assembly," *Proceedings of the IEEE*, vol. 94, no. 7, pp. 1425–1444, 2006.

[168] J. Barata, L. Camarinha-Matos, and M. Onori, "A multiagent based control approach for evolvable assembly systems," in *INDIN'05. 2005 3rd IEEE International Conference on Industrial Informatics, 2005.*, pp. 478–483, IEEE, 2005.

[169] J. Foerster, I. A. Assael, N. De Freitas, and S. Whiteson, "Learning to communicate with deep multi-agent reinforcement learning," in *Advances in neural information processing systems*, pp. 2137–2145, 2016.

[170] A. Dorri, S. S. Kanhere, and R. Jurdak, "Multi-agent systems: A survey," *Ieee Access*, vol. 6, pp. 28573–28593, 2018.

[171] P. J. Gmytrasiewicz and P. Doshi, "A framework for sequential planning in multi-agent settings," *Journal of Artificial Intelligence Research*, vol. 24, pp. 49–79, 2005.

[172] L. Matignon, G. J. Laurent, and N. Le Fort-Piat, "Independent reinforcement learners in cooperative markov games: a survey regarding coordination problems," *The Knowledge Engineering Review*, vol. 27, no. 1, pp. 1–31, 2012.

[173] F. A. Oliehoek, M. T. Spaan, and N. Vlassis, "Optimal and approximate q-value functions for decentralized pomdps," *Journal of Artificial Intelligence Research*, vol. 32, pp. 289–353, 2008.

[174] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein, "The complexity of decentralized control of markov decision processes," *Mathematics of operations research*, vol. 27, no. 4, pp. 819–840, 2002.

[175] P. Peng, Y. Wen, Y. Yang, Q. Yuan, Z. Tang, H. Long, and J. Wang, "Multi-agent bidirectionally-coordinated nets: Emergence of human-level coordination in learning to play starcraft combat games," *arXiv preprint arXiv:1703.10069*, 2017.

[176] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, 2018.

[177] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," *Advances in neural information processing systems*, vol. 30, 2017.

[178] W. Liu, W. Gu, W. Sheng, X. Meng, Z. Wu, and W. Chen, "Decentralized multi-agent system-based cooperative frequency control for autonomous microgrids with communication constraints," *IEEE Transactions on Sustainable Energy*, vol. 5, no. 2, pp. 446–456, 2014.

[179] Y.-P. Tian and C.-L. Liu, "Consensus of multi-agent systems with diverse input and communication delays," *IEEE Transactions on Automatic Control*, vol. 53, no. 9, pp. 2122–2128, 2008.

[180] K. You and L. Xie, "Network topology and communication data rate for consensusability of discrete-time multi-agent systems," *IEEE Transactions on Automatic Control*, vol. 56, no. 10, pp. 2262–2275, 2011.

[181] Z. Li and J. Chen, "Robust consensus for multi-agent systems communicating over stochastic uncertain networks," *SIAM Journal on Control and Optimization*, vol. 57, no. 5, pp. 3553–3570, 2019.

[182] A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru, J. Aru, and R. Vicente, "Multiagent cooperation and competition with deep reinforcement learning," *PloS one*, vol. 12, no. 4, p. e0172395, 2017.

[183] S. Sukhbaatar, R. Fergus, *et al.*, "Learning multiagent communication with backpropagation," *Advances in neural information processing systems*, vol. 29, 2016.

[184] D. Golovin, D. Sculley, B. McMahan, and M. Young, "Large-scale learning with less ram via randomization," in *International Conference on Machine Learning*, pp. 325–333, 2013.

[185] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1175–1191, 2017.

[186] W. Zhu, P. Kairouz, B. McMahan, H. Sun, and W. Li, "Federated heavy hitters discovery with differential privacy," in *International Conference on Artificial Intelligence and Statistics*, pp. 3837–3847, 2020.

[187] J. Ferber, O. Gutknecht, and F. Michel, "From agents to organizations: an organizational view of multi-agent systems," in *International workshop on agent-oriented software engineering*, pp. 214–230, Springer, 2003.

[188] Y. Zou, K. Xia, Z. Zuo, and Z. Ding, "Distributed interval consensus of multi-agent systems with pulse width modulation protocol," *IEEE Transactions on Automatic Control*, 2022.

[189] J. Larson, K.-Y. Liang, and K. H. Johansson, "A distributed framework for coordinated heavy-duty vehicle platooning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 1, pp. 419–429, 2014.

[190] P. Liu, A. Kurt, and U. Ozguner, "Distributed model predictive control for cooperative and flexible vehicle platooning," *IEEE Transactions on Control Systems Technology*, vol. 27, no. 3, pp. 1115–1128, 2018.

[191] G. T. Papadopoulos, M. Antona, and C. Stephanidis, "Towards open and expandable cognitive ai architectures for large-scale multi-agent human-robot collaborative learning," *IEEE Access*, vol. 9, pp. 73890–73909, 2021.

[192] K. Bakliwal, M. H. Dhada, A. S. Palau, A. K. Parlikad, and B. K. Lad, "A multi agent system architecture to implement collaborative learning for social industrial assets," *IFAC-PapersOnLine*, vol. 51, no. 11, pp. 1237–1242, 2018.

[193] L. Ma, Z. Wang, Q.-L. Han, and Y. Liu, "Consensus control of stochastic multi-agent systems: a survey," *Science China Information Sciences*, vol. 60, no. 12, pp. 1–15, 2017.

[194] J. Liu, Y. Yu, Q. Wang, and C. Sun, "Fixed-time event-triggered consensus control for multi-agent systems with nonlinear uncertainties," *Neurocomputing*, vol. 260, pp. 497–504, 2017.

[195] C. Ma, T. Li, and J. Zhang, "Consensus control for leader-following multi-agent systems with measurement noises," *Journal of Systems Science and Complexity*, vol. 23, no. 1, pp. 35–49, 2010.

[196] J. Khazaei and Z. Miao, "Consensus control for energy storage systems," *IEEE Transactions on Smart Grid*, vol. 9, no. 4, pp. 3009–3017, 2016.

[197] W. Qiao and R. Sipahi, "Consensus control under communication delay in a three-robot system: Design and experiments," *IEEE transactions on control systems technology*, vol. 24, no. 2, pp. 687–694, 2015.

[198] S. Motsch and E. Tadmor, "Heterophilious dynamics enhances consensus," *SIAM review*, vol. 56, no. 4, pp. 577–621, 2014.

[199] L. Conradt and T. J. Roper, "Consensus decision making in animals," *Trends in ecology & evolution*, vol. 20, no. 8, pp. 449–456, 2005.

[200] H. Peng, J. Wang, S. Wang, W. Shen, D. Shi, and D. Liu, "Coordinated motion control for a wheel-leg robot with speed consensus strategy," *IEEE/ASME Transactions on Mechatronics*, vol. 25, no. 3, pp. 1366–1376, 2020.

[201] X.-Y. Yao, J. H. Park, H.-F. Ding, and M.-F. Ge, "Event-triggered consensus control for networked underactuated robotic systems," *IEEE Transactions on Cybernetics*, vol. 52, no. 5, pp. 2896–2906, 2020.

[202] M. A. Joordens and M. Jamshidi, "Consensus control for a system of underwater swarm robots," *IEEE Systems Journal*, vol. 4, no. 1, pp. 65–73, 2010.

[203] J. Xi, N. Cai, and Y. Zhong, "Consensus problems for high-order linear time-invariant swarm systems," *Physica A: Statistical Mechanics and its Applications*, vol. 389, no. 24, pp. 5619–5627, 2010.

[204] A. V. Proskurnikov, A. S. Matveev, and M. Cao, "Opinion dynamics in social networks with hostile camps: Consensus vs. polarization," *IEEE Transactions on Automatic Control*, vol. 61, no. 6, pp. 1524–1536, 2015.

[205] L. Li, A. Scaglione, A. Swami, and Q. Zhao, "Consensus, polarization and clustering of opinions in social networks," *IEEE Journal on Selected Areas in Communications*, vol. 31, no. 6, pp. 1072–1083, 2013.

[206] K. Utkarsh, A. Trivedi, D. Srinivasan, and T. Reindl, "A consensus-based distributed computational intelligence technique for real-time optimal control in smart distribution grids," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 1, no. 1, pp. 51–60, 2016.

[207] H. A. Pham, T. Soriano, V. H. Ngo, and V. Gies, "Distributed adaptive neural network control applied to a formation tracking of a group of low-cost underwater drones in hazardous environments," *Applied Sciences*, vol. 10, no. 5, p. 1732, 2020.

[208] N. A. K. Zghair and A. S. Al-Araji, "A one decade survey of autonomous mobile robot systems," *International Journal of Electrical and Computer Engineering*, vol. 11, no. 6, p. 4891, 2021.

[209] J. Hu and A. Lanzon, "An innovative tri-rotor drone and associated distributed aerial drone swarm control," *Robotics and Autonomous Systems*, vol. 103, pp. 162–174, 2018.

[210] F. Ji, J. Wu, F. Chiclana, S. Wang, H. Fujita, and E. Herrera-Viedma, "The overlapping community driven feedback mechanism to support consensus in social network group decision making," *IEEE Transactions on Fuzzy Systems*, 2023.

[211] Z. Chen and E. G. Larsson, "Consensus-based distributed computation of link-based network metrics," *IEEE Signal Processing Letters*, vol. 28, pp. 249–253, 2021.

[212] Z. Li, Z. Duan, and F. L. Lewis, "Distributed robust consensus control of multi-agent systems with heterogeneous matching uncertainties," *Automatica*, vol. 50, no. 3, pp. 883–889, 2014.

[213] G. Xie and L. Wang, "Consensus control for a class of networks of dynamic agents," *International Journal of Robust and Nonlinear Control: IFAC-Affiliated Journal*, vol. 17, no. 10-11, pp. 941–959, 2007.

[214] A.-Y. Lu and G.-H. Yang, "Distributed consensus control for multi-agent systems under denial-of-service," *Information Sciences*, vol. 439, pp. 95–107, 2018.

[215] J. Qin and C. Yu, "Cluster consensus control of generic linear multi-agent systems under directed topology with acyclic partition," *Automatica*, vol. 49, no. 9, pp. 2898–2905, 2013.

[216] X.-M. Li, Q. Zhou, P. Li, H. Li, and R. Lu, "Event-triggered consensus control for multi-agent systems against false data-injection attacks," *IEEE transactions on cybernetics*, vol. 50, no. 5, pp. 1856–1866, 2019.

[217] J. Huang, C. Wen, W. Wang, and Y.-D. Song, "Adaptive finite-time consensus control of a group of uncertain nonlinear mechanical systems," *Automatica*, vol. 51, pp. 292–301, 2015.

[218] Q. Hui and W. M. Haddad, "Distributed nonlinear control algorithms for network consensus," *Automatica*, vol. 44, no. 9, pp. 2375–2381, 2008.

[219] J. Liu, Y. Yu, H. He, and C. Sun, "Team-triggered practical fixed-time consensus of double-integrator agents with uncertain disturbance," *IEEE Transactions on Cybernetics*, vol. 51, no. 6, pp. 3263–3272, 2020.

[220] H. Du, S. Li, and P. Shi, "Robust consensus algorithm for second-order multi-agent systems with external disturbances," *International Journal of Control*, vol. 85, no. 12, pp. 1913–1928, 2012.

[221] D. Zhang, J. Jiang, W. Zhang, *et al.*, "Robust and scalable management of power networks in dual-source trolleybus systems: A consensus control framework," *IEEE transactions on intelligent transportation systems*, vol. 17, no. 4, pp. 1029–1038, 2015.

[222] J. Seo, D. Ko, S. Kim, and S. Park, "A coordination technique for improving scalability of byzantine fault-tolerant consensus," *Applied Sciences*, vol. 10, no. 21, p. 7609, 2020.

[223] T. R. Krogstad and J. T. Gravdahl, "6-dof mutual synchronization of formation flying spacecraft," in *Proceedings of the 45th IEEE Conference on Decision and Control*, pp. 5706–5711, IEEE, 2006.

[224] W. Liu, H. Niu, I. Jang, G. Herrmann, and J. Carrasco, "Distributed neural networks training for robotic manipulation with consensus algorithm," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–15, 2022.

[225] E. Nuno, A. Loria, T. Hernández, M. Maghenem, and E. Panteley, "Distributed consensus-formation of force-controlled nonholonomic robots with time-varying delays," *Automatica*, vol. 120, p. 109114, 2020.

[226] W. Wang, J. Huang, C. Wen, and H. Fan, "Distributed adaptive control for consensus tracking with application to formation control of nonholonomic mobile robots," *Automatica*, vol. 50, no. 4, pp. 1254–1263, 2014.

[227] C. P. Chen, G.-X. Wen, Y.-J. Liu, and F.-Y. Wang, "Adaptive consensus control for a class of nonlinear multiagent time-delay systems using neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 6, pp. 1217–1226, 2014.

[228] D. Zhao, W. Ni, and Q. Zhu, "A framework of neural networks based consensus control for multiple robotic manipulators," *Neurocomputing*, vol. 140, pp. 8–18, 2014.

[229] Y.-J. Liu, W. Zhao, L. Liu, D. Li, S. Tong, and C. P. Chen, "Adaptive neural network control for a class of nonlinear systems with function constraints on states," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[230] L. Liu, Y.-J. Liu, A. Chen, S. Tong, and C. Chen, "Integral barrier lyapunov function-based adaptive control for switched nonlinear systems," *Science China Information Sciences*, vol. 63, no. 3, pp. 1–14, 2020.

[231] K. Wu, J. Hu, B. Lennox, and F. Arvin, "Sdp-based robust formation-containment coordination of swarm robotic systems with input saturation," *Journal of Intelligent & Robotic Systems*, vol. 102, no. 1, pp. 1–16, 2021.

[232] A. Zhang, D. Zhou, P. Yang, and M. Yang, "Event-triggered finite-time consensus with fully continuous communication free for second-order multi-agent systems," *International Journal of Control, Automation and Systems*, vol. 17, no. 4, pp. 836–846, 2019.

[233] K. Wu, J. Hu, B. Lennox, and F. Arvin, "Finite-time bearing-only formation tracking of heterogeneous mobile robots with collision avoidance," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2021.

[234] W. Ren, "Multi-vehicle consensus with a time-varying reference state," *Systems & Control Letters*, vol. 56, no. 7-8, pp. 474–483, 2007.

[235] F. Yu, H. Shang, Q. Zhu, H. Zhang, and Y. Chen, "An efficient rrt-based motion planning algorithm for autonomous underwater vehicles under cylindrical sampling constraints," *Autonomous Robots*, vol. 47, no. 3, pp. 281–297, 2023.

[236] Y. Li, X. Hao, Y. She, S. Li, and M. Yu, "Constrained motion planning of free-float dual-arm space manipulator via deep reinforcement learning," *Aerospace Science and Technology*, vol. 109, p. 106446, 2021.

[237] M. Schwarzer, N. Rajkumar, M. Noukhovitch, A. Anand, L. Charlin, R. D. Hjelm, P. Bachman, and A. C. Courville, "Pretraining representations for data-efficient reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 12686–12699, 2021.

[238] Z. Xu and J. H. Saleh, "Machine learning for reliability engineering and safety applications: Review of current status and future opportunities," *Reliability Engineering & System Safety*, vol. 211, p. 107530, 2021.

[239] S. Bussmann, N. Jennings, M. J. Wooldridge, and M. J. Wooldridge, *Multiagent systems for manufacturing control: a design methodology*. Springer, 2004.

[240] Y. Zhang, H. Zhu, D. Tang, T. Zhou, and Y. Gui, "Dynamic job shop scheduling based on deep reinforcement learning for multi-agent manufacturing systems," *Robotics and Computer-Integrated Manufacturing*, vol. 78, p. 102412, 2022.

[241] J. Axelsson, "Safety in vehicle platooning: A systematic literature review," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 5, pp. 1033–1045, 2016.

[242] M. Kamali, L. A. Dennis, O. McAree, M. Fisher, and S. M. Veres, "Formal verification of autonomous vehicle platooning," *Science of computer programming*, vol. 148, pp. 88–106, 2017.

[243] W. Liu, H. Niu, M. N. Mahyuddin, G. Herrmann, and J. Carrasco, "A model-free deep reinforcement learning approach for robotic manipulators path planning," in *2021 21st International Conference on Control, Automation and Systems (ICCAS)*, pp. 512–517, IEEE, 2021.

[244] K. Wei and B. Ren, "A method on dynamic path planning for robotic manipulator autonomous obstacle avoidance based on an improved rrt algorithm," *Sensors*, vol. 18, no. 2, p. 571, 2018.

[245] H. Niu, Z. Ji, F. Arvin, B. Lennox, H. Yin, and J. Carrasco, "Accelerated sim-to-real deep reinforcement learning: Learning collision avoidance from human player," in *2021 IEEE/SICE International Symposium on System Integration (SII)*, pp. 144–149, IEEE, 2021.

[246] H. Niu, Z. Ji, Z. Zhu, H. Yin, and J. Carrasco, "3d vision-guided pick-and-place using kuka lbr iiwa robot," in *2021 IEEE/SICE International Symposium on System Integration (SII)*, pp. 592–593, IEEE, 2021.

[247] N. T. Dantam, "Robust and efficient forward, differential, and inverse kinematics using dual quaternions," *The International Journal of Robotics Research*, vol. 40, no. 10-11, pp. 1087–1105, 2021.

[248] H. Ye, D. Wang, J. Wu, Y. Yue, and Y. Zhou, "Forward and inverse kinematics of a 5-dof hybrid robot for composite material machining," *Robotics and Computer-Integrated Manufacturing*, vol. 65, p. 101961, 2020.

[249] M. Alebooyeh and R. J. Urbanic, "Neural network model for identifying workspace, forward and inverse kinematics of the 7-dof yumi 14000 abb collaborative robot," *IFAC-PapersOnLine*, vol. 52, no. 10, pp. 176–181, 2019.

[250] F. Xiao, G. Li, D. Jiang, Y. Xie, J. Yun, Y. Liu, L. Huang, and Z. Fang, "An effective and unified method to derive the inverse kinematics formulas of general six-dof manipulator with simple geometry," *Mechanism and Machine Theory*, vol. 159, p. 104265, 2021.

[251] L. Sciavicco and B. Siciliano, *Modelling and control of robot manipulators*. Springer Science & Business Media, 2012.

[252] R. Singh, V. Kukshal, and V. S. Yadav, "A review on forward and inverse kinematics of classical serial manipulators," *Advances in Engineering Design: Select Proceedings of ICOIED 2020*, pp. 417–428, 2021.

[253] P. James, A. Prakash, V. Kalburgi, and P. Sreedharan, "Design, analysis, manufacturing of four-legged walking robot with insect type leg," *Materials Today: Proceedings*, vol. 46, pp. 4647–4652, 2021.

[254] F. rui Zhang, J.-f. Han, and P. Ruan, "Beam pointing analysis and a novel coarse pointing assembly design in space laser communication," *Optik*, vol. 189, pp. 130–147, 2019.

[255] C. Klug, D. Schmalstieg, T. Gloor, and C. Arth, "A complete workflow for automatic forward kinematics model extraction of robotic total stations using the denavit-hartenberg convention," *Journal of Intelligent & Robotic Systems*, vol. 95, pp. 311–329, 2019.

[256] G. Li, F. Zhang, Y. Fu, and S. Wang, "Kinematic calibration of serial robot using dual quaternions," *Industrial Robot: the international journal of robotics research and application*, vol. 46, no. 2, pp. 247–258, 2019.

[257] J. Gallardo-Alvarado, M. A. Garcia-Murillo, L. A. Alcaraz-Caracheo, F. J. Torres, and X. Y. Sandoval-Castro, "Forward kinematics and singularity analyses of an uncoupled parallel manipulator by algebraic screw theory," *IEEE Access*, vol. 10, pp. 4513–4522, 2021.

[258] M. Cardona and C. G. Cena, "Direct kinematics and jacobian analysis of exoskeleton robots using screw theory and simscape multibody™," in *2019 IEEE 39th Central America and Panama Convention (CONCAPAN XXXIX)*, pp. 1–6, IEEE, 2019.

[259] D. E. Whitney, "Resolved motion rate control of manipulators and human prostheses," *IEEE Transactions on man-machine systems*, vol. 10, no. 2, pp. 47–53, 1969.

[260] C. W. Wampler, "Manipulator inverse kinematic solutions based on vector formulations and damped least-squares methods," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 16, no. 1, pp. 93–101, 1986.

[261] Y. Nakamura and H. Hanafusa, "Inverse kinematic solutions with singularity robustness for robot manipulator control," 1986.

[262] A. A. Maciejewski, "Dealing with the ill-conditioned equations of motion for articulated figures," *IEEE Computer Graphics and Applications*, vol. 10, no. 3, pp. 63–71, 1990.

[263] A. R. Almusawi, L. C. Dülger, and S. Kapucu, "A new artificial neural network approach in solving inverse kinematics of robotic arm (denso vp6242)," *Computational intelligence and neuroscience*, vol. 2016, 2016.

[264] R. KöKer, "A genetic algorithm approach to a neural-network-based inverse kinematics solution of robotic manipulators based on error minimization," *Information Sciences*, vol. 222, pp. 528–543, 2013.

[265] C. Hasberg, S. Hensel, and C. Stiller, "Simultaneous localization and mapping for path-constrained motion," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 2, pp. 541–552, 2011.

[266] A. T. Hasan, N. Ismail, A. M. S. Hamouda, I. Aris, M. H. Marhaban, and H. Al-Assadi, "Artificial neural network-based kinematics jacobian solution for serial manipulator passing through singular configurations," *Advances in Engineering Software*, vol. 41, no. 2, pp. 359–367, 2010.

[267] Y. Yang, G. Peng, Y. Wang, and H. Zhang, "A new solution for inverse kinematics of 7-dof manipulator based on genetic algorithm," in *2007 IEEE international conference on automation and logistics*, pp. 1947–1951, IEEE, 2007.

[268] M. Klingensmith, S. S. Sirinivasa, and M. Kaess, "Articulated robot motion for simultaneous localization and mapping (arm-slam)," *IEEE robotics and automation letters*, vol. 1, no. 2, pp. 1156–1163, 2016.

[269] C. Liu, C.-Y. Lin, and M. Tomizuka, "The convex feasible set algorithm for real time optimization in motion planning," *SIAM Journal on Control and optimization*, vol. 56, no. 4, pp. 2712–2733, 2018.

[270] J. L. Vázquez, M. Brühlmeier, A. Liniger, A. Rupenyan, and J. Lygeros, "Optimization-based hierarchical motion planning for autonomous racing," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2397–2403, IEEE, 2020.

[271] J. Ichnowski, Y. Avigal, V. Satish, and K. Goldberg, "Deep learning can accelerate grasp-optimized motion planning," *Science Robotics*, vol. 5, no. 48, p. eabd7710, 2020.

[272] M. N. Ab Wahab, S. Nefti-Meziani, and A. Atyabi, "A comparative review on mobile robot path planning: Classical or meta-heuristic methods?," *Annual Reviews in Control*, vol. 50, pp. 233–252, 2020.

[273] H. Wu, J. Yu, J. Pan, and X. Pei, "A novel obstacle avoidance heuristic algorithm of continuum robot based on fabrik," *Science China Technological Sciences*, pp. 1–15, 2022.

[274] Z. Kingston, M. Moll, and L. E. Kavraki, "Exploring implicit spaces for constrained sampling-based planning," *The International Journal of Robotics Research*, vol. 38, no. 10-11, pp. 1151–1178, 2019.

[275] A. M. Wells, N. T. Dantam, A. Shrivastava, and L. E. Kavraki, "Learning feasibility for task and motion planning in tabletop environments," *IEEE robotics and automation letters*, vol. 4, no. 2, pp. 1255–1262, 2019.

[276] P. Beeson and B. Ames, "Trac-ik: An open-source library for improved solving of generic inverse kinematics," in *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pp. 928–935, IEEE, 2015.

[277] Y. Wu, "A survey on population-based meta-heuristic algorithms for motion planning of aircraft," *Swarm and Evolutionary Computation*, vol. 62, p. 100844, 2021.

[278] B. Ichter and M. Pavone, "Robot motion planning in learned latent spaces," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2407–2414, 2019.

[279] J. Pan and D. Manocha, "Efficient configuration space construction and optimization for motion planning," *Engineering*, vol. 1, no. 1, pp. 046–057, 2015.

[280] S. Liu and P. Liu, "Benchmarking and optimization of robot motion planning with motion planning pipeline," *The International Journal of Advanced Manufacturing Technology*, pp. 1–13, 2021.

[281] D. Rakita, B. Mutlu, and M. Gleicher, "Relaxedik: Real-time synthesis of accurate and feasible robot arm motion.," in *Robotics: Science and Systems*, pp. 26–30, Pittsburgh, PA, 2018.

[282] J. Mainprice, N. Ratliff, M. Toussaint, and S. Schaal, "An interior point method solving motion planning problems with narrow passages," in *2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, pp. 547–552, IEEE, 2020.

[283] H. J. Kim, Q. Wang, S. Rahmatalla, C. C. Swan, J. S. Arora, K. Abdel-Malek, and J. G. Assouline, "Dynamic motion planning of 3d human locomotion using gradient-based optimization," *Journal of biomechanical engineering*, vol. 130, no. 3, 2008.

[284] A. Akbari, F. Lagriffoul, and J. Rosell, "Combined heuristic task and motion planning for bi-manual robots," *Autonomous robots*, vol. 43, no. 6, pp. 1575–1590, 2019.

[285] R. Terasawa, Y. Ariki, T. Narihira, T. Tsuboi, and K. Nagasaka, "3d-cnn based heuristic guided task-space planner for faster motion planning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9548–9554, IEEE, 2020.

[286] A. Aristidou and J. Lasenby, "Fabrik: A fast, iterative solver for the inverse kinematics problem," *Graphical Models*, vol. 73, no. 5, pp. 243–260, 2011.

[287] K. Jang, J. Baek, S. Park, and J. Park, "Motion planning for closed-chain constraints based on probabilistic roadmap with improved connectivity," *IEEE/ASME Transactions on Mechatronics*, vol. 27, no. 4, pp. 2035–2043, 2022.

[288] R. Kumar, A. Mandalika, S. Choudhury, and S. Srinivasa, "Lego: Leveraging experience in roadmap generation for sampling-based planning," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1488–1495, IEEE, 2019.

[289] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, vol. 2, pp. 995–1001, IEEE, 2000.

[290] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the rrt," in *2011 IEEE International Conference on Robotics and Automation*, pp. 1478–1483, IEEE, 2011.

[291] H.-T. L. Chiang, J. Hsu, M. Fiser, L. Tapia, and A. Faust, "Rl-rrt: Kinodynamic motion planning via learning reachability estimators from rl policies," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4298–4305, 2019.

[292] T. Rybus, "Point-to-point motion planning of a free-floating space manipulator using the rapidly-exploring random trees (rrt) method," *Robotica*, vol. 38, no. 6, pp. 957–982, 2020.

[293] Y. Kuwata, G. A. Fiore, J. Teo, E. Frazzoli, and J. P. How, "Motion planning for urban driving using rrt," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1681–1686, IEEE, 2008.

[294] O. Salzman and D. Halperin, "Asymptotically near-optimal rrt for fast, high-quality motion planning," *IEEE Transactions on Robotics*, vol. 32, no. 3, pp. 473–483, 2016.

[295] I. Noreen, A. Khan, and Z. Habib, "Optimal path planning using rrt* based approaches: a survey and future directions," *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 11, 2016.

[296] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[297] P. Cunningham, M. Cord, and S. J. Delany, "Supervised learning," *Machine learning techniques for multimedia: case studies on organization and retrieval*, pp. 21–49, 2008.

[298] H. B. Barlow, "Unsupervised learning," *Neural computation*, vol. 1, no. 3, pp. 295–311, 1989.

[299] W. Qiang and Z. Zhongli, "Reinforcement learning model, algorithms and its application," in *2011 International Conference on Mechatronic Science, Electric Engineering and Computer (MEC)*, pp. 1143–1146, IEEE, 2011.

[300] Z. Zhang, D. Zhang, and R. C. Qiu, "Deep reinforcement learning for power system applications: An overview," *CSEE Journal of Power and Energy Systems*, vol. 6, no. 1, pp. 213–225, 2019.

[301] J. Garcıa and F. Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437–1480, 2015.

[302] L. Buşoniu, R. Babuška, and B. D. Schutter, "Multi-agent reinforcement learning: An overview," *Innovations in multi-agent systems and applications-1*, pp. 183–221, 2010.

[303] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[304] H. Zhang, H. Jiang, Y. Luo, and G. Xiao, "Data-driven optimal consensus control for discrete-time multi-agent systems with unknown dynamics using reinforcement learning method," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 5, pp. 4091–4100, 2016.

[305] H. Kandath, J. Senthilnath, and S. Sundaram, "Mutli-agent consensus under communication failure using actor-critic reinforcement learning," in *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1461–1465, IEEE, 2018.

[306] X. Wang, S. Wang, X. Liang, D. Zhao, J. Huang, X. Xu, B. Dai, and Q. Miao, "Deep reinforcement learning: a survey," *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[307] J. Morimoto and K. Doya, "Robust reinforcement learning," *Neural computation*, vol. 17, no. 2, pp. 335–359, 2005.

[308] S. Sinha, A. Mandlekar, and A. Garg, "S4rl: Surprisingly simple self-supervision for offline reinforcement learning in robotics," in *Conference on Robot Learning*, pp. 907–917, PMLR, 2022.

[309] A. A. Apolinarska, M. Pacher, H. Li, N. Cote, R. Pastrana, F. Gramazio, and M. Kohler, "Robotic assembly of timber joints using reinforcement learning," *Automation in Construction*, vol. 125, p. 103569, 2021.

[310] B. Singh, R. Kumar, and V. P. Singh, "Reinforcement learning in robotic applications: a comprehensive survey," *Artificial Intelligence Review*, pp. 1–46, 2022.

[311] H. Bae, G. Kim, J. Kim, D. Qian, and S. Lee, "Multi-robot path planning method using reinforcement learning," *Applied sciences*, vol. 9, no. 15, p. 3057, 2019.

[312] H. Oliff, Y. Liu, M. Kumar, M. Williams, and M. Ryan, "Reinforcement learning for facilitating human-robot-interaction in manufacturing," *Journal of Manufacturing Systems*, vol. 56, pp. 326–340, 2020.

[313] P. Chen and W. Lu, "Deep reinforcement learning based moving object grasping," *Information Sciences*, vol. 565, pp. 62–76, 2021.

[314] T. Boroushaki, J. Leng, I. Clester, A. Rodriguez, and F. Adib, "Robotic grasping of fully-occluded objects using rf perception," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 923–929, IEEE, 2021.

[315] H. Hu, K. Zhang, A. H. Tan, M. Ruan, C. Agia, and G. Nejat, "A sim-to-real pipeline for deep reinforcement learning for autonomous robot navigation in cluttered rough terrain," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 6569–6576, 2021.

[316] A. Malik, Y. Lischuk, T. Henderson, and R. Prazenica, "A deep reinforcement-learning approach for inverse kinematics solution of a high degree of freedom robotic manipulator," *Robotics*, vol. 11, no. 2, p. 44, 2022.

[317] B. Beyret, A. Shafti, and A. A. Faisal, "Dot-to-dot: Explainable hierarchical reinforcement learning for robotic manipulation," in *2019 IEEE/RSJ International Conference on intelligent robots and systems (IROS)*, pp. 5014–5019, IEEE, 2019.

[318] B. Recht, "A tour of reinforcement learning: The view from continuous control," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 2, pp. 253–279, 2019.

[319] G. Ning, X. Zhang, and H. Liao, "Autonomic robotic ultrasound imaging system based on reinforcement learning," *IEEE Transactions on Biomedical Engineering*, vol. 68, no. 9, pp. 2787–2797, 2021.

[320] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. Al Sallab, S. Yogamani, and P. Pérez, "Deep reinforcement learning for autonomous driving: A survey," *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[321] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[322] S. Dong, P. Wang, and K. Abbas, "A survey on deep learning and its applications," *Computer Science Review*, vol. 40, p. 100379, 2021.

[323] M. A. Nielsen, *Neural networks and deep learning*, vol. 25. Determination press San Francisco, CA, USA, 2015.

[324] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis, "Deep learning for computer vision: A brief review," *Computational intelligence and neuroscience*, vol. 2018, 2018.

[325] J. Chai, H. Zeng, A. Li, and E. W. Ngai, "Deep learning in computer vision: A critical review of emerging techniques and application scenarios," *Machine Learning with Applications*, vol. 6, p. 100134, 2021.

[326] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," *ieee Computational intelligenCe magazine*, vol. 13, no. 3, pp. 55–75, 2018.

[327] D. W. Otter, J. R. Medina, and J. K. Kalita, "A survey of the usages of deep learning for natural language processing," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 2, pp. 604–624, 2020.

[328] K. Cobbe, O. Klimov, C. Hesse, T. Kim, and J. Schulman, "Quantifying generalization in reinforcement learning," in *International Conference on Machine Learning*, pp. 1282–1289, PMLR, 2019.

[329] A. Haydari and Y. Yilmaz, "Deep reinforcement learning for intelligent transportation systems: A survey," *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[330] C. Janiesch, P. Zschech, and K. Heinrich, "Machine learning and deep learning," *Electronic Markets*, vol. 31, no. 3, pp. 685–695, 2021.

[331] M. Hüttenrauch, S. Adrian, G. Neumann, *et al.*, "Deep reinforcement learning for swarm systems," *Journal of Machine Learning Research*, vol. 20, no. 54, pp. 1–31, 2019.

[332] T. G. Thuruthel, E. Falotico, F. Renda, and C. Laschi, "Model-based reinforcement learning for closed-loop dynamic control of soft robotic manipulators," *IEEE Transactions on Robotics*, vol. 35, no. 1, pp. 124–134, 2018.

[333] J.-J. Park, J.-H. Kim, and J.-B. Song, "Path planning for a robot manipulator based on probabilistic roadmap and reinforcement learning," *International Journal of Control, Automation, and Systems*, vol. 5, no. 6, pp. 674–680, 2007.

[334] Y. Yang, H. Liang, and C. Choi, "A deep learning approach to grasping the invisible," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2232–2239, 2020.

[335] T. Inoue, G. De Magistris, A. Munawar, T. Yokoya, and R. Tachibana, "Deep reinforcement learning for high precision assembly tasks," in *2017 IEEE/RSJ*

International Conference on Intelligent Robots and Systems (IROS), pp. 819–825, IEEE, 2017.

[336] S. K. Pradhan and B. Subudhi, "Real-time adaptive control of a flexible manipulator using reinforcement learning," *IEEE Transactions on Automation Science and Engineering*, vol. 9, no. 2, pp. 237–249, 2012.

[337] D. Zhao, H. Wang, K. Shao, and Y. Zhu, "Deep reinforcement learning with experience replay based on sarsa," in *2016 IEEE symposium series on computational intelligence (SSCI)*, pp. 1–6, IEEE, 2016.

[338] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE transactions on systems, man, and cybernetics*, no. 5, pp. 834–846, 1983.

[339] T. Szandała, "Review and comparison of commonly used activation functions for deep neural networks," *Bio-inspired neurocomputing*, pp. 203–224, 2021.

[340] A. Apicella, F. Donnarumma, F. Isgrò, and R. Prevete, "A survey on modern trainable activation functions," *Neural Networks*, vol. 138, pp. 14–32, 2021.

[341] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, and H. Arshad, "State-of-the-art in artificial neural network applications: A survey," *Heliyon*, vol. 4, no. 11, p. e00938, 2018.

[342] Y. Ito, "Representation of functions by superpositions of a step or sigmoid function and their applications to neural network theory," *Neural Networks*, vol. 4, no. 3, pp. 385–394, 1991.

[343] A. Jacot, F. Gabriel, and C. Hongler, "Neural tangent kernel: Convergence and generalization in neural networks," *Advances in neural information processing systems*, vol. 31, 2018.

[344] Y. Li and Y. Yuan, "Convergence analysis of two-layer neural networks with relu activation," *Advances in neural information processing systems*, vol. 30, 2017.

[345] S. Fujimoto, D. Meger, and D. Precup, "Off-policy deep reinforcement learning without exploration," in *International conference on machine learning*, pp. 2052–2062, PMLR, 2019.

[346] J. Clifton and E. Laber, "Q-learning: Theory and applications," *Annual Review of Statistics and Its Application*, vol. 7, pp. 279–301, 2020.

[347] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[348] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, pp. 1057–1063, 2000.

[349] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[350] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: a survey," in *2020 IEEE symposium series on computational intelligence (SSCI)*, pp. 737–744, IEEE, 2020.

[351] V. Uc-Cetina, N. Navarro-Guerrero, A. Martin-Gonzalez, C. Weber, and S. Wermter, "Survey on reinforcement learning for language processing," *Artificial Intelligence Review*, vol. 56, no. 2, pp. 1543–1575, 2023.

[352] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*, pp. 1861–1870, PMLR, 2018.

[353] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[354] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, pp. 1889–1897, PMLR, 2015.

[355] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, pp. 1928–1937, PMLR, 2016.

[356] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International conference on machine learning*, pp. 1587–1596, PMLR, 2018.

[357] J. Wang, Y. Yan, Z. Liu, C. P. Chen, C. Zhang, and K. Chen, "Finite-time consensus control for multi-agent systems with full-state constraints and actuator failures," *Neural Networks*, vol. 157, pp. 350–363, 2023.

[358] Y. Yang, R. Li, J. Huang, and X. Su, "Distributed optimal output feedback consensus control for nonlinear euler-lagrange systems under input saturation," *Journal of the Franklin Institute*, vol. 360, no. 8, pp. 5857–5877, 2023.

[359] R. Zuo, Y. Li, M. Lv, and Z. Liu, "Distributed asynchronous consensus control of nonlinear multi-agent systems under directed switching topologies," *Automatica*, vol. 152, p. 110952, 2023.

[360] H. Jian, S. Zheng, P. Shi, Y. Xie, and H. Li, "Consensus for multiple random mechanical systems with applications on robot manipulator," *IEEE Transactions on Industrial Electronics*, 2023.

[361] X. Li, M. Gao, Z. Kang, X. Chen, X. Zeng, S. Chen, H. Sun, and A. Zhang, "Cooperative path tracking for swarm of masss based on consensus theory," *Journal of Marine Science and Engineering*, vol. 11, no. 2, p. 312, 2023.

[362] T. Gai, M. Cao, F. Chiclana, Z. Zhang, Y. Dong, E. Herrera-Viedma, and J. Wu, "Consensus-trust driven bidirectional feedback mechanism for improving consensus in social network large-group decision making," *Group decision and negotiation*, vol. 32, no. 1, pp. 45–74, 2023.

[363] K. Li, C. K. Ahn, and C. Hua, "Delays-based distributed bipartite consensus control of nonlinear multiagent systems with switching signed topologies," *IEEE Transactions on Network Science and Engineering*, 2023.

[364] C. Liu, J. Zhao, B. Jiang, and R. J. Patton, "Fault-tolerant consensus control of multi-agent systems under actuator/sensor faults and channel noises: A distributed anti-attack strategy," *Information Sciences*, vol. 623, pp. 1–19, 2023.

[365] M. H. DeGroot, "Reaching a consensus," *Journal of the American Statistical Association*, vol. 69, no. 345, pp. 118–121, 1974.

[366] V. Borkar and P. Varaiya, "Asymptotic agreement in distributed estimation," *IEEE transactions on automatic control*, vol. 27, no. 3, pp. 650–655, 1982.

[367] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pp. 25–34, 1987.

[368] T. Vicsek, A. Czirók, E. Ben-Jacob, I. Cohen, and O. Shochet, "Novel type of phase transition in a system of self-driven particles," *Physical review letters*, vol. 75, no. 6, p. 1226, 1995.

[369] A. Jadbabaie, J. Lin, and A. S. Morse, "Coordination of groups of mobile autonomous agents using nearest neighbor rules," *IEEE Transactions on automatic control*, vol. 48, no. 6, pp. 988–1001, 2003.

[370] J. Liu and X. Wang, "Secure consensus control for multi-agent systems subject to consecutive asynchronous dos attacks," *International Journal of Control, Automation and Systems*, vol. 21, no. 1, pp. 61–70, 2023.

[371] A. Giuseppi, S. Manfredi, and A. Pietrabissa, "A weighted average consensus approach for decentralized federated learning," *Machine Intelligence Research*, vol. 19, no. 4, pp. 319–330, 2022.

[372] I. Ravanshadi, E. A. Boroujeni, and M. Pourgholi, "Centralized and distributed model predictive control for consensus of non-linear multi-agent systems with time-varying obstacle avoidance," *ISA transactions*, vol. 133, pp. 75–90, 2023.

[373] G. Subathra, A. Antonidoss, and B. K. Singh, "Decentralized consensus blockchain and ipfs-based data aggregation for efficient data storage scheme," *Security and Communication Networks*, vol. 2022, 2022.

[374] H. Ma, D. Liu, D. Wang, F. Tan, and C. Li, "Centralized and decentralized event-triggered control for group consensus with fixed topology in continuous time," *Neurocomputing*, vol. 161, pp. 267–276, 2015.

[375] P. Yu, K.-Z. Liu, X. Liu, X. Li, M. Wu, and J. She, "Robust consensus tracking control of uncertain multi-agent systems with local disturbance rejection," *IEEE/CAA Journal of Automatica Sinica*, vol. 10, no. 2, pp. 427–438, 2023.

[376] Y. Tian, H. Li, and Q. Han, "Finite-time average consensus of directed second-order multi-agent systems with markovian switching topology and impulsive disturbance," *Neural Computing and Applications*, pp. 1–14, 2023.

[377] H. Li, G. Chen, T. Huang, Z. Dong, W. Zhu, and L. Gao, "Event-triggered distributed average consensus over directed digital networks with limited communication bandwidth," *IEEE transactions on cybernetics*, vol. 46, no. 12, pp. 3098–3110, 2016.

[378] T. Li, M. Fu, L. Xie, and J.-F. Zhang, "Distributed consensus with limited communication data rate," *IEEE Transactions on Automatic Control*, vol. 56, no. 2, pp. 279–292, 2010.

[379] W. Ren, R. W. Beard, and E. M. Atkins, "Information consensus in multivehicle cooperative control," *IEEE Control systems magazine*, vol. 27, no. 2, pp. 71–82, 2007.

[380] W. Ren, R. W. Beard, and E. M. Atkins, "A survey of consensus problems in multi-agent coordination," in *Proceedings of the 2005, American Control Conference, 2005.*, pp. 1859–1864, IEEE, 2005.

[381] L. Moreau, "Stability of multiagent systems with time-dependent communication links," *IEEE Transactions on automatic control*, vol. 50, no. 2, pp. 169–182, 2005.

[382] W. Ren and R. W. Beard, "Consensus seeking in multiagent systems under dynamically changing interaction topologies," *IEEE Transactions on automatic control*, vol. 50, no. 5, pp. 655–661, 2005.

[383] R. A. Horn and C. R. Johnson, *Matrix analysis*. Cambridge university press, 2012.

[384] P. M. Weichsel, "The kronecker product of graphs," *Proceedings of the American mathematical society*, vol. 13, no. 1, pp. 47–52, 1962.

[385] R. Gould, *Graph theory*. Courier Corporation, 2012.

[386] O. Sporns, "Graph theory methods: applications in brain networks," *Dialogues in clinical neuroscience*, 2022.

[387] Z. Li, X. Liu, W. Ren, and L. Xie, "Distributed tracking control for linear multiagent systems with a leader of bounded unknown input," *IEEE Transactions on Automatic Control*, vol. 58, no. 2, pp. 518–523, 2012.

[388] N. Biggs, E. K. Lloyd, and R. J. Wilson, *Graph Theory, 1736-1936*. Oxford University Press, 1986.

[389] W. Yu, G. Chen, W. Ren, J. Kurths, and W. X. Zheng, "Distributed higher order consensus protocols in multiagent dynamical systems," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 58, no. 8, pp. 1924–1932, 2011.

[390] Z. Zhou and X. Wang, "Constrained consensus in continuous-time multiagent systems under weighted graph," *IEEE Transactions on Automatic Control*, vol. 63, no. 6, pp. 1776–1783, 2017.

[391] D. B. West *et al.*, *Introduction to graph theory*, vol. 2. Prentice hall Upper Saddle River, 2001.

[392] J. L. Gross, J. Yellen, and M. Anderson, *Graph theory and its applications*. Chapman and Hall/CRC, 2018.

[393] M. Porfiri and D. J. Stilwell, "Consensus seeking over random weighted directed graphs," *IEEE Transactions on Automatic Control*, vol. 52, no. 9, pp. 1767–1773, 2007.

[394] A. H. Dekker and B. D. Colbert, "Network robustness and graph topology," in *Proceedings of the 27th Australasian conference on Computer science-Volume 26*, pp. 359–368, 2004.

[395] J. A. Bondy, U. S. R. Murty, *et al.*, *Graph theory with applications*, vol. 290. Macmillan London, 1976.

[396] L. R. Foulds, *Graph theory applications*. Springer Science & Business Media, 2012.

[397] R. Merris, "Laplacian matrices of graphs: a survey," *Linear algebra and its applications*, vol. 197, pp. 143–176, 1994.

[398] R. Agaev and P. Chebotarev, "On the spectra of nonsymmetric laplacian matrices," *Linear Algebra and its Applications*, vol. 399, pp. 157–168, 2005.

[399] R. B. Bapat, "The laplacian matrix of a graph," *Mathematics Student-India*, vol. 65, no. 1, pp. 214–223, 1996.

[400] F. L. Lewis, H. Zhang, K. Hengster-Movric, and A. Das, *Cooperative control of multi-agent systems: optimal and adaptive design approaches*. Springer Science & Business Media, 2013.

[401] Z. Li, G. Wen, Z. Duan, and W. Ren, "Designing fully distributed consensus protocols for linear multi-agent systems with directed graphs," *IEEE Transactions on Automatic Control*, vol. 60, no. 4, pp. 1152–1157, 2014.

[402] Z. Li and Z. Duan, *Cooperative control of multi-agent systems: a consensus region approach*. CRC press, 2017.

[403] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *ICML*, 2010.

[404] E. Rohmer, S. P. Singh, and M. Freese, "V-rep: A versatile and scalable robot simulation framework," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1321–1326, IEEE, 2013.

[405] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, p. 5, Kobe, Japan, 2009.

[406] I. A. Sucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, 2012.

[407] H. R. Kam, S.-H. Lee, T. Park, and C.-H. Kim, "Rviz: a toolkit for real domain data visualization," *Telecommunication Systems*, vol. 60, no. 2, pp. 337–345, 2015.

[408] F. Wang, J. R. G. Olvera, and G. Cheng, "Optimal order pick-and-place of objects in cluttered scene by a mobile manipulator," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 6402–6409, 2021.

[409] K. Wu, J. Hu, B. Lennox, and F. Arvin, "Mixed controller design for multi-vehicle formation based on edge and bearing measurements," in *2022 European Control Conference (ECC)*, pp. 1666–1671, IEEE, 2022.

[410] M. Thabet, M. Patacchiola, and A. Cangelosi, "Sample-efficient deep reinforcement learning with imaginary rollouts for human-robot interaction," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5079–5085, 2019.

[411] H. Niu, Y. Lu, A. Savvaris, and A. Tsourdos, "An energy-efficient path planning algorithm for unmanned surface vehicles," *Ocean Engineering*, vol. 161, pp. 308–321, 2018.

[412] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Thirty-first AAAI conference on artificial intelligence*, 2017.

[413] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[414] M. Lutz, *Programming python*. " O'Reilly Media, Inc.", 2001.

[415] J. Ge, H. Saeidi, M. Kam, J. Opfermann, and A. Krieger, "Supervised autonomous electrosurgery for soft tissue resection," in *2021 IEEE 21st International Conference on Bioinformatics and Bioengineering (BIBE)*, pp. 1–7, IEEE, 2021.

[416] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Advances in neural information processing systems*, vol. 12, 1999.

[417] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3, pp. 279–292, 1992.

[418] R. Olfati-Saber and R. M. Murray, "Consensus problems in networks of agents with switching topology and time-delays," *IEEE Transactions on automatic control*, vol. 49, no. 9, pp. 1520–1533, 2004.

[419] K. Zhang, Z. Yang, and T. Basar, "Networked multi-agent reinforcement learning in continuous spaces," in *2018 IEEE Conference on Decision and Control (CDC)*, pp. 2771–2776, IEEE, 2018.

[420] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," 2014.

[421] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Advances in neural information processing systems*, pp. 1008–1014, 2000.

[422] I. Grondman, L. Busoniu, G. A. Lopes, and R. Babuska, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 6, pp. 1291–1307, 2012.

[423] G. H. Golub and C. F. Van Loan, *Matrix computations*, vol. 3. JHU press, 2013.

[424] V. Chellaboina and W. M. Haddad, *Nonlinear dynamical systems and control: A Lyapunov-based approach*. Princeton University Press, 2008.

[425] H. R. Maei, C. Szepesvári, S. Bhatnagar, and R. S. Sutton, "Toward off-policy learning control with function approximation," in *ICML*, 2010.

[426] T. Degris, M. White, and R. S. Sutton, "Off-policy actor-critic," *arXiv preprint arXiv:1205.4839*, 2012.

[427] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, vol. 3, pp. 2149–2154, IEEE, 2004.

[428] B. Beyret, A. Shafti, and A. A. Faisal, "Dot-to-dot: Explainable hierarchical reinforcement learning for robotic manipulation," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5014–5019, 2019.

[429] Y. Han, I. H. Zhan, W. Zhao, J. Pan, Z. Zhang, Y. Wang, and Y.-J. Liu, "Deep reinforcement learning for robot collision avoidance with self-state-attention and sensor fusion," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 6886–6893, 2022.

[430] R. Han, S. Chen, S. Wang, Z. Zhang, R. Gao, Q. Hao, and J. Pan, "Reinforcement learned distributed multi-robot navigation with reciprocal velocity obstacle shaped rewards," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 5896–5903, 2022.

[431] Y. Xu, J. Sun, Z.-G. Wu, and G. Wang, "Fully distributed adaptive event-triggered control of networked systems with actuator bias faults," *IEEE Transactions on Cybernetics*, 2021.

[432] J. Cai, H. Cheng, Z. Zhang, and J. Su, "Metagrasp: Data efficient grasping by affordance interpreter network," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 4960–4966, IEEE, 2019.

[433] S. Lambert-Lacroix and L. Zwald, "Robust regression through the huber's criterion and adaptive lasso penalty," *Electronic Journal of Statistics*, vol. 5, pp. 1015–1053, 2011.

[434] C. Godsil and G. F. Royle, *Algebraic graph theory*, vol. 207. Springer Science & Business Media, 2013.

[435] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level

control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[436] T. Haarnoja, V. Pong, A. Zhou, M. Dalal, P. Abbeel, and S. Levine, "Composable deep reinforcement learning for robotic manipulation," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6244–6251, IEEE, 2018.

[437] Y. Zhao, Y. Ma, and S. Hu, "Usv formation and path-following control via deep reinforcement learning with random braking," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.