Integration of Model Predictive Control and Reinforcement Learning for Dynamic Systems with Application to Robot Manipulators

by

Pengcheng Hu

BEng., University of Victoria, 2021

A Thesis Submitted in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF APPLIED SCIENCE

in the Department of Mechanical Engineering

Integration of Model Predictive Control and Reinforcement Learning for Dynamic Systems with Application to Robot Manipulators

by

Pengcheng Hu

BEng., University of Victoria, 2021

Supervisory Committee

Dr. Yang Shi, Supervisor

(Department of Mechanical Engineering)

Dr. Daniela Constantinescu, Departmental Member

(Department of Mechanical Engineering)

ABSTRACT

The last decade has witnessed great progress in the development of reinforcement learning (RL) across many applications, such as games and autonomous driving. RL is effective in solving control problems for complex systems whose dynamics are intractable to be accurately modeled. In an RL algorithm, the agent learns the optimal policy in terms of the maximum reward based on measurement samples from the interactions with the environment. To obtain the optimal policy, RL requires collecting sufficiently large number of samples, which is challenging in real-world applications, e.g., robotics, manufacturing, and so on. To tackle this problem, model predictive control-based RL (MPC-based RL) is proposed to improve the sample efficiency. In the MPC-based RL algorithm, a model is learned from collected samples, the learned model and MPC are utilized to predict trajectories over a specified prediction horizon, and an action is obtained through the RL algorithm by maximizing the cumulative reward. This thesis is devoted to the investigation of the MPC-based RL design and its application to robot manipulators.

In Chapter 2, an MPC-based deep RL framework for constrained linear systems with bounded disturbances is proposed. In the proposed framework, a rigid tube-based MPC (RTMPC) method is employed to predict a trajectory by solving the corresponding optimization problem. Then, the predicted trajectory is stored in a replay buffer as the form of data pairs. Further, the soft actor-critic (SAC) algorithm is applied to modify the loss function and update the policy online, based on the predicted data pairs. Numerical simulations validate the effectiveness of the proposed method. In addition, comparison results demonstrate the advantages of the proposed method including requirement of fewer real samples and providing better control performance with comparable computational complexity to RTMPC.

In Chapter 3, we investigate the application of three methods for manipulators.

Firstly, we apply an MPC-based RL algorithm, a nonlinear MPC (NMPC) method, and two model-free RL algorithms to tackle the regulation problem for a 2-degree-of-freedom manipulator system, and compare their training control performance. Secondly, the training and control performance evaluation for the model-free RL algorithm and the MPC-based RL algorithm are provided. The MPC-based RL algorithm shows better training performance in terms of sample efficiency and total return but poorer control performance. Thirdly, simulation studies are provided to compare the training performance of the MPC-based RL algorithm and two model-free RL algorithms. From the simulation results, the MPC-based RL algorithm presents poorer training performance compared with model-free RL algorithms for the twelve-dimensional system.

In Chapter 4, conclusions and future work are summarized.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

The completion of this thesis is a significant milestone in my academic journey, and it would not have been possible without the support, guidance, and encouragement of many individuals. I would like to take this moment to express my sincere gratitude to those who have helped me to achieve this goal.

First and foremost, I would like to extend my deepest thanks to my supervisor, Dr. Yang Shi. His expertise, constructive feedback, and constant support were invaluable throughout my M.A.Sc. program. He provided not only academic guidance but also the motivation needed to overcome many challenges. Moreover, his understanding and support during the time of the COVID-19 pandemic were incredibly reassuring and provided me with the strength to keep moving forward, both academically and personally. Once again, I am grateful for his patience and encouragement to strive for excellence.

I would like to extend my deepest thanks to our ACIPL group members who provided lots of professional research suggestions. Specifically, I would thank Dr. Kunwu Zhang for helping me revise my conference paper and learn MPC's coding. I would also say thanks to Xinxin Shang for providing advice of my future career. I would express my appreciation to Tianxiang Lu for attending the 7th IEEE ICPS with me in St. Louis. Moreover, I would thank Yue Song for introducing me to the lab and being my classmate in the advanced optimization course. I would also thank Dr. Xin Huang, Yufan Dai, and Linwei Guo for being my basketball teammates every weekend. Furthermore, I would express my sincere appreciation to Wentao Wu, Zhaoming Sheng, and Ying Zhang for helping me with the revision of this thesis. In addition, I would thank Dr. Tianyu Tan, Dr. Binyan Xu, Dr. Henglai Wei, Dr. Sen Li, Xiang Sheng, Rui Zhao, Zheng Li, Lei Xu, Wencong Zhang, and many other students for every moment we spent together.

Lastly, I am especially grateful to my family, whose unconditional love and unwavering support have been my anchor throughout my M.A.Sc. program. They have been my biggest cheerleaders, always encouraging me to pursue my dreams during moments of self-doubt. Their sacrifices and understanding support me to accomplish this achievement.

Acronyms

RL	reinforcement learning
DRL	deep reinforcement learning
TD	temporal difference
BOE	Bellman optimality equation
MDP	Markov decision process
DQN	deep Q network
DDPG	deep deterministic policy gradient
SAC	soft actor-critic
MBMF	model-based model-free
MBPO	model-based policy optimization
EBPNNs	ensemble of bootstrapped probabilistic neural networks
MPC	model predictive control
NMPC	nonlinear model predictive control
RPI	robust positively invariant
RMPC	robust model predictive control
RTMPC	rigid tube-based model predictive control
LTI	linear time-invariant
MPPI	model predictive path integral
AVs	autonomous vehicles
IT-MPC	information-theoretic model predictive control
KL	Kullback–Leibler
DOF	degree-of-freedom

Chapter 1

Introduction

1.1 Reinforcement Learning (RL)

Over the last decade, considerable attention has been directed toward reinforcement learning (RL) due to its capacity for exploring optimal behaviors in many areas [30]. RL differs from supervised learning, which directly learns from a labeled training set [49], and unsupervised learning, which learns from an unlabeled training set [25]. The essential idea of RL is to iteratively solve the Markov decision process (MDP) and to learn a policy, from states to actions, that maximizes the cumulative reward over a period of time [77]. There are seven key components of RL algorithms, i.e., an agent, an environment, a state space, an action space, a policy, a value function, and a reward. Specifically, the policy, the reward, and the value function are described as follows:

- A policy is a strategy from various states of the environment to actions that an agent takes.
- A reward is the immediate feedback that an agent receives for taking an action.
- A value function estimates the total reward from a given state to the desired

state.

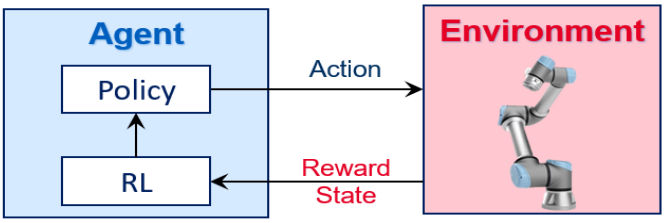

Figure 1.1: The system diagram of RL [1]

From Figure 1.1, we can see that an agent receives a state and a reward from the environment. The reward is utilized to improve the policy by using RL algorithms. Then, the agent applies an action which is obtained based on the improved policy. Lastly, the environment will update the state, and the agent will receive a new state at the next time instant.

1.1.1 Traditional RL

Traditional RL algorithms, such as ϵ-greedy [86], count-based exploration approach [78], Q-learning [81], and SARSA [64], utilize a lookup table or linear function approximation to solve systems with discrete state and action spaces. To explicitly illustrate discrete state and action spaces, a cliff walking environment is given in Figure 1.2, as a classical example for traditional RL algorithms. Consider a grid world with a limited number of states, an agent needs to find the shortest path from a start position to a goal position. At each step, the agent can choose an action from an action space (i.e., move up, down, left, right). In this study, we assume that the agent is awarded a reward of 0 upon successfully reaching the designated goal position and a penalty of -100 for any instance of falling off a cliff. In all other circumstances, the agent is subjected to a constant reward of -1 for each step taken. Following the develop-

[1] https://www.universal-robots.com/products/ur5-robot/

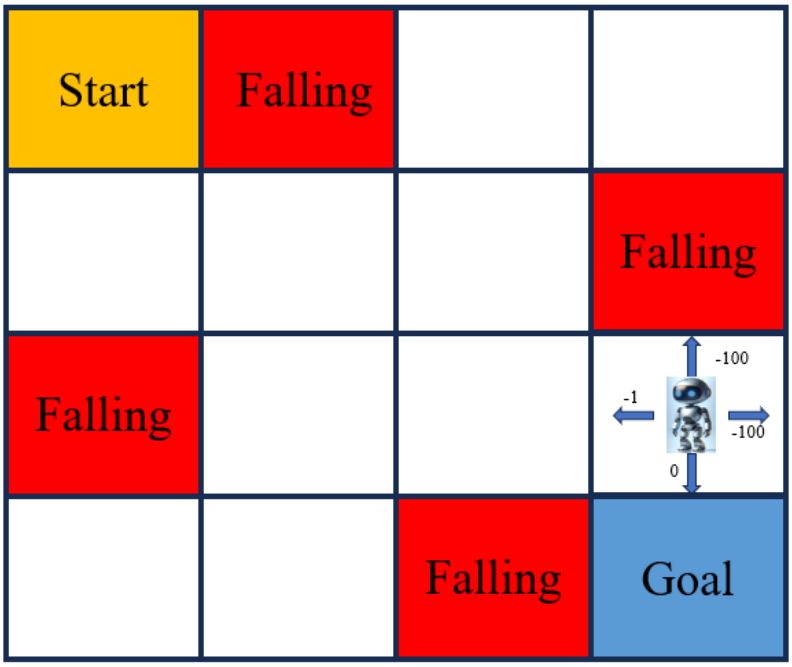

Figure 1.2: Environment of cliff walking

ment of the environment for a cliff walking task, we can obtain a state-action-reward trajectory, starting from the initial time instant $k = 0$,

$$\boldsymbol{x}_k \xrightarrow[r(\boldsymbol{x}_k,\boldsymbol{u}_k)]{\boldsymbol{u}_k} \boldsymbol{x}_{k+1} \xrightarrow[r(\boldsymbol{x}_{k+1},\boldsymbol{x}_{k+1})]{\boldsymbol{x}_{k+1}} \boldsymbol{x}_{k+2} \xrightarrow[r(\boldsymbol{x}_{k+2},\boldsymbol{x}_{k+2})]{\boldsymbol{x}_{k+2}} \boldsymbol{x}_{k+3} \cdots,$$

where $\boldsymbol{x}_k \in \mathbb{X}$, $\boldsymbol{u}_k \in \mathbb{U}$, $r(\boldsymbol{x}_k, \boldsymbol{u}_k)$ are the system state, the action, and the immediate reward at time instant k, respectively. Therefore, the discounted total return of the trajectory is

$$\begin{aligned} G(\boldsymbol{x}_k) &= r(\boldsymbol{x}_k, \boldsymbol{u}_k) + \gamma r(\boldsymbol{x}_{k+1}, \boldsymbol{x}_{k+1}) + \gamma^2 r(\boldsymbol{x}_{k+2}, \boldsymbol{x}_{k+2}) \cdots \\ &= r(\boldsymbol{x}_k, \boldsymbol{u}_k) + \gamma G(\boldsymbol{x}_{k+1}), \end{aligned}$$

where $\gamma \in [0, 1]$ is the discount rate. As aforementioned, the value function is one of the most important components of an RL algorithm. The state-value function is

obtained by calculating the expectation of total returns

$$\begin{aligned} V(\boldsymbol{x}_k) &= \mathbb{E}[G(\boldsymbol{x}_k)] \\ &= \mathbb{E}[r(\boldsymbol{x}_k)] + \gamma \mathbb{E}[G(\boldsymbol{x}_{k+1})]. \end{aligned} \tag{1.2}$$

It is worth noting that $\mathbb{E}[r(\boldsymbol{x}_k)]$ in (1.2) is the expectation of the immediate reward that can be received from state $\boldsymbol{x}_k$. The second term $\mathbb{E}[G(\boldsymbol{x}_{k+1})]$ is the expectation of the future total returns. By utilizing the law of total expectation, one of the most fundamental equations, the Bellman equation [7,8] is obtained from (1.2) and can be written as

$$V(\boldsymbol{x}_k) = \sum_{\boldsymbol{u}_k} \pi(\boldsymbol{u}_k|\boldsymbol{x}_k) \sum_{\boldsymbol{x}_{k+1} \in \mathbb{X}} p(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k) \Big[r(\boldsymbol{x}_k, \boldsymbol{u}_k) + \gamma V(\boldsymbol{x}_{k+1}) \Big], \tag{1.3}$$

where $\pi(\boldsymbol{u}_k|\boldsymbol{x}_k)$ denotes the policy; $\gamma \in [0, 1]$ is the discount rate; $p(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k)$ represents the probability for an agent moving from the state $\boldsymbol{x}_k$ to the next state $\boldsymbol{x}_{k+1}$ by taking the action $\boldsymbol{u}_k$. We next introduce the Bellman equation in the form of action-value function. Similar to the state-value function, the action-value function calculates the expectation of total returns, starting from $\boldsymbol{x}_k$ and taking $\boldsymbol{u}_k$:

$$\begin{aligned} Q(\boldsymbol{x}_k, \boldsymbol{u}_k) &= r(\boldsymbol{x}_k, \boldsymbol{u}_k) + \gamma \mathbb{E}[G(\boldsymbol{x}_{k+1})] \\ &= r(\boldsymbol{x}_k, \boldsymbol{u}_k) + \gamma \sum_{\boldsymbol{x}_{k+1} \in \mathbb{X}} p(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k, \boldsymbol{u}_k) V(\boldsymbol{x}_{k+1}). \end{aligned} \tag{1.4}$$

In RL algorithms, the optimal policy can be obtained if the maximum value $V^*(\boldsymbol{x}_k)$ or $Q^*(\boldsymbol{x}_k, \boldsymbol{u}_k)$ can be found [8,13,58]. Thus, the Bellman optimality equation (BOE) should be introduced as a significant mathematical component in RL algorithms. In order to obtain the maximum value, the agent, at each time instant, must choose an action that can reach the next state with the best reward. We can derive $V^*(\boldsymbol{x}_k)$ and

$Q^*(\boldsymbol{x}_k, \boldsymbol{u}_k)$ in the form

$$V^*(\boldsymbol{x}_k) = \max_{\boldsymbol{u}_k \in \mathbb{U}} Q^*(\boldsymbol{x}_k, \boldsymbol{u}_k), \tag{1.5a}$$

$$Q^*(\boldsymbol{x}_k, \boldsymbol{u}_k) = r(\boldsymbol{x}_k, \boldsymbol{u}_k) + \gamma \max_{\boldsymbol{u}_{k+1} \in \mathbb{U}} Q^*(\boldsymbol{x}_{k+1}, \boldsymbol{u}_{k+1}). \tag{1.5b}$$

Another key concept of traditional RL algorithms is the temporal difference (TD) method [77] where the predicted state or action-value at the current time instant is updated based on the actual value at the next time instant. The primary intuition behind TD method is to minimize the TD error, which represents the difference between the predicted value and the actual value. Therefore, the update of the action-value function under the TD method can be written as

$$\begin{aligned} Q(\boldsymbol{x}_k, \boldsymbol{u}_k)_{\text{new}} \leftarrow & Q(\boldsymbol{x}_k, \boldsymbol{u}_k)_{\text{current}} + \lambda \Big(r(\boldsymbol{x}_k, \boldsymbol{u}_k) \\ & + \gamma \sum_{\boldsymbol{x}_{k+1} \in \mathbb{X}} p(\boldsymbol{x}_{k+1} | \boldsymbol{x}_k, \boldsymbol{u}_k) V(\boldsymbol{x}_{k+1}) - Q(\boldsymbol{x}_k, \boldsymbol{u}_k)_{\text{current}} \Big), \end{aligned} \tag{1.6}$$

where $\lambda \in [0, 1]$ is the learning rate; $r(\boldsymbol{x}_k, \boldsymbol{u}_k) + \gamma \sum_{\boldsymbol{x}_{k+1} \in \mathbb{X}} p(\boldsymbol{x}_{k+1} | \boldsymbol{x}_k, \boldsymbol{u}_k) V(\boldsymbol{x}_{k+1})$ is called the TD target, which is obtained from a Q-value estimator or Q-table. Therefore, the TD error can be expressed

$$\delta = r(\boldsymbol{x}_k, \boldsymbol{u}_k) + \gamma \sum_{\boldsymbol{x}_{k+1} \in \mathbb{X}} p(\boldsymbol{x}_{k+1} | \boldsymbol{x}_k, \boldsymbol{u}_k) V(\boldsymbol{x}_{k+1}) - Q(\boldsymbol{x}_k, \boldsymbol{u}_k)_{\text{current}}. \tag{1.7}$$

Traditional RL algorithms repeatedly improve the policy based on the TD method and value functions determined from different actions until achieving an optimal policy. One of the most famous traditional RL algorithms is Q-learning [81], which can obtain the optimal policy by solving the BOE and utilizing the TD method. The update

process under Q-learning can be described as

$$Q(\boldsymbol{x}_k, \boldsymbol{u}_k)_{\text{new}} \leftarrow Q(\boldsymbol{x}_k, \boldsymbol{u}_k)_{\text{current}} + \lambda\Big(r(\boldsymbol{x}_k, \boldsymbol{u}_k) + \gamma \sum_{\boldsymbol{x}_{k+1}\in\mathbb{X}} p(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k, \boldsymbol{u}_k) \max_{\boldsymbol{x}_{k+1}\in\mathbb{U}} Q(\boldsymbol{x}_{k+1}, \boldsymbol{x}_{k+1}) - Q(\boldsymbol{x}_k, \boldsymbol{u}_k)_{\text{current}}\Big). \tag{1.8}$$

Traditional RL algorithms have been extensively employed across a myriad of domains, such as robotics [6, 34, 35] and unmanned aerial vehicles (UAVs) [11, 28, 80]. However, as aforementioned, traditional RL algorithms are inherently designed for systems characterized by discrete state and action spaces, which implies that traditional RL algorithms are unsuitable for systems with continuous state and action spaces. This limitation primarily stems from the fact that traditional RL algorithms rely on tabular methods to store actions associated with each state. When confronted with a vast or infinite number of states or actions, it becomes impracticable to construct such a table due to the computational complexity [3].

1.1.2 Deep RL

Considering that it is difficult to obtain the optimal policy for systems with a large number of states and actions by using traditional tabular methods, deep reinforcement learning (DRL) has been proposed as an efficient methodology [3]. The principal idea of DRL is to integrate deep neural networks as value function estimators. Therefore, the DRL algorithms improve traditional RL algorithms with two more key enhancements.

- DRL incorporates an experience replay buffer β during its training process. The buffer β stores the tuples $\{\boldsymbol{x}_k, \boldsymbol{u}_k, r(\boldsymbol{x}_k, \boldsymbol{u}_k), \boldsymbol{x}_{k+1}\}$ which are online obtained through agent-environment interactions at each discrete-time step k. During the training process, a minibatch of transition samples is randomly selected from

β, and is used to update the network parameter θ by employing the stochastic gradient descent (SGD) algorithm [10].

- DRL algorithms employ deep neural networks to approximate the current Q-value, and use a separate network to generate the target Q-value. Specifically, the output of the current Q-value network $Q_\theta(x_k, \boldsymbol{u}_k)$ is utilized to assess the value function associated with the present state-action pairs, while the output of the target Q-value network $Q_{\theta^-}(\boldsymbol{x}_k, \boldsymbol{u}_k)$ serves as an optimization target. The parameters of the current Q-value network θ are updated in real-time, by minimizing the error between the current Q-value and the target Q-value. The error is normally described as a TD error function, which is also known as the loss function. The loss function for updating the current Q-value can be described as

$$L = \frac{1}{N}\sum_{k}^{E}(r(\boldsymbol{x}_k, \boldsymbol{u}_k) + Q_{\theta^-}(\boldsymbol{x}_{k+1}, \boldsymbol{u}_{k+1}) - Q_\theta(\boldsymbol{x}_k, \boldsymbol{u}_k))^2,$$

where E is the size of the minibatch sampled from β. The parameters of the target Q-value network θ^- are updated based on the current Q-value network, using different updating approaches, such as the soft update method [23, 41] and the direct copy method [48].

Based on the aforementioned improvement, deep Q network (DQN) [48] was proposed and has become one of the most classical DRL algorithms. However, DQN can only handle systems with continuous state spaces coupled by discrete and low-dimensional action spaces. The reason behind this limitation is illustrated as follows. To solve the BOE (1.5b) and update the action-value function (1.8), the agent has to choose an action that maximizes the action-value function for the next time instant. However, it is difficult to determine an optimal action from a continuous and high-

dimensional action space. Thus, DQN faces challenges in learning optimal policies for the environments requiring actions from an infinite set, such as manipulating the angle of robotic arms. To remove the above limitations, a deep deterministic policy gradient (DDPG) algorithm was proposed to handle the system with a continuous and high-dimensional action space in [41]. Compared with DQN, the DDPG algorithm has the following three improvements: (1) Based on the deterministic policy gradient (DPG) [73], the DDPG algorithm utilizes an actor-critic architecture [36], which can deal with a continuous action space; (2) Unlike the direct copy method for updating target Q-value network, the DDPG algorithm uses soft target update method to calculate target values for both critic and actor networks. This soft update technique results in a stable target; (3) To achieve the exploration performance in continuous action spaces, the DDPG algorithm adds noises to the outputs of the policy. In Section 3.3.1, we will introduce the DDPG algorithm with its loss functions.

DRL has been proven to be efficient in achieving superhuman performance in many practical applications, such as the game of Go [72,74], video games [19,47,53,85], and robotics [21,27,31,65]. However, DRL directly learns optimal strategies from raw data without requiring explicit system dynamics, which naturally brings several challenges related to training safety and converging rate, especially for controlled systems.

1.2 Model Predictive Control (MPC)

The versatility of MPC has been demonstrated across a diverse range of applications, such as automobiles [32], unmanned aerial vehicles [71], and marine robots [69, 70]. This section provides an overview of MPC, including its formulation, strengths, and limitations. MPC has recently emerged as a preeminent method to address complex systems with constraints [39]. At each control time instant, MPC exploits an explicit

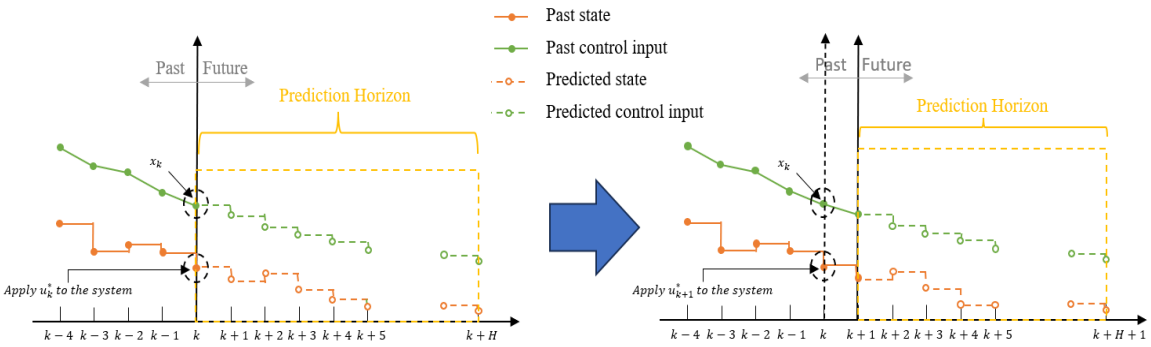

Figure 1.3: The schematic diagram of MPC

system model to predict system states over a predefined prediction horizon. Subsequently, MPC constructs and solves an optimization problem to determine a sequence of optimal control inputs that minimize a specific cost function while satisfying system constraints. Only the first control input is implemented upon determining the optimal control sequence. This process is repeated at the next time instant based on new measurements [37].

1.2.1 MPC Formulation

Consider a discrete-time, deterministic system

$$\boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{u}_k), \tag{1.9}$$

where $\boldsymbol{x}_k \in \mathbb{R}^n$ and $\boldsymbol{u}_k \in \mathbb{R}^m$ are the system states and the control inputs, respectively. The system is subject to hard constraints $\boldsymbol{x}_k \in \mathbb{X}$ and $\boldsymbol{u}_k \in \mathbb{U}$, where the sets $\mathbb{X}$ and $\mathbb{U}$ are compact and convex. Each set of $\mathbb{X}$ and $\mathbb{U}$ contains the origin in its interior. The function $f(\cdot)$ is the actual system dynamics, which can be linear or nonlinear. For the case that the equilibrium point of system in (1.9) is the origin, the control objective is to regulate the state (1.9) to the origin.

Ideally, the main insight of MPC strategy is to obtain a sequence of control inputs that minimize a cost function for the current system state. The cost function can be described as

$$J(\boldsymbol{x}_k, \boldsymbol{u}_k) = \sum_{l=k}^{\infty} \ell(\boldsymbol{x}_{l|k}, \boldsymbol{u}_{l|k}), \tag{1.10}$$

where $\boldsymbol{x}_{l|k}$ and $\boldsymbol{u}_{l|k}$ represent the predicted system state and control input, respectively, l steps ahead from time k; $\ell(\boldsymbol{x}_{l|k}, \boldsymbol{u}_{l|k})$ denotes the stage cost function. $\boldsymbol{u}_k$ and $\boldsymbol{x}_k$ are the predicted control sequence and state sequence.

However, it becomes computationally intractable to optimize the cost function (1.10) in practical applications, primarily due to the infinite prediction horizon and a large number of optimization variables. To make the optimization problem feasible, a method was proposed to divide the infinite prediction horizon into two parts [15]. The first part predicts system states and control inputs over a finite prediction horizon, whereas the second part represents the trajectory of the subsequent infinite prediction horizon under a pre-designed feedback control law. Furthermore, a terminal state inequality constraint must be taken into account with the guarantee of feasibility and stability. Thus, the conventional MPC problem can be expressed as

$$\min_{u_{k|k},\cdots} J(\boldsymbol{x}_k, \boldsymbol{u}_k) = \sum_{l=k}^{k+N-1} \ell(\boldsymbol{x}_{l|k}, \boldsymbol{u}_{l|k}) + \ell_f(\boldsymbol{x}_{k+N|k}), \tag{1.11a}$$

$$\text{s.t. } \boldsymbol{x}_{k|k} = \boldsymbol{x}_k, \tag{1.11b}$$

$$\boldsymbol{x}_{l+1|k} = f(\boldsymbol{x}_{l|k}, \boldsymbol{u}_{l|k}), \tag{1.11c}$$

$$\boldsymbol{x}_{l|k} \in \bar{\mathbb{X}}, \boldsymbol{u}_{l|k} \in \bar{\mathbb{U}}, l = k, \ldots, k+N-1 \tag{1.11d}$$

$$\boldsymbol{x}_{k+N|k} \in \mathbb{X}_f, \tag{1.11e}$$

where N is the predefined prediction horizon; $\mathbb{X}_f$ is the terminal constraint set; $\ell_f(\boldsymbol{x}_{k+N|k})$ denotes the terminal cost function. At the sampling time instant k, the

optimal control input can be obtained by solving the MPC problem (1.11).

1.2.2 Robust MPC

As shown in the last section, MPC is a promising method to predict system states and obtain control inputs for the deterministic system dynamics. However, the conventional MPC has the limited robustness [71], making it difficult to tackle the uncertainties. Therefore, robust MPC (RMPC) has been developed as an extension of the conventional MPC to handle uncertainties. RMPC is a particularly powerful method in applications where the system dynamics may not be perfectly accurate. Notably, a sequence of control inputs is obtained from the RMPC framework, while satisfying system constraints and guaranteeing robust stability. The existing results in RMPC can be categorised into two main methods: tube-based methods and min-max optimization-based methods. The min-max optimization-based methods require a high computational burden compared with tube-based methods. Thus, in this section, we briefly introduce the tube-based MPC.

The essential idea of tube-based MPC is to keep all possible trajectories within a tube around the nominal trajectory [71]. To accomplish this, a robust positively invariant (RPI) set needs to be determined based on the local feedback control law. Moreover, the RPI set and the feedback control law can be calculated either online or offline, resulting in different tube-based MPC methods. A rigid tube-based MPC (RTMPC) method was proposed in [46], which tightens system constraints by offline calculating the RPI set and feedback control law. At each time instant, this work considers an initial system state and a control input sequence as decision variables. Therefore, by offline determining the RPI set and the feedback control law, the structure of tube cross-sections is fixed during the online optimization problem. Compared with the conventional MPC method, this method has a comparable computational

complexity. Later, the work in [59] proposed homothetic tube-based MPC (HTMPC) with reduced conservatism. In HTMPC, while the shape of cross-section is determined offline, the sizes of tube cross-sections are constantly updated by considering the tube parameter as an additional decision variable in the optimization problem. Moreover, the elastic tube-based MPC (ETMPC) was proposed in [60] as an extension of HTMPC. ETMPC online updates the shape and size of tube cross-section. By introducing different types of RMPC with tube-based methods, we conclude that RMPC is a powerful control technique to deal with uncertainties. However, with less conservatism, this control method has higher computational complexity. Thus, this limitation poses a challenge when applying RMPC to tackle practical control problems with highly nonlinear dynamics and uncertainties.

1.3 Combination of MPC and RL

As aforementioned, MPC determines the predicted sequences by minimizing a designed cost function, while satisfying system constraints. RL aims to explore an optimal policy that maximizes cumulative rewards. The primary objective of these two different methods is to obtain an optimal control input (action) with a given state. MPC utilizes explicit system dynamics to solve optimization problems, while RL is a model-free method that an agent uses collected sample to learn an optimal policy. When accurate system dynamics are complicated to establish, RL has emerged as a powerful learning method to improve the performance of MPC. Therefore, many research results have been made for combining RL and MPC. The existing results in this research direction can be classified into two categories.

- One category is RL-based MPC, such as work [20, 44, 88]. In these results, RL algorithms are utilized to update system dynamics and tune MPC parameters

while guaranteeing the stability. The updated system dynamics and parameters are implemented in the MPC optimization problem, resulting in better control performance. However, these results require a high computational complexity due to jointly updating the policy and solving the optimization problem at each time instant. Moreover, these results consider low-dimensional system dynamics in their simulation, such as two-dimensional nonlinear system dynamics in [20, 88] and four-dimensional linear system dynamics in [44]. This limitation makes RL-based MPC challenging to implement in practical applications.

- Another category is MPC-based RL which combines MPC with RL. MPC-based RL has received considerable attention due to its ability to tackle challenging control problems for complex systems. Further, a literature review of MPC-based RL will be given.

In RL algorithms, model-free algorithms [23,41,48,81] offer a reliable approach for directly learning a policy that maps states to actions. However, a significant drawback of model-free techniques is the requirement of extensive samples to obtain an optimal policy. Thus, improving sample efficiency during the training process has emerged as a considerable research topic. A well-known solution involves adopting model-based techniques [18, 38, 76], which utilize either function approximators or Bayesian models to simulate the real environment. During the process of training the policy, the utilization of data from both the learned model and the real environment can significantly enhance sample efficiency [57]. Another solution, MPC-based RL, offers an improved sample efficiency over both model-free and model-based RL techniques.

The essential idea of MPC-based RL is to predict future state and action sequences by using MPC, and then utilize RL algorithms to optimize the policy with the assistance of predicted sequences. Different from the optimization problem (1.11)

in conventional MPC, MPC-based RL considers the following optimization problem

$$\min_{u_{k|k},\cdots} \sum_{l=k}^{k+N-1} r(\boldsymbol{x}_{l|k}, \boldsymbol{u}_{l|k}), \tag{1.12a}$$

$$\text{s.t. } \boldsymbol{x}_{l+1|k} = f_\theta(\boldsymbol{x}_{l|k}, \boldsymbol{u}_{l|k}), \tag{1.12b}$$

$$\boldsymbol{x}_{k|k} = \boldsymbol{x}_k, \tag{1.12c}$$

where f_θ is the learned model. Due to the high nonlinearity of f_θ, it is intractable to directly calculate the optimum of (1.12). Thus, many research efforts have been made to obtain approximate optimal solutions of (1.12) by using sampling-based methods. The authors in [82] proposed a model predictive path integral (MPPI) method and applied it to stochastic systems. In the MPPI method, the learned model is characterized by full-connected, multi-layer, neural networks. At each time instant, the MPPI method utilizes the importance sampling method to obtain H predicted action sequences over a finite N prediction horizon. These predicted action sequences are then implemented in the learned model to obtain the predicted state sequences. Afterward, the predicted action and state sequences are utilized to update the importance sampling weight and minimize the Kullback–Leibler divergence between the optimal and current action sequence distributions. Later in [84], an extension of the MPPI method, information-theoretic MPC (IT-MPC), was proposed by incorporating the information-theoretic approach to the MPPI method. The IT-MPC method was successfully applied to an autonomous vehicles (AVs) application in [83]. Recently, the constrained tube-MPPI method [5] and the covariance-controlled MPPI (CC-MPPI) method [87] were proposed based on the IT-MPC method. However, due to the computational complexity, these results considered seven-dimensional systems with partially unknown dynamics. To address this limitation, a model-based DRL with model-free fine-tuning (MBMF) algorithm was proposed in [52]. Figure

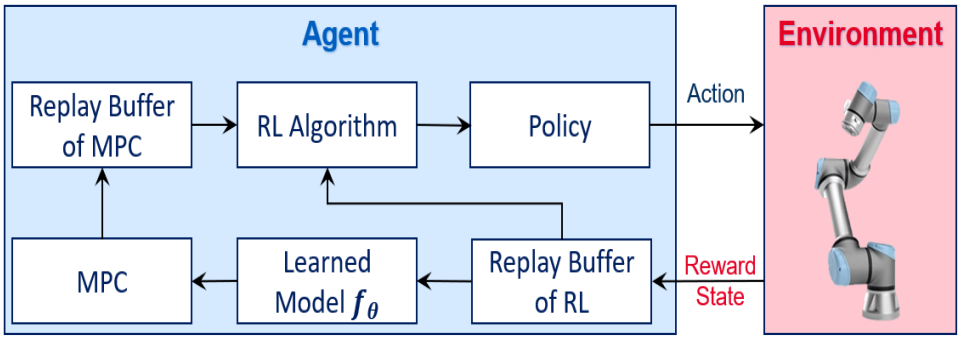

Figure 1.4: The system diagram of MPC-based RL

1.4 illustrates a typical diagram that combines MPC and a model-free RL algorithm. The MBMF algorithm employs the random shooting (RS) method to generate H predicted action sequences and identify the corresponding predicted state sequences using the learned model, which is constructed with multi-layer neural networks. Then, a DRL algorithm is applied to update the parameters of the learned model by using the predicted samples and real samples. By combining MPC with the trust region policy optimization (TRPO) [67] algorithm, the MBMF algorithm shows better performance in term of sample efficiency on high-dimensional tasks from the MuJoCo physics engine, e.g., twenty-three-dimensional half-cheetah and forty-one-dimensional ant. Previous results establish a learned model with multi-layer neural networks, and the learned model outputs a deterministic state $\boldsymbol{x}_{l+1|k}$ by taking the action $\boldsymbol{u}_{l|k}$ at the state $\boldsymbol{x}_{l|k}$. However, there are two types of uncertainty: aleatoric uncertainty and epistemic uncertainty [26], which cannot be modeled by deterministic neural networks. To handle the aleatoric uncertainty and epistemic uncertainty, probabilistic neural networks capture aleatoric uncertainty, such as observation noise and process noise, by outputting the mean and the covariance of a distribution. Moreover, epistemic uncertainties are estimated by an ensemble of bootstrapped models. Based on the above two techniques, K. Chua and R. Calandra enhanced the MBMF with an ensemble

of bootstrapped probabilistic neural networks, resulting in the algorithm known as PETS in [16]. Instead of using the RS method to solve the MPC optimization problem (1.12) in [52], the PETS algorithm utilizes the cross-entropy method (CEM) to obtain predicted rollouts over a specified prediction horizon. Compared with prior RL algorithms, such as SAC [23], proximal policy optimization (PPO) [68], and MBMF, the PETS algorithm requires 8 times, 125 times, and 2 times fewer samples, respectively, to achieve the optimal policy on the twenty-three-dimensional half-cheetah task. Although many research efforts have been made to enhance the accuracy of the learned model, a model exploitation problem has recently occurred in MPC-based learning as a primary challenge. The model exploitation problem refers that an agent overly exploits the learned model to maximize short-term rewards, and neglects exploration of true dynamics. This exploitation can lead the algorithm's policy to a local optimum. To address this problem, the work in [29] proposed a model-based policy optimization (MBPO) algorithm that leverages an ensemble of bootstrapped probabilistic neural networks (EBPNNs) and a large number of rollouts, over a short prediction horizon, to solve the MPC optimization (1.12). The theoretical analysis and simulation results demonstrate that the MBPO algorithm can handle the model exploitation problem with a sufficiently short prediction horizon. Further, the MBPO algorithm leads to enhanced performance regarding total return and sample efficiency compared with PETS and model-free RL algorithms.

In the literature, there are two main categories of combining MPC and RL, i.e., RL-based MPC and MPC-based RL. A brief summary is presented in Table 1.1.

Table 1.1: Classification of the combination of RL and MPC

MPC-based RL			
Related work	Constraint types	System dynamics	Methods
[82]	Unconstrained	Nonlinear systems	MPPI
[83]	Unconstrained	Nonlinear systems	IT-MPC
[84]	Unconstrained	AVs	IT-MPC
[87]	Unconstrained	Nonlinear systems	CC-MPPI
[5]	Constrained	Nonlinear systems	Tube-MPPI
[52]	Unconstrained	Nonlinear systems	RS + TRPO
[16]	Unconstrained	Nonlinear systems	CEM
[29]	Unconstrained	Nonlinear systems	EBPNNs+SAC
RL-based MPC			
Related work	Constraint types	System dynamics	Methods
[88]	Constrained	Linear systems	Q-learning + RMPC
[20]	Constrained	Linear systems	Q-learning + NMPC
[44]	Constrained	Linear systems	SARSA + RTMPC

1.4 Motivation and Proposed Methodologies

As seen from the aforementioned introduction, although existing results on MPC-based RL outperform model-free RL in term of sample efficiency, the MPC-based RL algorithms still need a large number of real samples to obtain the optimal policy. Specifically, the MBPO algorithm requires around 4000, 100000, 300000, and 500000 on inverted pendulum, hopper, ant, and half-cheetah tasks. This limitation makes MPC-based RL intractable to practical applications. To tackle this challenging problem, we introduce an MPC-based RL algorithm for uncertain linear systems. The essential idea of this work lies in a novel integration of MPC and DRL. In the proposed method, RTMPC is utilized to obtain the predicted state and action se-

quences. Then the predicted sequences are employed in the SAC algorithm to form TD targets and to update the policy online. By utilizing the generated TD targets in the DRL-MPC framework, the optimal policy is obtained without requiring a large number of real datasets. Moreover, the proposed method has a comparable computational complexity to RTMPC. Numerical simulations and comparisons are performed to demonstrate that our framework leads to better performance at the convergence rate towards the equilibrium point..

Many research efforts have been made to develop control methods, such as MPC, RL, and MPC-based RL, for robot manipulator systems. Since that these control methods are designed for various applications with different motivations, we aim to explore the performance of existing MPC-based RL algorithms, MPC methods, and model-free RL algorithms for regulation and tracking problems of robot manipulator systems. Three comparison cases are conducted from the following perspectives.

- Simulation and comparison studies are conducted on a 2-degree-of-freedom (DOF) and a 6-DOF manipulator systems by using nonlinear MPC [15], DDPG [41], SAC [23], and MBPO [29] algorithms.

- The comparison studies focus on the regulation and tracking problems. Further, the control and training performance of the representative methods and algorithms are investigated.

1.5 Thesis Organization

The remainder of this thesis is organized as follows.

Chapter 2 proposes an SAC-RTMPC framework for a class of constrained linear systems with bounded additive disturbances. Numerical simulations and comparisons are provided to demonstrate that the proposed framework can achieve better control

performance than conventional MPC.

Chapter 3 investigates the application of three representative methods for robot manipulators by a series of simulation studies. Firstly, an MPC-based RL algorithm, two model-free RL algorithms, and a nonlinear MPC method are employed to conduct simulation and comparison studies for a 2-DOF manipulator based on its dynamics model. Secondly, model-free RL and MPC-based RL algorithms are applied to a 2-DOF manipulator using the kinematics model. Thirdly, an MPC-based RL and two model-free RL algorithms are implemented to UR10e which is a 6-DOF manipulator system in practical.

Chapter 4 concludes this thesis and lists the future research directions.

Chapter 2

A Framework of MPC-based RL for Uncertain Linear Systems

2.1 Introduction

Due to the ability to handle constraints, MPC provides a potential solution to meet safety requirement in the RL training process. Therefore, many research efforts have been devoted to combining MPC and RL to inherit the benefits of both RL and MPC [4,44,50,52,82,84,88]. Existing results on this research direction can be classified into two main categories. (1) On the one hand, a real dataset is required to construct a learned model. Further, the prediction results of MPC are utilized to guide RL's exploration in the action space [52, 82, 84]. In [52], the action for the next time instant is obtained from the RL's policy with MPC's assistance, thus improving the convergence performance, compared with RL algorithms. (2) On the other hand, when the system dynamics are partially known, the accuracy of system dynamics can be improved by using outputs obtained from RL algorithms, and the action for the next time instant is calculated by solving the MPC optimization problem

[20, 44, 50, 88]. These methods can ensure the safety during the training process.

In many practical control problems, it is hard to obtain sufficient data for training, making it difficult for the first category in applications [82, 84]. To tackle this challenging problem, we follow [20] to develop the incorporation of RL and MPC. This work proposed a framework to combine RL and MPC for a class of linear systems with bounded disturbances. The framework utilizes the Q-learning algorithm to approximate system matrices of the dynamic model. Then, the MPC optimization problem is solved with the updated system matrices. Later in [44], the RTMPC framework [46] and the feedback control law were jointly considered to improve the convergence performance of system states. However, the main limitation of work [20, 44, 88] lies in the computational complexity since system matrices and the feedback control law are obtained from the RL algorithm at each time instant. Moreover, the related RPI set and the terminal region, which are necessary for ensuring the closed-loop performance, need to be determined online. Another limitation is that the utilized RL algorithm is not able to solve problems with a large action space, thus requiring the initial guess of system matrices to be close to true values. Otherwise, the control performance may not be able to achieve the desired performance.

In this chapter, we introduce a DRL scheme based on RTMPC principles for a class of constrained linear systems with bounded additive disturbances. Compared with the method presented in [44], we employ RTMPC to approximate the action-value function. Moreover, in [20, 44, 88], the utilized RL algorithms are traditional RL, i.e., Q-learning and SARSA, which may not be able to achieve learning objectives with continuous state and action spaces [48], and this limitation may obtain a policy with limited control performance. To tackle this problem, we use the SAC algorithm because of its strength in exploring the optimal policy for discrete-time systems with continuous states and action spaces. Compared with existing work, the

main contributions of our work are as follows.

1. By employing RTMPC to modify the target Q-value, under the proposed method, the learning policy can converge to the optimum with a reduction of conservatism. We also integrate four neural networks to handle the extensive data and exploration in the DRL.

2. A computationally tractable SAC-RTMPC framework is designed. In the proposed framework, the feedback control law, RPI set and terminal region are not required to update at each time instant. The control inputs (actions) are directly obtained from the learning policy, and safety can be ensured during the control process.

3. By storing predicted nominal trajectories obtained from RTMPC in a replay buffer, our proposed framework makes use of predicted trajectories, thus easing the requirement of a large number of samples.

The remainder of this chapter is structured as follows: Section 2.2 introduces the problem formulation and briefly presents the preliminary results of RTMPC and SAC algorithms. Section 2.3 discusses the relation between RL and MPC, and presents the proposed SAC-RTMPC algorithm. Section 2.4 shows the simulation and comparison studies. Lastly, Section 2.5 concludes this work.

Notations

The sets of real numbers, real column vectors with n components and real matrices with m rows and n columns are denoted by $\mathbb{R}, \mathbb{R}^n$ and $\mathbb{R}^{m\times n}$, respectively. Let $\|x\|$ and $\|x\|_\infty$ denote the vector Euclidean norm and infinity norm, respectively, and we define $\|x\|_{\boldsymbol{Q}} = x^\top Qx$. The Minkowski sum of sets $\mathbb{X} \in \mathbb{R}^n$ and $\mathbb{Y} \in \mathbb{R}^n$ is

denoted by $\mathbb{X} \oplus \mathbb{Y} = \{x + y | x \in \mathbb{X}, y \in \mathbb{Y}\}$, and the Pontryagin difference is $\mathbb{X} \ominus \mathbb{Y} = \{z \in \mathbb{R}^n : z + y \in \mathbb{X}; \forall y \in \mathbb{Y}\}$.

2.2 Preliminaries and Problem Formulation

Consider a discrete-time linear time-invariant (LTI) system with a bounded disturbance $\boldsymbol{\omega}_k \in \mathbb{R}^n$

$$\boldsymbol{x}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{B}\boldsymbol{u}_k + \boldsymbol{\omega}_k, \tag{2.1}$$

where $\boldsymbol{x}_k \in \mathbb{R}^n$ and $\boldsymbol{u}_k \in \mathbb{R}^m$ are the system state and the control input, respectively. The system is subject to constraints $\boldsymbol{x}_k \in \mathbb{X}$, $\boldsymbol{u}_k \in \mathbb{U}$ and $\boldsymbol{\omega}_k \in \mathbb{W}$, where sets $\mathbb{X}$, $\mathbb{U}$, $\mathbb{W}$ are compact and convex with the origin in its interior. We assume that system matrices $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ and $\boldsymbol{B} \in \mathbb{R}^{n \times m}$ are known. K is a feedback control gain and satisfies that $\boldsymbol{A}_k \equiv \boldsymbol{A} + \boldsymbol{B}K$ is stable. $\mathbb{Z}$ is the RPI set, which satisfies $\boldsymbol{A}_k \mathbb{Z} \oplus \mathbb{W} \subseteq \mathbb{Z}$ [9].

2.2.1 Rigid Tube-based MPC

We firstly review the RTMPC proposed in [46] with the associated optimization problem formulated as:

$$\min_{\boldsymbol{x}_{k|k},\boldsymbol{u}_k} J_N(\boldsymbol{x}_k,\boldsymbol{u}_k) = \sum_{l=k}^{k+N-1} \ell(\boldsymbol{x}_{l|k},\boldsymbol{u}_{l|k}) + \ell_f(\boldsymbol{x}_{k+N|k}), \tag{2.2a}$$

$$\text{s.t. } \boldsymbol{x}_{l+1|k} = \boldsymbol{A}\boldsymbol{x}_{l|k} + \boldsymbol{B}\boldsymbol{u}_{l|k}, \tag{2.2b}$$

$$\boldsymbol{x}_k \in \boldsymbol{x}_{k|k} \oplus \mathbb{Z}, \tag{2.2c}$$

$$\boldsymbol{x}_{l|k} \in \bar{\mathbb{X}} \equiv \mathbb{X} \ominus \mathbb{Z}, \tag{2.2d}$$

$$\boldsymbol{u}_{l|k} \in \bar{\mathbb{U}} \equiv \mathbb{U} \ominus K\mathbb{Z}, \tag{2.2e}$$

$$\boldsymbol{x}_{k+N|k} \in \mathbb{X}_f, \tag{2.2f}$$

where $\boldsymbol{x}_{l|k}$ and $\boldsymbol{u}_{l|k}$ denote the predicted nominal system state and control input l steps ahead from time k. $\boldsymbol{u}_k$ and $\boldsymbol{x}_k$ are the control sequence and state sequence, i.e., $\boldsymbol{u}_k = [\boldsymbol{u}_{k|k}^\top, \boldsymbol{u}_{k+1|k}^\top, \ldots, \boldsymbol{u}_{k+N-1|k}^\top]^\top$, respectively. N is the prediction horizon, and $\ell(\boldsymbol{x}_{l|k},\boldsymbol{u}_{l|k})$ and $\ell_f(\boldsymbol{x}_{k+N|k})$ denote the stage cost and terminal cost functions, respectively. $\mathbb{X}_f$ is the terminal constraint set. The optimal control sequence $\boldsymbol{u}_k^*$ is obtained by solving the optimization problem (2.2), then the first input in the sequence, $\boldsymbol{u}_k^* = \boldsymbol{u}_{k|k}^*$, is implemented to the system, i.e.,

$$\boldsymbol{u}_k = \boldsymbol{u}_k^* + K(\boldsymbol{x}_k - \boldsymbol{x}_{k|k}^*), \tag{2.3}$$

where $\boldsymbol{x}_{k|k}^*$ is the optimal decision variable of the problem (2.2). The system state $\boldsymbol{x}_{k+1}$ at next time instant is obtained from the dynamic system (2.1) under the control input (2.3).

To ensure robust constraint satisfaction, the RPI set should satisfy $\mathbb{Z} \subset$ interior $(\mathbb{X})$ and $\mathbb{K}\mathbb{Z} \subset$ interior $(\mathbb{U})$ [46]. The terminal cost function $\ell_f(\boldsymbol{x}_{k+N|k})$ and the

terminal constraint set $\mathbb{X}_f$ in (2.2) need to satisfy the following conditions [45]

$$\boldsymbol{A}_k\mathbb{X}_f \subset \mathbb{X}_f, \mathbb{X}_f \subset \mathbb{X} \ominus \mathbb{Z}, K\mathbb{X}_f \subset \mathbb{U} \ominus K\mathbb{Z}.$$
$$\ell_f(\boldsymbol{A}_k\boldsymbol{x}_{k+N|k}) + \ell(\boldsymbol{x}_{k+N|k}, u_{k+N|k}) \leqslant \ell_f(\boldsymbol{A}_k\boldsymbol{x}_{k+N|k}), \forall x \in \mathbb{X}_f.$$

2.2.2 Soft Actor-Critic (SAC)

Consider a discrete-time Markov Decision Process (MDP) characterized by a tuple $(\mathbb{X}, \mathbb{U}, \mathcal{R}, \lambda, \pi_\theta)$, where $\mathbb{X} \in \mathbb{R}^n$, $\mathbb{U} \in \mathbb{R}^m$ and $\mathcal{R}$ denote the state space, the action space and the reward function, respectively. $\lambda \in [0, 1]$ is the learning rate, and π_θ is the learning policy π with respect to the parameter θ. One of the fundamental principles to design RL algorithms is the Bellman equation expressed in the form of a state-action value function

$$Q(\boldsymbol{x}_k, \boldsymbol{u}_k) = r(\boldsymbol{x}_k, \boldsymbol{u}_k) + \gamma \sum_{\boldsymbol{x}_{k+1} \in \mathbb{X}} p(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k, \boldsymbol{u}_k)V(\boldsymbol{x}_{k+1}), \tag{2.5}$$

where $\gamma \in [0, 1]$ is the discount rate; $p(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k, \boldsymbol{u}_k)$ represents the probability for an agent moving from the state $\boldsymbol{x}_k$ to the next state $\boldsymbol{x}_{k+1}$ by taking the action $\boldsymbol{u}_k$, and the value function $V(\boldsymbol{x}_k)$ can be described as

$$V(\boldsymbol{x}_k) = r(\boldsymbol{x}_k) + \gamma \sum_{\boldsymbol{x}_{k+1} \in \mathbb{X}} p(\boldsymbol{x}_{k+1}|\boldsymbol{u}_k)V(\boldsymbol{x}_{k+1}). \tag{2.6}$$

The SAC [23] algorithm is the extension of the initial Actor-Critic (AC) [36] algorithm. SAC is built upon two key concepts: AC learning and maximum entropy reinforcement learning [90]. Combining these two concepts enables SAC to achieve stability and exploration in a high-dimensional continuous action space. In maximum entropy RL, the objective is to maximize the expected return with an additional

entropy term

$$H(\pi(\cdot|\boldsymbol{x}_k)) = \mathbb{E}_{\boldsymbol{u}_k \sim \pi(\cdot|\boldsymbol{x}_k)}[-\log \pi(\boldsymbol{u}_k|\boldsymbol{x}_k)].$$

The practical of entropy term is to regulate the balance between exploration and exploitation during training. A higher target entropy value encourages the algorithm to explore more optimal policy. Therefore, the objective function for maximum entropy RL can be written as

$$\pi^*_{\mathbf{MaxEnt}} = \arg\max_{\pi} \mathbb{E}_{\pi}\left[\sum_k r(\boldsymbol{x}_k, \boldsymbol{u}_k) + \xi H(\pi(\cdot|\boldsymbol{u}_k))\right],$$

where ξ is the temperature parameter. The parameter ξ can be adaptively learned by optimizing its own loss function given by

$$L(\xi) = \mathbb{E}_{\boldsymbol{x}_k \sim \beta, \boldsymbol{u}_k \sim \pi(\cdot|\boldsymbol{x}_k)}[-\xi(\log \pi(\boldsymbol{u}_k|\boldsymbol{x}_k) + H_0)], \tag{2.7}$$

where β denotes the replay buffer for past experience, and H_0 is the target entropy. The value function (2.6) in maximum entropy RL is rewritten as

$$V(\boldsymbol{x}_k) = \sum_{\boldsymbol{u}_k} \pi(\boldsymbol{u}_k|\boldsymbol{x}_k) Q(\boldsymbol{x}_k, \boldsymbol{u}_k) + H(\pi(\cdot|\boldsymbol{x}_k)).$$

It is noteworthy that SAC employs two critic neural networks with parameters η_1 and η_2, two target critic neural networks with parameters η_1^- and η_2^-, and one actor neural network with the parameter θ. When utilizing critic networks, the algorithm selects the one with a lower Q-value to alleviate the issue of overestimating Q-values. The TD error functions for critic neural networks and the actor neural network are

given by

$$\begin{aligned}\delta(\eta) =&\mathbb{E}_{\boldsymbol{x}_k,\boldsymbol{u}_k,r_k,\boldsymbol{x}_{k+1}\sim\beta,\boldsymbol{u}_{k+1}\sim\pi_\theta(\cdot|\boldsymbol{x}_{k+1})}[Q_\eta(\boldsymbol{x}_k,\boldsymbol{u}_k)\\&-(r_k+\gamma(\min_{j=1,2}Q_{\eta_j^-}(\boldsymbol{x}_{k+1},\boldsymbol{u}_{k+1})-\xi\log\pi(\boldsymbol{u}_{k+1}|\boldsymbol{x}_{k+1})))],\end{aligned}\tag{2.8a}$$

$$\delta(\theta) =\mathbb{E}_{\boldsymbol{x}_k\sim\beta,\boldsymbol{u}_k\sim\pi_\theta}[\xi\log(\pi_\theta(\boldsymbol{u}_k|\boldsymbol{x}_k))-Q_\eta(\boldsymbol{x}_k,\boldsymbol{u}_k)],\tag{2.8b}$$

The target parameter η^- can be updated by

$$\eta_{1,2}^-\leftarrow\tau\eta_{1,2}+(1-\tau)\eta_{1,2}^-,\tag{2.9}$$

where $\tau\in[0,1]$ is an updating rate.

2.2.3 Control Objectives

Our primary control objective is to regulate the system state of the LTI system in (2.1) to the origin as close as possible when $k\to\infty$. Meanwhile, the state and control input constraints must be satisfied for all admissible disturbances $\boldsymbol{\omega}_k$. Furthermore, the secondary objective is to minimize the value of the reward function to achieve the optimal learning policy. However, it is not necessary to satisfy the secondary objective as long as the system states are getting close to the origin when $k\to\infty$. Thus, in this section, our priority is to ensure that system states are approaching the origin while respecting system constraints.

2.3 Deep RL Rigid Tube-based MPC Algorithm

In this section, we present a DRL-RTMPC framework for the system in (2.2), based on SAC and RTMPC.

2.3.1 State-Action Value Function in MPC

Fundamentally, MPC optimizes system behaviours over a finite prediction horizon, while RL aims to maximize the cumulative rewards over an infinite prediction horizon. For (2.2a), we consider the stage cost function and terminal cost function in a quadratic form

$$\ell(\boldsymbol{x}_{l|k}, \boldsymbol{u}_{l|k}) = \sum_{l=k}^{k+N-1} (\|\boldsymbol{x}_{l|k}\|_{\boldsymbol{Q}} + \|\boldsymbol{u}_{l|k}\|_{\boldsymbol{R}}), \tag{2.10a}$$

$$\ell_f(\boldsymbol{x}_{k+N|k}) = \|\boldsymbol{x}_{k+N|k}\|_{\boldsymbol{P}}, \tag{2.10b}$$

where $\boldsymbol{Q}, \boldsymbol{R}$ and $\boldsymbol{P}$ are positive definite. The stage cost function (2.10a) can be decomposed into the predicted current and future stage costs

$$\ell(\boldsymbol{x}_{l|k}, \boldsymbol{u}_{l|k}) = (\|\boldsymbol{x}_{k|k}\|_{\boldsymbol{Q}} + \|\boldsymbol{u}_{k|k}\|_{\boldsymbol{R}}) + \sum_{l=k+1}^{k+N} (\|\boldsymbol{x}_{l|k}\|_{\boldsymbol{Q}} + \|\boldsymbol{u}_{l|k}\|_{\boldsymbol{R}}). \tag{2.11}$$

Then, the cost function of MPC optimization problem (2.2a) can be written as

$$\begin{aligned} J_N(\boldsymbol{x}_k, \boldsymbol{u}_k) =& (\|\boldsymbol{x}_{k|k}\|_{\boldsymbol{Q}} + \|\boldsymbol{u}_{k|k}\|_{\boldsymbol{R}}) + \sum_{l=k+1}^{k+N-1} \gamma^{l-k} (\|\boldsymbol{x}_{l|k}\|_{\boldsymbol{Q}} \\ &+ \|\boldsymbol{u}_{l|k}\|_{\boldsymbol{R}}) + \gamma^N \|\boldsymbol{x}_{k+N|k}\|_{\boldsymbol{P}}. \end{aligned} \tag{2.12}$$

By comparing (2.5) and (2.12), if we assume the discount rate $\gamma = 1$, and the control input (action) $\boldsymbol{u}_k$ is obtained from the MPC policy, it is evident to show that (2.12) becomes a special case of (2.5). The current reward function $r(\boldsymbol{x}_k, \boldsymbol{u}_k)$, probability for taking control input $\boldsymbol{u}_k^*$ at state $\boldsymbol{x}_k$, and the learning policy $\pi(\boldsymbol{x}_k)$ can

be defined as follows

$$r(\boldsymbol{x}_k, \boldsymbol{u}_k) = (\|\boldsymbol{x}_{k|k}\|_{\boldsymbol{Q}} + \|\boldsymbol{u}_{k|k}\|_{\boldsymbol{R}}), \tag{2.13a}$$

$$p(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k, \boldsymbol{u}_k) = 1, \tag{2.13b}$$

$$\pi(\boldsymbol{x}_k) = \boldsymbol{u}_k^*. \tag{2.13c}$$

Next, we can define the Bellman optimality equation by using the policy of MPC

$$\begin{aligned} Q(\boldsymbol{x}_{l|k}, \boldsymbol{u}_{l|k}) = \min_{\boldsymbol{x}_k, \boldsymbol{u}_k} (\|x_{k|k}\|_{\boldsymbol{Q}} + \|\boldsymbol{u}_{k|k}\|_{\boldsymbol{R}}) + \sum_{l=k+1}^{k+N-1} \gamma^{l-k} (\|\boldsymbol{x}_{l|k}\|_{\boldsymbol{Q}} + \|\boldsymbol{u}_{l|k}\|_{\boldsymbol{R}}) \\ + \gamma^N \|\boldsymbol{x}_{k+N|k}\|_{\boldsymbol{P}}, \\ \text{s.t.} (2.2\text{b}) - (2.2\text{f}). \end{aligned}$$

2.3.2 Temporal Difference from MPC to RL

In Section 2.3.1, we have identified the connection between MPC and the Bellman optimality equation. Furthermore, for the model-free RL-based MPC, the TD error is essential to be determined. This subsection presents how to obtain the TD error using MPC. As mentioned in Section 2.2.2, the TD algorithm initially requires the specification of the TD target. Typically, the TD target is obtained by adding the current reward $r(\boldsymbol{x}_k, \boldsymbol{u}_k)$ to the state-action value of the next time instant $Q(\boldsymbol{x}_{k+1}, \boldsymbol{u}_{k+1})$. Consequently, the TD error can be derived via subtracting the $Q(\boldsymbol{x}_{l|k}, \boldsymbol{u}_{l|k})$ value of the current policy from the TD target. In our approach, we substitute the sequence of predictive states, from the MPC optimization problem, into the TD error; thus,

(1.7) can be rewritten as

$$\delta = Q(\boldsymbol{x}_{l|k}, \boldsymbol{u}_{l|k}) - r(\boldsymbol{x}_k, \boldsymbol{u}_k) + \gamma \sum_{\boldsymbol{x}_{k+1} \in \mathbb{X}} p(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k, \boldsymbol{u}_k) V(\boldsymbol{x}_{k+1}). \tag{2.15}$$

In (2.15), $Q(\boldsymbol{x}_{l|k}, \boldsymbol{u}_{l|k})$ is obtained by solving the MPC optimization problem, and the rest part $r(\boldsymbol{x}_k, \boldsymbol{u}_k) + \gamma \sum_{\boldsymbol{x}_{k+1} \in \mathbb{X}} p(\boldsymbol{x}_{k+1}|\boldsymbol{x}_k, \boldsymbol{u}_k) V(\boldsymbol{x}_{k+1})$ can be predicted by the current policy of the Q-value estimator/DNN (with policy parameter η) by (2.5). Therefore, the TD error function (2.15) can be rewritten as

$$\begin{aligned} \delta =& Q(\boldsymbol{x}_{l|k}, \boldsymbol{u}_{l|k}) - Q_\eta(\boldsymbol{x}_k, \boldsymbol{u}_k) \\ =& (\|\boldsymbol{x}_k\|_{\boldsymbol{Q}} + \|\boldsymbol{u}_k\|_{\boldsymbol{R}}) + \sum_{l=k+1}^{k+N-1} \lambda^{l-k} (\|\boldsymbol{x}_{l|k}\|_{\boldsymbol{Q}} \\ & + \|\boldsymbol{u}_{l|k}\|_{\boldsymbol{R}}) + \lambda^N \|\boldsymbol{x}_{k+N|k}\|_{\boldsymbol{P}} - Q_\eta(\boldsymbol{x}_k, \boldsymbol{u}_k). \end{aligned} \tag{2.16}$$

2.3.3 SAC Rigid Tube-based MPC

As mentioned in Section 2.2.2, the SAC algorithm involves five neural networks: two critic neural networks, two target critic neural networks, and one actor neural network. In the proposed methodology, we use two critics neural networks with the parameters η_1 and η_2, one target critic neural network with the policy parameter η^-, and one action neural network with the policy parameter θ from the original SAC framework, and the optimal value of the MPC objective function serves as another target critic factor. Thus, we utilize both the target critic neural network and the optimal value of the MPC objective function as estimators of target critic neural networks to have a trade-off between the stability and exploration. Then, by comparing state-action values $Q(x, u)$ from the aforementioned estimators, the relatively small value of $Q(x, u)$ is employed to guide the critic neural network to learn a policy $\pi_\theta(\boldsymbol{x}_k)$. Specifically, if the $Q(\boldsymbol{x}_{l|k}, \boldsymbol{u}_{l|k})$, from the MPC, is relatively

Algorithm 1 SAC-RTMPC

Input: Learning rate λ, discount rate γ, target entropy H_0, and updating rate τ; MPC parameters $\boldsymbol{Q}, R\boldsymbol{R}, \boldsymbol{A}, \boldsymbol{B}$ and N. Initialize the policy parameters $\xi, \eta_1, \eta_2, \eta^-, \theta$

for time instant $k = 0, 1, 2, \ldots, n$ **do**
  Measure the system state $\boldsymbol{x}_k$; then, solve the MPC problem (2.2) and use (2.3) to obtain predicted sequences $\boldsymbol{x}_k, \boldsymbol{u}_k$
  Store $\boldsymbol{x}_k, \boldsymbol{u}_k$ to the replay buffer β
  Compute the TD target from modified MPC cost function (2.12)
  for numbers of training **do**
    Sample T data pairs from β, and obtain the TD target from target critic network by using (2.5).
    Determine the TD error of critic networks $\delta(\eta)$ by using (2.17); then, update the policy parameter $\eta_{1,2}$
    Based on the current policy parameters $\eta_{1,2}$ and θ, calculate the TD error $\delta(\theta)$ by using (2.8b), and update the policy parameter θ,
    Update the target policy parameter $\eta^-_{1,2}$ from (2.9) and the temperature parameter ξ by optimizing (2.7)
  end for
  Obtain the control input from the actor neural network, i.e., $\boldsymbol{u}_k = \pi_\theta(\boldsymbol{x}_k)$
  Implement $\boldsymbol{u}_k$ to the system in (2.1) to generate a new system state $\boldsymbol{x}_{k+1}$
end for

smaller, our algorithm treats $Q(\boldsymbol{x}_{l|k}, \boldsymbol{u}_{l|k})$ as the TD target and updates the policy parameter $\eta_{1,2}$ of the critic neural networks. $Q_{\eta_{1,2}}(\boldsymbol{x}_k, \boldsymbol{u}_k)$ is then used to amend the policy parameter θ of the action neural network. Stability can be ensured during the training process. If the $Q_{\eta^-}(\boldsymbol{x}_k, \boldsymbol{u}_k)$ derived from the target critic neural network is found to be relatively smaller, $Q_{\eta^-}(\boldsymbol{x}_k, \boldsymbol{u}_k)$ will serve as the TD target and repeat the steps as mentioned above. Therefore, we need to redefine the TD error function (2.8a) and (2.16)

$$\begin{aligned}\delta(\eta) =& \mathbb{E}_{\boldsymbol{x}_k, \boldsymbol{u}_k, r_k, \boldsymbol{x}_{k+1} \sim \beta, \boldsymbol{u}_{k+1} \sim \pi_\theta(\cdot|\boldsymbol{x}_{k+1})}[Q_\eta(\boldsymbol{x}_k, \boldsymbol{u}_k) - \min(Q(\boldsymbol{x}_{l|k}, \boldsymbol{u}_{l|k}), \\ & (r_k + \gamma(Q_{\eta^-}(\boldsymbol{x}_{k+1}, \boldsymbol{u}_{k+1})) - \gamma\xi \log \pi(\boldsymbol{u}_{k+1}|\boldsymbol{x}_{k+1})))].\end{aligned} \tag{2.17}$$

The essential idea of the proposed method is to exploit the predicted results of optimization problems (2.2)-(2.3) and to modify the TD error function of critic neu-

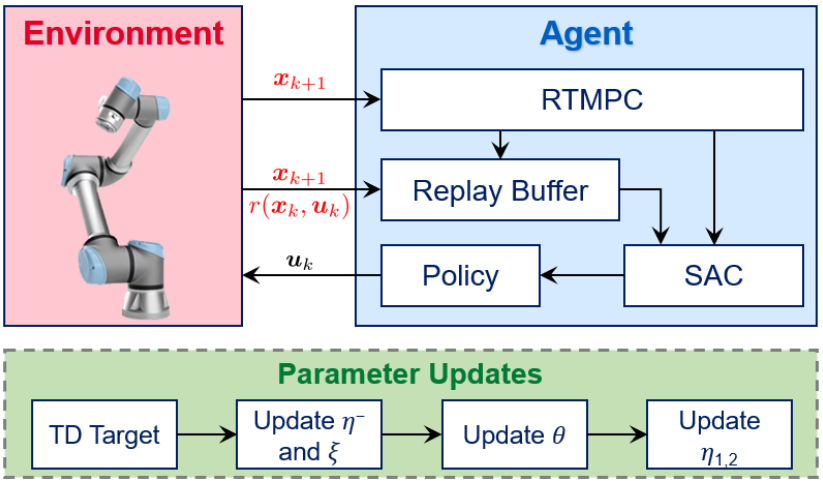

Figure 2.1: Scheme of the proposed SAC-RTMPC framework

ral networks by using (2.17). To be more specific, we firstly solve MPC problems (2.2) and (2.3). Then, the results $\boldsymbol{x}_k$ and $\boldsymbol{u}_k$ are reconstructed in the form of $\{\boldsymbol{x}_{l|k}, \boldsymbol{u}_{l|k}, r_{l|k}, \boldsymbol{x}_{l+1|k}\}$, $l = 0, 1, 2, \ldots, 9$, and stored in the replay buffer β. Then, by repeatedly sampling T data pairs, the policy parameters are updated. Lastly, the control input is obtained from the updated DRL policy, i.e., $\boldsymbol{u}_k = \pi_\theta(\boldsymbol{x}_k)$. In the next time instant, we repeat the aforementioned procedures to calculate the new control input. Overall, the proposed method is summarized in Algorithm 1, and the scheme of the proposed SAC-RTMPC framework is shown in Figure 2.1.

2.4 Simulation Results

In this section, a numerical sample is given to show the effectiveness of the proposed framework as detailed in Algorithm 1. The numerical simulation is conducted in Python. We consider the following LTI system in the test [46]

$$\boldsymbol{x}_{k+1} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \boldsymbol{x}_k + \begin{bmatrix} 0.5 \\ 1 \end{bmatrix} \boldsymbol{u}_k + \boldsymbol{\omega}_k. \tag{2.18}$$

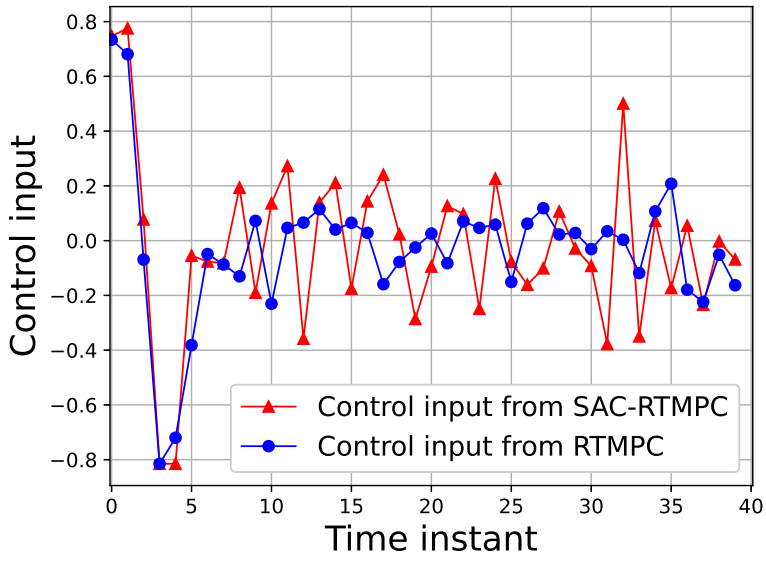

Figure 2.2: Comparison of the trajectories of control input $\boldsymbol{u}_k$ for time instants $k = 0, \ldots, 39$ under SAC-RTMPC and RTMPC methods

The state constraint, the control input constraint and the bounded disturbance are listed as follows:

$$\boldsymbol{x} \in \mathbb{X} \triangleq \left\{ \boldsymbol{x} \mid \begin{bmatrix} -10 \\ -2 \end{bmatrix} \leqslant \boldsymbol{x} \leqslant \begin{bmatrix} 2 \\ 2 \end{bmatrix} \right\},$$

$$\boldsymbol{u} \in \mathbb{U} \triangleq \{\boldsymbol{u} \mid |\boldsymbol{u}| \leqslant 1\}, \ \boldsymbol{\omega} \in \mathbb{W} \triangleq \{\|\boldsymbol{\omega}\|_\infty \leqslant 0.15\}.$$

The weighting matrices are $\boldsymbol{Q} = I_2$ and $\boldsymbol{R} = 0.1$. The prediction horizon is $N = 10$. For the SAC, the actor neural network is built up by a single hidden layer with the Softplus activation function, while the critic neural network contains two hidden layers with ReLU activation functions. Each hidden layer consists of 120 neurons. The learning rate γ, the temperature parameter ξ, the target entropy H_0, and the updating rate τ are selected as 0.98, $\log(0.01)$, -1 and 0.001, respectively. The initial state is set as $\boldsymbol{x_0} = \begin{bmatrix} -7 & 0.5 \end{bmatrix}^\top$.

We compare the proposed framework summarized in Algorithm 1 with the RTMPC

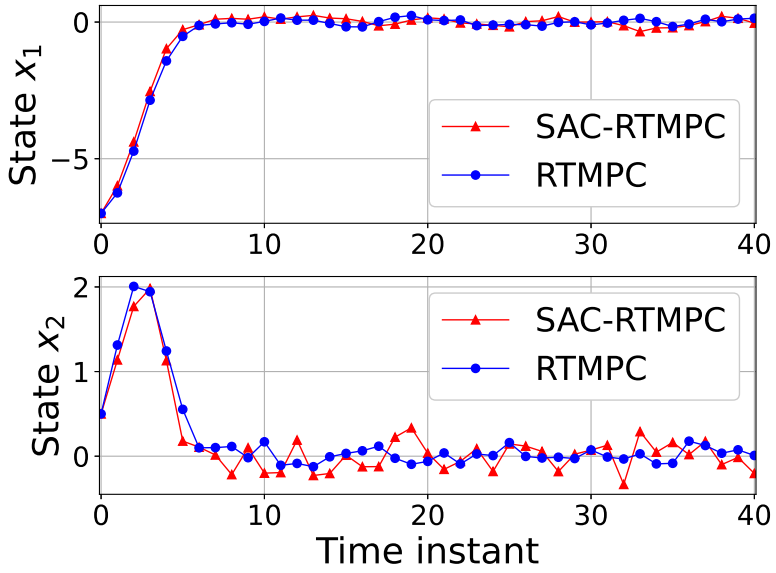

Figure 2.3: Comparison of the trajectories of state $\boldsymbol{x}_k$ for time instants $k = 0, \ldots, 40$ under SAC-RTMPC and RTMPC methods

method in [46]. Figures 2.2-2.3 show the trajectories of control inputs and system

Table 2.1: The comparison of system performance

	Time instants	***Algorithm 1***	***RTMPC***
J	10	121.86	130.50
	40	123.47	133.33

states, which are obtained by Algorithm 1 and RTMPC [46]. Figure 2.2 shows that the trajectories of control inputs from our method satisfy input constraints. Meanwhile, by implementing control inputs to the system in (2.1), our algorithm exhibits better control performance of convergence. The reason for this improvement is that the value of the reward function (2.13a) is more sensitive to changes in system states. If the current reward is too large, Algorithm 1 explores a more appropriate control input, which can heavily reduce state cost for the next time instant. Furthermore, Figure 2.3 illustrates that state constraints are respected during the control process. In addition, we introduce an index $J = \sum_{k=0}^{T}(\|\boldsymbol{x}_k\|_{\boldsymbol{Q}} + \|\boldsymbol{u}_k\|_{\boldsymbol{R}})$, where T denotes the simulation

time. The comparison is shown in Table 2.1. We observe that the total cost J, from our proposed algorithm, is lower than that of the tube MPC. To summarize, while satisfying safety concerns, the proposed framework can improve the performance of the convergence and total cost compared with the RTMPC method [46].

2.5 Conclusion

In this chapter, we proposed a DRL-MPC framework that incorporates SAC and RTMPC. Unlike most DRL algorithms requiring a large number of real samples, in our approach, RTMPC is employed as an estimator to predict future system states and control inputs. The predicted sequence is stored in the replay buffer, which facilitates the SAC algorithm to determine an optimal policy online. Thus, the performance of convergence is improved in the proposed framework. Moreover, by deploying MPC, the proposed framework can ensure the safety during the training, which is hard to achieve in DRL algorithms. Numerical simulations have demonstrated the effectiveness of the proposed approach compared with the RTMPC scheme.

Chapter 3

Incorporation of MPC and DRL for Manipulator Systems

3.1 Introduction

In recent decades, robot manipulators have received considerable attention due to their wide applications in different areas, such as surgery assistance [12, 55] and industrial manufacturing [1, 2]. Robot manipulators are often applied to solve control problems with complex, high-dimensional, and dynamic environments, requiring real-time decision-making, precise control, and robustness to uncertainties. The essential control objectives are to achieve a reference point and track a trajectory by the end-effector. Many results have been proposed to control robot manipulators, such as PID control, MPC, and RL. Traditional control methods, such as PID control [14,62], struggle to achieve optimal performance for a highly nonlinear manipulator system with disturbances. Therefore, many robust control methods have been proposed for manipulator systems with high nonlinearities subject to external disturbances. In [43], a self-tuning fuzzy PID controller was designed for a 4-degree-of-freedom (4-

DOF) manipulator system with a bounded disturbance. By implementing an adaptive neural network, the work in [24, 79] showed a good performance, compared with PD control, while dealing with a disturbed 2-DOF manipulator system. However, these approaches can not adequately satisfy the system constraints.

Due to the above limitations, MPC is proposed to handle system constraints and disturbances simultaneously [71]. By giving explicit manipulator dynamics, MPC constructs and solves an optimization problem to determine a sequence of optimal control inputs with the satisfaction of system constraints. Many efforts have been made to control robot manipulators by using MPC [17, 33, 56, 61]. The first result on this research direction can be found in [61], which solved the MPC optimization problem with an approximated linear model of a KUKA robot. Later, the work in [56] proposed a nonlinear MPC controller for a 2-DOF manipulator system. Recently, a robust MPC method was proposed in [17], which not only addressed the trajectory tracking problem for a Baxter robot in the presence of disturbances, but also guaranteed the satisfaction of system constraints. However, these results only considered a 2-DOF manipulator system for simulations and experiments. Although MPC shows better control performance compared to the traditional control methods, the limitations of MPC are requirement of the high computational burden and explicit system dynamics, notably for high-dimensional systems.

Fortunately, model-free DRL offers a framework that enables the robot manipulator to perform complex and high-dimensional tasks without the requirement of precise system dynamics. Many model-free DRL methods have been proposed for manipulators, involving the work in [21, 27, 31, 40, 54, 65, 89]. In [89], a model-free RL algorithm was utilized to optimize the policy of the proposed algorithm for a 6-DOF collaborative manipulator. Based on the normalized advanced functions (NAF) in [22], the authors in [21] proposed a DRL method for a 6-DOF Kinova JACO arm

with additional 3-DOF fingers. It is noted that the aforementioned model-free RL algorithms require a large number of samples to obtain the optimal policy. However, this requirement is usually difficult to achieve in real environment. Thus, it is significant to improve the sample efficiency during training the DRL policy.

Recently, MPC-based DRL has emerged as a promising approach for improving the sample efficiency due to its ability to predict future actions. Unlike the model-free DRL algorithm, MPC-based DRL algorithms learn system dynamics from real samples. By optimizing the policy with real samples and predictive rollouts, the sample efficiency can be significantly improved. Compared with the conventional MPC, MPC-based DRL incorporates model-free DRL and sampling-based MPC. The MPC-based DRL algorithms were developed and utilized to control robot manipulators in [42, 51, 63, 66]. In these results, the evaluation of control performance and computational burden is often neglected, and only the performance of total return with different algorithms is compared. Moreover, in practical applications, it is challenging to obtain sufficient samples for training, due to mechanical limits, system constraints, and safety specifications. In this chapter, we focus on the simulation studies of applying different model-free RL algorithms, MPC-based RL algorithms, and a nonlinear MPC (NMPC) method to robot manipulators. Comprehensive comparisons will show different features of these representative algorithms. Specifically, we will compare the computational burden, control performance, and training performance of manipulators by using different algorithms, including NMPC [15], two model-free DRL algorithms SAC [23] and DDPG [41], and an MPC-based DRL algorithm MBPO [29]. Three case studies will be test and evaluated. Each case study considers different control objectives and manipulator system dynamics.

The remainder of this chapter is structured as follows: Section 3.2 presents the problem formulation. Section 3.3 firstly introduces the preliminary results of NMPC,

Figure 3.1: Universal Robots UR10e manipulator [1]

DDPG, and MBPO algorithms; then discusses the simulation and comparison studies for a regulation problem of a 2-DOF manipulator based on its dynamics model. Section 3.4 shows simulation results for a tracking problem of 2-DOF manipulator using its kinematics model. Section 3.5 focuses on the comparison results of training performance for a regulation problem of UR10e for its kinematics model.

3.2 Modeling of Robot Manipulators

In this section, we first introduce the kinematics and dynamics model of a 2-DOF planar elbow manipulator. Then, the kinematics of the Universal Robots UR10e industrial collaborative robot (shown in Figure 3.1) is presented. The kinematics equations of manipulators represent the relationship between the angles of each joint

[1] https://www.universal-robots.com/products/ur10-robot/

Link	$\theta_i[rad]$	$\alpha_i[rad]$	$d_i[m]$	$a_i[m]$
1	θ_1	$\pi/2$	0	0.55
2	θ_2	0	0	0.3

Table 3.1: The D-H parameters of 2-DOF planar elbow manipulator

and the end-effector's position and orientation. Given the joint angles, the position and orientation of the end-effector can be obtained by solving the forward kinematics. In contrast, joint angles can be determined by solving the inverse kinematics utilizing the reference position and orientation of the end-effector. The dynamic equations describe the relationship between the motion and force.

3.2.1 Kinematics and Dynamics of Planar Elbow Manipulator

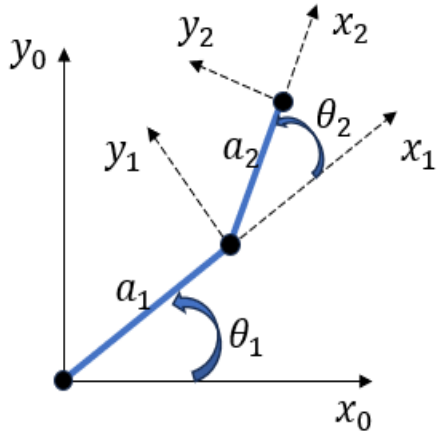

Figure 3.2: 2-DOF planar manipulator

Kinematics

In order to derive kinematics of the manipulator, the 4×4 homogeneous transformation matrix of each joint angle needs to be determined in advance. The homogeneous

matrix T_i can be described as follows [75]:

$$T_i = \begin{bmatrix} c\theta_i & -s\theta_i c\alpha_i & s\theta_i s\alpha_i & a_i c\theta_i \\ s\theta_i & c\theta_i c\alpha_i & -c\theta_i s\alpha_i & a_i s\theta_i \\ 0 & s\alpha_i & c\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3.1}$$

where $\theta_i, \alpha_i, d_i, a_i$ denote the link angle, the link twist, the link offset, and the link length associated with link i, respectively. $c\theta_i$ and $s\theta_i$ denote $\cos\theta_i$ and $\sin\theta_i$. The transformation matrix ${}^0_iH$ is obtained as follows:

$${}^0_iH = T_1 \dots T_i. \tag{3.2}$$

By giving the Denavit–Hartenberg (D-H) coordinates frame of the 2-DOF planar elbow manipulator in Figure 3.2, the D-H parameters can be established in Table 3.1. Therefore, the total transformation matrix ${}^0_2H$ can be calculated by using (3.1) and (3.2)

$${}^0_2H = \begin{bmatrix} c\theta_{12} & -s\theta_{12} & 0 & d_x \\ s\theta_{12} & c\theta_{12} & 0 & d_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3.3}$$

where $\theta_{12} = \theta_1 + \theta_2$. d_x and d_y denote the end-effector's position, which are given by

$$\begin{bmatrix} d_x \\ d_y \end{bmatrix} = \begin{bmatrix} a_1 c\theta_1 + a_2 c\theta_{12} \\ a_1 s\theta_1 + a_2 s\theta_{12} \end{bmatrix}.$$

The transformation of the inverse kinematics problem for the 2-DOF planar elbow manipulator system is straightforward to be shown as follows:

$$\theta_2 = \text{atan2}\left(\frac{d_x^2 + d_y^2 - a_1^2 - a_2^2}{2a_1a_2}, \pm\sqrt{1 - \left(\frac{d_x^2 + d_y^2 - a_1^2 - a_2^2}{2a_1a_2}\right)^2}\right), \tag{3.4}$$

$$\theta_1 = \text{atan2}\left(d_y, d_x\right) - \text{atan2}\left(a_1 + a_2c\theta_2, a_2s\theta_2\right),$$

where atan2 is the four-quadrant inverse tangent function.

Dynamics

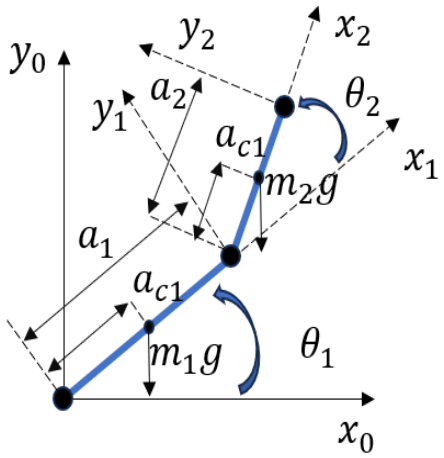

Figure 3.3: Dynamics model of 2-DOF planar elbow manipulator

Link	1	2
$\theta_i[rad]$	θ_1	θ_2
$m_i[kg]$	3.8749	1.0651
$a_{ci}[m]$	0.2110	0.1410
$a_i[m]$	0.55	0.3
$I_i[kg \cdot m^2]$	0.1395	0.0401

Table 3.2: Parameters of dynamics modeling of 2-DOF planar elbow manipulator

By considering the 2-DOF planar elbow manipulation shown in Figure 3.3, we obtain the Euler-Lagrange equation [75] as

$$\mathbf{M}(\boldsymbol{\theta})\ddot{\boldsymbol{\theta}} + \mathbf{C}(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}})\dot{\boldsymbol{\theta}} + \mathbf{v}(\boldsymbol{\theta}) = \boldsymbol{\tau}, \tag{3.5}$$

where $\boldsymbol{\theta} = [\theta_1, \theta_2]^\top$, $\dot{\boldsymbol{\theta}} = [\dot{\theta_1}, \dot{\theta_2}]^\top$ and $\ddot{\boldsymbol{\theta}} = [\ddot{\theta_1}, \ddot{\theta_2}]^\top$ are the joint angles, the angular velocities, and the angular accelerations, respectively. $\boldsymbol{\tau} = [\tau_1, \tau_2]^\top$ denotes the applied force. $\mathbf{M}(\boldsymbol{\theta}) \in \mathbb{R}^{2\times 2}$ is the mass matrix. $\mathbf{C}(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}) \in \mathbb{R}^{2\times 2}$ is the matrix of Coriolis and centripetal terms. $\mathbf{v}(\boldsymbol{\theta}) \in \mathbb{R}^2$ denotes the gravitational vector. From [75], we obtain the $\mathbf{M}(\boldsymbol{\theta}), \mathbf{C}(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}})$, and $\mathbf{v}(\boldsymbol{\theta})$ as follows:

$$\mathbf{M}(\boldsymbol{\theta}) = \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix}, \mathbf{C}(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}) = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}, \mathbf{v}(\boldsymbol{\theta}) = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}, \tag{3.6}$$

where

$$\begin{aligned}
m_{11} &= m_1 a_{c1}^2 + m_2(a_1^2 + a_{c2}^2 + 2a_1 a_{c2}^2 + 2a_1 a_{c2} c\theta_2) + I_1 + I_2, \\
m_{12} &= m_{21} = m_2(a_{c2}^2 + a_1 a_{c2} c\theta_2) + I_2, \\
m_{22} &= m_2 a_{c2}^2 + I_2, \\
c_{11} &= -m_2 a_1 a_{c2} s\theta_2 \dot{\theta_2}, \\
c_{12} &= -m_2 a_1 a_{c2} s\theta_2 (\dot{\theta_1} + \dot{\theta_2}), \\
c_{21} &= m_2 a_1 a_{c2} s\theta_2 \dot{\theta_1}, \\
c_{22} &= 0, \\
v_1 &= (m_1 a_{c1} + m_2 a_1) g c\theta_1 + m_2 a_{c2} g c(\theta_1 + \theta_2), \\
v_2 &= m_2 a_{c2} c(\theta_1 + \theta_2).
\end{aligned}$$

As aforementioned, the dynamics equations of the manipulator explicitly represent the relation between the motion and force. However, it is difficult to obtain the Euler-Lagrange equation for UR10e. Thus, we only describe kinematics of UR10e in the next section.

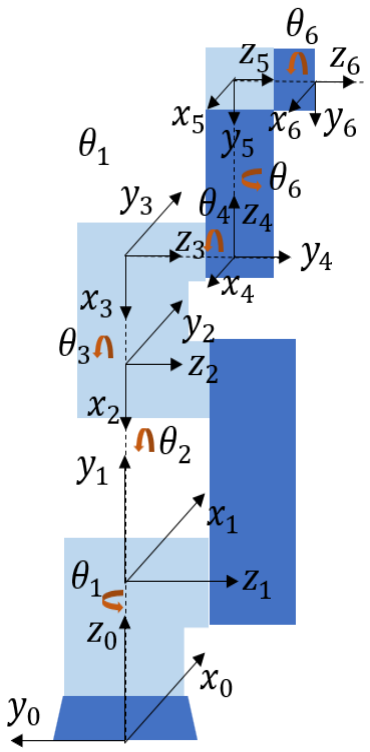

Figure 3.4: D-H coordinate frame for UR10e

3.2.2 Kinematics of UR10e

Forward Kinematics

For the UR10e manipulator, the joint variable is θ_i. The other three parameters α_i, d_i, a_i are the D-H parameters that are based on the coordinate frame in Figure 3.4 and described in Table 3.3.

Link	$\theta_i[rad]$	$\alpha_i[rad]$	$d_i[m]$	$a_i[m]$
1	θ_1	$\pi/2$	0.1807	0
2	θ_2	0	0	-0.6127
3	θ_3	0	0	-0.5716
4	θ_4	$\pi/2$	0.1742	0
5	θ_5	$-\pi/2$	0.1199	0
6	θ_6	0	0.1166	0

Table 3.3: D-H parameters of UR10e

Thus, the transformation matrix T_i can be computed as

$$T_1 = \begin{bmatrix} c\theta_1 & 0 & s\theta_1 & 0 \\ s\theta_1 & 0 & -c\theta_1 & 0 \\ 0 & 1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \; T_2 = \begin{bmatrix} c\theta_2 & -s\theta_2 & 0 & a_2 c\theta_2 \\ s\theta_2 & c\theta_2 & 0 & a_2 s\theta_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \; T_3 = \begin{bmatrix} c\theta_3 & -s\theta_3 & 0 & a_3 c\theta_3 \\ s\theta_3 & c\theta_3 & 0 & a_3 s\theta_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$T_4 = \begin{bmatrix} c\theta_4 & 0 & s\theta_4 & 0 \\ s\theta_4 & 0 & -c\theta_4 & 0 \\ 0 & 1 & 0 & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad T_5 = \begin{bmatrix} c\theta_5 & 0 & -s\theta_5 & 0 \\ s\theta_5 & 0 & c\theta_5 & 0 \\ 0 & -1 & 0 & d_5 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad T_6 = \begin{bmatrix} c\theta_6 & -s\theta_6 & 0 & 0 \\ s\theta_6 & c\theta_6 & 0 & 0 \\ 0 & 0 & 1 & d_6 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

The following total transformation matrix ${}^0_6H$ is composed of the rotation matrix R and the end-effector position $d = [d_x, d_y, d_z]^\top$.

$${}^0_6H = T_1 T_2 T_3 T_4 T_5 T_6 = \begin{bmatrix} R^0_6 & d \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & d_x \\ r_{21} & r_{22} & r_{23} & d_x \\ r_{31} & r_{32} & r_{33} & d_x \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3.7}$$

where

$$
\begin{aligned}
r_{11} &= -s\theta_1 s\theta_5 c\theta_6 + c\theta_1(-s\theta_{234}s\theta_6),\\
r_{12} &= s\theta_1 s\theta_5 s\theta_6 - c\theta_1(s\theta_{234}c\theta_6 = c\theta_{234}c\theta_5 s\theta_6),\\
r_{13} &= s\theta_1 c\theta_5 + c\theta_1 c\theta_{234}s\theta_5,\\
r_{21} &= c\theta_1 s\theta_5 s\theta_6 + s\theta_1(-s\theta_{234}s\theta_6 + c\theta_{234}c\theta_5 c\theta_6),\\
r_{22} &= -c\theta_1 s\theta_5 s\theta_6 - s\theta_1(s\theta_{234}c\theta_6 + c\theta_{234}c\theta_5 s\theta_6),\\
r_{23} &= -c\theta_1 c\theta_5 + s\theta_1 c\theta_{234}s\theta_5,\\
r_{31} &= c\theta_{234}s\theta_6 + s\theta_{234}c\theta_5 c\theta_6,\\
r_{32} &= c\theta_{234}c\theta_6 - s\theta_{234}c\theta_5 s\theta_6,\\
r_{33} &= s\theta_{234}s\theta_5,\\
d_x &= d_6(s\theta_1 c\theta_5 + c\theta_1 c\theta_{234}s\theta_5) + d_4 s\theta_1 - c\theta_1(a_2\theta_2 + a_3 s\theta_{23} + d_5 s\theta_{234}),\\
d_y &= d_6(-c\theta_1 c\theta_5 + s\theta_1 c\theta_{234}s\theta_5) - d_4 c\theta_1 - s\theta_1(a_2\theta_2 + a_3 s\theta_{23} + d_5 s\theta_{234}),\\
d_z &= d_1 + d_6(s\theta_{234}s\theta_5) + a_2 c\theta_2 + a_3 c\theta_{23} + d_5\theta_{234}.
\end{aligned}
$$

The position of end-effector d_x, d_y, d_z can be obtained by solving the forward kinematics problem (3.7).

Inverse Kinematics

Unlike the forward kinematics problem, the inverse kinematics problem derives the angle of each joint with the target position [d_x, d_y, d_z] and the orientation [O_ϕ, O_θ, O_ψ]. For a 6-DOF manipulator, the joint variables $\theta_1 \ldots \theta_6$ are difficult to be directly obtained by using the relationships outlined in (3.7). In this subsection, we will use the geometric approach and the inverse orientation to address the positioning and the orientation problem, respectively.

From Figure 3.4, we can see that the UR10e has a spherical wrist, which implies

that the axes z_3, z_4, z_5 intersect at a point. Thus, the coordinates of the wrist center x_c, y_c, z_c are calculated by

$$\begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = \begin{bmatrix} d_x - d_6 r_1 3 \\ d_y - d_6 r_2 3 \\ d_z - d_6 r_3 3 \end{bmatrix}. \tag{3.8}$$

Now, the geometric approach can be utilized to determine the first joint angles $\theta_1, \theta_2, \theta_3$

$$\begin{aligned} \theta_1 &= \operatorname{atan2}(x_c, y_c), \\ \theta_3 &= \operatorname{atan2}\left(\frac{x_c^2 + y_c^2 - d_4^2 + z_c^2 - a_2^2 - a_3^2}{2a_2 a_3}, \right. \\ &\qquad \left. \pm\sqrt{1 - \left(\frac{x_c^2 + y_c^2 - d_4^2 + z_c^2 - a_2^2 - a_3^2}{2a_2 a_3} \right)^2} \right), \\ \theta_2 &= \operatorname{atan2}\left(\sqrt{x_c^2 + y_c^2 - d_4^2}, z_c \right) - \operatorname{atan2}\left(a_2 + a_3 c\theta_3, a_3 s\theta_3 \right). \end{aligned} \tag{3.9}$$

To find $\theta_4, \theta_5, \theta_6$, the inverse orientation problem needs to be tackled. From (3.7), we can obtain the rotation matrix R_3^0, R_6^3 from $H_3^0 = T_1 T_2 T_3$ and $H_6^3 = T_4 T_5 T_6$, respectively. Consequently, we can derive the equations for $\theta_4, \theta_5, \theta_6$ as follows:

$$R_6^3 = (R_3^0)^\top R_6^0, \tag{3.10}$$

where

$$R_6^3 = \begin{bmatrix} c\theta_4 c\theta_5 c\theta_6 - s\theta_4 s\theta_6 & -s\theta_4 c\theta_6 - c\theta_4 c\theta_5 s\theta_6 & -c\theta_4 s\theta_5 \\ c\theta_4 s\theta_6 + c\theta_5 c\theta_6 s\theta_4 & c\theta_4 c\theta_6 - s\theta_4 c\theta_5 s\theta_6 & -s\theta_4 s\theta_5 \\ c\theta_6 s\theta_5 & -s\theta_5 s\theta_6 & c\theta_5 \end{bmatrix},$$

$$R_3^0 = \begin{bmatrix} c\theta_1 c\theta_{23} & -c\theta_1 c\theta_2 c\theta_3 - c\theta_1 s\theta_2 c\theta_3 & s\theta_1 \\ s\theta_1 c\theta_{23} & -s\theta_1 s\theta_{23} & -c\theta_1 \\ s\theta_{23} & c\theta_{23} & 0 \end{bmatrix}.$$

Therefore, $\theta_4, \theta_5, \theta_6$ can be calculated by

$$\begin{aligned}
\theta_4 &= \text{atan2}(r_{33}s\theta_{23} + r_{13}c\theta_1 c\theta_{23} + r_{23}s\theta_1 c\theta_{23}, r_{33}c\theta_{23} - r_{13}c\theta_1 s\theta_{23} - r_{23}s\theta_1\theta_{23}), \\
\theta_5 &= \text{atan2}\left(s\theta_1 r_{13} - c\theta_1 r_{23}, \pm\sqrt{1-(s\theta_1 r_{13} - c\theta_1 r_{23})^2}\right), \\
\theta_6 &= \text{atan2}(r_{11}s\theta_1 - r_{21}c\theta_1, r_{12}s\theta_1 - r_{22}c\theta_1).
\end{aligned} \tag{3.11}$$

In summary, we have introduced three models: (1) Kinematics of a 2-DOF manipulator, (2) Dynamics of a 2-DOF manipulator, and (3) Kinematics of a 6-DOF manipulator. In the following sections, we will apply different algorithms for each model and compare their performance.

3.3 Case Study: Regulation Problem of 2-DOF Planar Elbow Manipulator Systems

In this case study, we address the regulation problem associated with 2-DOF manipulator dynamics by using the following methods and algorithms:

- MPC: we apply an NMPC method proposed in [15].
- DRL: we apply and test two well-known DRL algorithms, DDPG [41] and SAC [23] for the manipulator system with continuous state and action spaces.
- MPC-based DRL: we adopt the MBPO algorithm detailed in [29].

The primary control objective is to regulate the system states $\boldsymbol{x} = [\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}]^\top$ from the origin to the reference states $\boldsymbol{x}_{\mathbf{ref}} = [\frac{\pi}{4}, \frac{\pi}{4}, 0, 0]^\top$, while satisfying the system constraints.

3.3.1 Methodologies

Nonlinear MPC

As aforementioned, MPC exploits the system model to determine a sequence of optimal control inputs by solving the formulated optimization problem. Before presenting the MPC optimization problem, we need to transform the nonlinear system dynamic (3.5) to a linear form by feedback linearization. Firstly, Equation (3.5) needs to be rewritten as

$$\ddot{\boldsymbol{\theta}} = -\mathbf{M}(\boldsymbol{\theta})^{-1}\mathbf{C}(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}})\dot{\boldsymbol{\theta}} + \mathbf{M}(\boldsymbol{\theta})^{-1}(\boldsymbol{\tau} - \mathbf{v}(\boldsymbol{\theta})). \tag{3.12}$$

By denoting $\ddot{\boldsymbol{\theta}} = \boldsymbol{w} = -\mathbf{M}(\boldsymbol{\theta})^{-1}\mathbf{C}(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}})\dot{\boldsymbol{\theta}} + \mathbf{M}(\boldsymbol{\theta})^{-1}(\boldsymbol{\tau} - \mathbf{v}(\boldsymbol{\theta}))$, we obtain the following LTI system in a state-space form:

$$\begin{aligned} \dot{\boldsymbol{x}} &= \mathbf{A}\boldsymbol{x} + \mathbf{B}\boldsymbol{w} \\ &= \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \boldsymbol{w}, \end{aligned} \tag{3.13}$$

where $\boldsymbol{x} = [\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}]^\top$ is a vector of joint angles and angular velocities. By using the zero-order hold method, the model (3.13) can be discretized with a sampling period ΔT. Thus, the discrete-time system matrices $\mathbf{A}_d$ and $\mathbf{B_d}$ shown as follows:

$$\mathbf{A}_d = e^{\mathbf{A}\Delta T}, \mathbf{B_d} = \int_0^{\Delta T} e^{\mathbf{A}\delta}\mathbf{B}d\delta. \tag{3.14}$$

The discrete-time LIT system is described as

$$\boldsymbol{x}_{k+1} = \begin{bmatrix} 1 & 0 & \Delta T & 0 \\ 0 & 1 & 0 & \Delta T \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \boldsymbol{x}_k + \begin{bmatrix} \frac{\Delta T^2}{2} & 0 \\ 0 & \frac{\Delta T^2}{2} \\ \Delta T & 0 \\ 0 & \Delta T \end{bmatrix} \boldsymbol{w}_k. \tag{3.15}$$

By substituting $\ddot{\boldsymbol{\theta}} = \boldsymbol{w}_k$ into (3.5), the control inputs $\boldsymbol{\tau}$ can be expressed as

$$\boldsymbol{u}_k = \boldsymbol{\tau}_k = \mathbf{M}(\boldsymbol{\theta}_k)\boldsymbol{w}_k + \mathbf{C}(\boldsymbol{\theta}_k, \dot{\boldsymbol{\theta}}_k)\dot{\boldsymbol{\theta}}_k + \mathbf{v}(\boldsymbol{\theta}_k). \tag{3.16}$$

Therefore, the NMPC optimization problem can be constructed as

$$\min_{\boldsymbol{w}_{k|k},\cdots} \sum_{l=k}^{k+N_{\mathrm{M}}-1} (\|\boldsymbol{x}_{\mathbf{ref}} - \boldsymbol{x}_{l|k}\|_{\boldsymbol{Q}} + \|\boldsymbol{w}_{l|k}\|_{\boldsymbol{R}}) + \|\boldsymbol{x}_{\mathbf{ref}} - \boldsymbol{x}_{k+N_{\mathrm{M}}|k}\|_{\boldsymbol{P}}, \tag{3.17a}$$

$$\text{s.t. } \boldsymbol{x}_{l+1|k} = \mathbf{A}_d \boldsymbol{x}_{l|k} + \mathbf{B}_d \boldsymbol{w}_{l|k}, \tag{3.17b}$$

$$\boldsymbol{u}_{l|k} = \mathbf{M}(\boldsymbol{\theta}_{l|k})\boldsymbol{w}_{l|k} + \mathbf{C}(\boldsymbol{\theta}_{l|k}, \dot{\boldsymbol{\theta}}_{l|k})\dot{\boldsymbol{\theta}}_{l|k} + \mathbf{v}(\boldsymbol{\theta}_{l|k}), \tag{3.17c}$$

$$\boldsymbol{x}_{k|k} = \boldsymbol{x}_k, \tag{3.17d}$$

$$\boldsymbol{x}_{l|k} \in \mathbb{X}, \boldsymbol{u}_{l|k} \in \mathbb{U}, \tag{3.17e}$$

$$\boldsymbol{x}_{k+N|k} \in \mathbb{X}_f, \tag{3.17f}$$

where $\boldsymbol{x}_{l|k}$ and $\boldsymbol{u}_{l|k}$ denote the predicted system states and control inputs, respectively. N_{M} is the prediction horizon. $\boldsymbol{Q}, \boldsymbol{R}$ and $\boldsymbol{P}$ denote positive definite weighting matrices. $\mathbb{X}_f$ is the terminal constraint set.

DDPG

In this case study, two model-free DRL algorithms will be tested and applied. The fisrt one is the SAC algorithm shown in Section 2.2.2, and the other one is the DDPG

algorithm [41]. The DDPG algorithm implements a deterministic policy, where the algorithm directly outputs a specific action rather than the distribution of actions. In contrast, SAC operates with a stochastic policy that maps states to a probability distribution.

We consider a standard setup of a discrete-time MDP with a tuple $(\mathbb{X}, \mathbb{U}, \mathcal{R}, \lambda, \pi_\theta)$, where $\mathbb{X} \in \mathbb{R}^n$, $\mathbb{U} \in \mathbb{R}^m$, and $\mathcal{R}$ are the state space, the action space, and the reward function, respectively. $\lambda \in [0, 1]$ is the learning rate. π_θ normally represents the stochastic learning policy π with respect to the parameter θ. We denote the policy as $\boldsymbol{u}_k \sim \pi_\theta(\cdot|\boldsymbol{x}_k)$. In this subsection, to distinguish the difference between the stochastic policy and the deterministic policy, we describe the deterministic policy as $\boldsymbol{u}_k = \mu_\theta(\boldsymbol{x}_k)$.

As aforementioned, DDPG combines the ideas from DPG [73] and DQN [48]. Compared with DQN, DDPG can handle problems with a continuous action space. Moreover, DDPG utilizes a target critic neural network with parameter η^-, a critic neural network with parameter η, a target actor neural network with parameter θ^-, and an actor neural network with parameter θ. The critic neural network estimates the action-value function $Q_\eta(\boldsymbol{x}_k, \boldsymbol{u}_k)$, which evaluates the quality of the action obtained by the actor neural network. The loss functions for updating the critic neural network and the actor neural network are as follows:

$$\begin{aligned} \delta(\eta) =& \mathbb{E}_{\boldsymbol{x}_k, \boldsymbol{u}_k, r_k, x_{k+1} \sim \beta}[(r_k + \gamma(Q_{\eta^-}(x_{k+1}, \mu_\theta^-(x_{k+1})))) - Q_\eta(\boldsymbol{x}_k, \boldsymbol{u}_k)], \\ \delta(\theta) =& \mathbb{E}_{\boldsymbol{x}_k \sim \beta}[Q_\eta(\boldsymbol{x}_k, \mu_\theta(\boldsymbol{x}_k))], \end{aligned} \tag{3.18}$$

where β denotes the replay buffer for storing measurement samples. Similar to SAC, DDPG also employs the soft updating method to update the target critic and actor

neural networks with a rate $\xi \in [0, 1]$:

$$\begin{aligned} \eta^- &\leftarrow \xi\eta + (1-\xi)\eta^-, \\ \theta^- &\leftarrow \xi\theta + (1-\xi)\theta^-. \end{aligned} \tag{3.19}$$

DDPG encourages the exploration in a continuous action space by adding noises $\mathcal{N}$ to the action obtained from the actor neural network. Thus, the action is denoted as

$$\boldsymbol{u}_k = \mu_\theta(\boldsymbol{x}_k) + \mathcal{N}. \tag{3.20}$$

MBPO

In subsections 2.2.2 and 3.3.1, we have discussed two model-free DRL algorithms. In this subsection, we will turn our attention to an MPC-based DRL algorithm, known as MBPO [29]. The MBPO algorithm combines the ideas of prediction from MPC and the strengths of model-free techniques from DRL.

In MPC, explicit system dynamics are constructed based on the existing knowledge and utilized to solve the optimization problem. This method predicts an action (control input) sequence over a prediction horizon of N_{M}, based on the current system state $\boldsymbol{x}_k$. However, if the system dynamics cannot be accurately modeled, the MPC method may result in undesired performance. In the MBPO algorithm, the model is trained by the collected samples to approximate the system dynamics as close as possible. The work in [29] constructed the model by a bootstrap ensemble of probabilistic neural networks, which output a Gaussian distribution. This model is employed to predict H rollouts, starting from the system state $\boldsymbol{x}_k$, over a prediction horizon of N_{R}. These rollouts are then implemented to solve the following MPC

optimization

$$\min_{\boldsymbol{u}_{k|k},\cdots} \sum_{l=k}^{k+N_{\mathrm{R}}-1} r(\boldsymbol{x}_{l|k}, \boldsymbol{u}_{l|k}), \tag{3.21a}$$

$$\text{s.t. } \boldsymbol{x}_{l+1|k} = f_\theta(\boldsymbol{x}_{l|k}, \boldsymbol{u}_{l|k}), \tag{3.21b}$$

$$\boldsymbol{x}_{k|k} = \boldsymbol{x}_k, \tag{3.21c}$$

where f_θ is an ensemble of bootstrapped probabilistic neural networks. While the data obtained from the interaction between the environment and the agent is stored in the replay buffer β_{env}, there is another replay buffer β_{model} for archiving rollouts that are predicted by the model. By collecting samples from both replay buffers and utilizing samples in a policy optimization algorithm, the MBPO algorithm would have better performance in term of the sample efficiency. In this case study, we utilize the DDPG algorithm for policy updating.

3.3.2 Simulation Results

In this subsection, a simulation is given to compare the performance of the NMPC [15], SAC [23], DDPG [41], and MBPO [29] algorithms for a 2-DOF manipulator system. Consider the systems in (3.15) and (3.16) with system constraints as follows:

$$\boldsymbol{x} \in \mathbb{X} \triangleq \left\{ \boldsymbol{x} \mid \begin{bmatrix} -\pi \\ -\pi \\ -\frac{\pi}{2} \\ -\frac{\pi}{2} \end{bmatrix} \leqslant \boldsymbol{x} \leqslant \begin{bmatrix} \pi \\ \pi \\ \frac{\pi}{2} \\ \frac{\pi}{2} \end{bmatrix} \right\}, \boldsymbol{u} \in \mathbb{U} \triangleq \{\boldsymbol{u} \mid \|\boldsymbol{u}\|_\infty \leqslant 20\}.$$

The sampling time is chosen as $\Delta T = 0.1s$. For NMPC, the weighting matrices are determined as $\boldsymbol{Q} = 100I_4, \boldsymbol{R} = I_2$, and the prediction horizon is chosen as $N_{\mathrm{M}} = 3$.

For DRL and MBPO algorithms, the prediction horizon is $N_{\mathrm{R}} = 1$, and the parameters are given in Table 3.4. During training process, we observe that a minor change in parameters may lead to significantly different results. Thus, we first choose hidden dimension as 64 in this study. Then, we manually tune the learning rates in a range from $1e^{-3}$ to $5e^{-5}$. In order to fairly compare the algorithm performance, we aim to use the most appropriate learning rate for each algorithm. Furthermore we choose different learning rates in this study. These algorithms contain 3 hidden layers for each actor and critic neural networks. The ReLU activation functions are used in DRL and MBPO algorithms. The SAC algorithm uses the softplus activation function between the last hidden layer and the output layer in the actor neural network. The reward function is selected as

$$r(\boldsymbol{x}_k) = -(\|\boldsymbol{x}_{\mathbf{ref}} - \boldsymbol{x}_k\| + \|\boldsymbol{u}_k\|). \tag{3.23}$$

The initial and the reference states are set as $\boldsymbol{x}_0 = [0, 0, 0, 0]^\top$ and $\boldsymbol{x}_{\mathbf{ref}} = [\frac{\pi}{4}, \frac{\pi}{4}, 0, 0]^\top$, respectively.

Table 3.4: Key parameters adopted in the DDPG, the SAC, and the MBPO algorithms

	MBPO	**DDPG**	**SAC**
The actor learning rate	$5e^{-4}$	$5e^{-4}$	$5e^{-3}$
The critic learning rate	$5e^{-3}$	$5e^{-3}$	$5e^{-3}$
The Learning rate γ	0.9	0.9	0.9
The temperature parameter ξ	N/A	N/A	$5e^{-3}$
The target entropy H_0	N/A	N/A	-1
The updating rate τ	0.005	0.005	0.005
Hidden dimension	64	64	64
Number of episodes	20	20	20
Size of buffer β_{env}	10000	10000	10000
Size of buffer β_{model}	1000	N/A	N/A

Figures 3.5, 3.6, and 3.7 show the trajectories of states $\boldsymbol{\theta}$, $\dot{\boldsymbol{\theta}}$, and actions(control

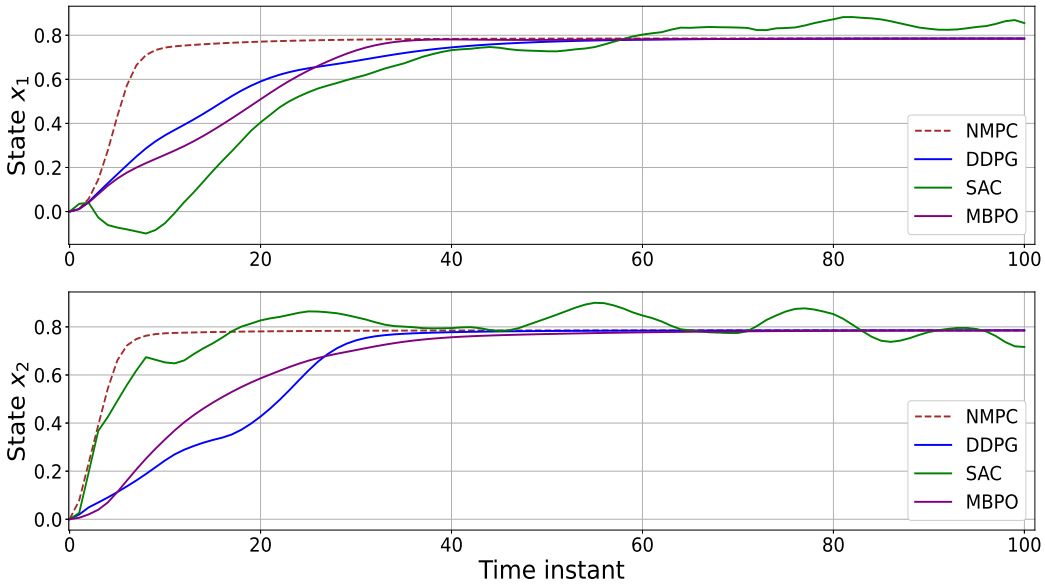

Figure 3.5: Comparison results of the trajectories of state $x_i, i = 1, 2$ for time instants $k = 0, \ldots, 100$ under NMPC, DDPG, SAC,and MBPO algorithms

inputs) $\boldsymbol{u}_k$, respectively, based on NMPC [15], DDPG [41], SAC [23], and MBPO algorithms [29]. From Figures 3.5 and 3.6, we can see that system states are regulated to the reference states as close as possible when time instant goes to infinity. Meanwhile, system constraints are satisfied during the control process. The only one that fails the primary control objective is the SAC algorithm. The reason is that the SAC algorithm uses a stochastic policy, and the implementation of the stochastic policy to a deterministic dynamical system may result in this fluctuation.

In addition to comparing control performance, we introduce two indexes $J_1 = \sum_{k=0}^{T}(\|\boldsymbol{x}_{\mathbf{ref}} - x_k\| + \|\boldsymbol{u}_k\|)$ and $J_2 = \sum_{k=0}^{T} \|\boldsymbol{x}_{\mathbf{ref}} - \boldsymbol{x}_k\|$. Table 3.5 presents the comparison results under different methods. When considering changes of both system states and control inputs, we observe that NMPC has an obviously better performance of J_1. When only considering the changes of system states, the index J_2 under DRL and MBPO algorithms are better than that under MPC. The reason why the

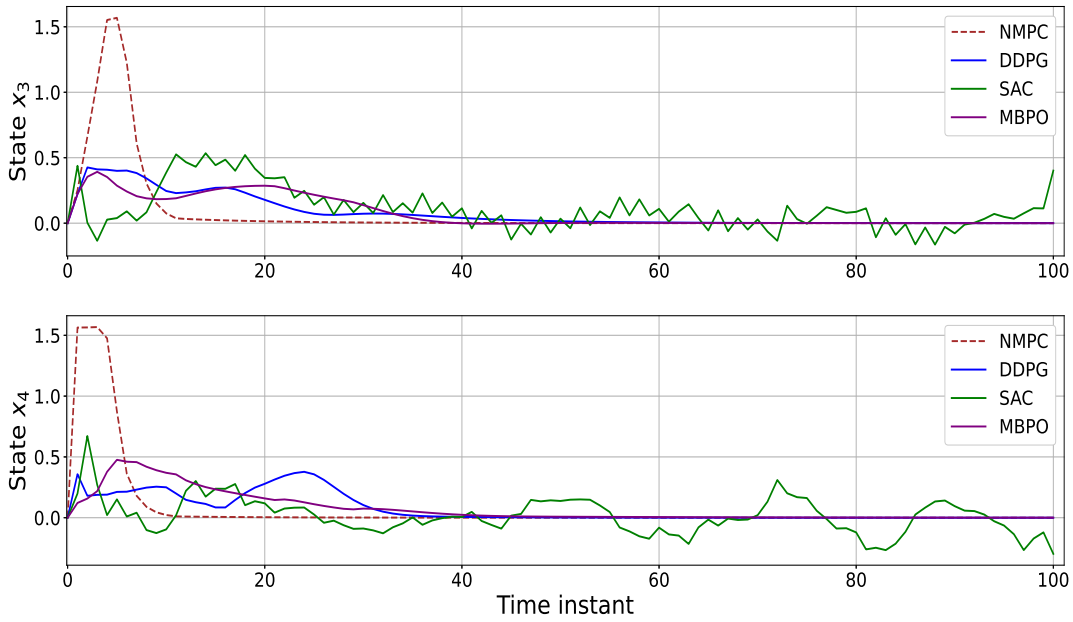

Figure 3.6: Comparison results of the trajectories of state $x_i, i = 3, 4$ for time instants $k = 0, \ldots, 100$ under DDPG, SAC, and MBPO algorithms

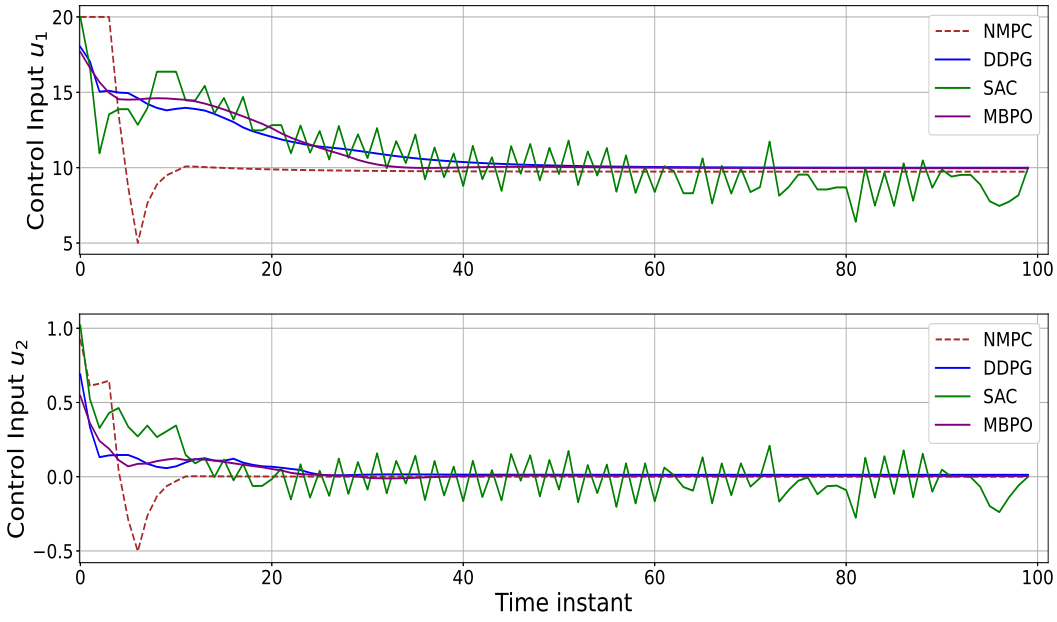

Figure 3.7: Comparison results of the trajectories of control input $\boldsymbol{u}_k$ under NMPC, DDPG, SAC, and MBPO algorithms

same methods have totally distinct performance between J_1 and J_2 is that the reward function (3.23) is purely relevant to the system states $\boldsymbol{x}_k$. Specifically, the optimal policies of DRL and MBPO algorithms aim to find an action (control input) sequence to minimize the total cost of system states.

Besides, we compare the computational burden from the time instant $k = 0$ to $k = 100$, which is shown in Table 3.6. The optimal policies of DRL and MBPO algorithms take less than 0.5 second to obtain the whole action sequence, while the NMPC method takes 71.34 seconds. The reason for this huge difference is that NMPC solves an optimization problem to obtain a control input at each time instant.

Table 3.5: The comparison results of control performance for time instants k=0, . . . , 100

	NMPC	***DDPG***	***SAC***	***MBPO***
J_1	10743.50	12643.59	22164.52	12606.19
J_2	23.2750	16.6288	20.2320	16.5028

Further, we compare the performance of total return and sample efficiency during the policy training process. Figures 3.8 and 3.9 show the trajectories of total return and sample sizes, respectively, during training the policy by implementing DDPG, SAC, and MBPO algorithms. Figure 3.8 demonstrates that the policy trained by the MBPO algorithm reaches the optimum 4 episodes and 6 episodes faster than the policies trained by both DDPG and SAC algorithms. From Figure 3.9, we see that the MBPO algorithm obtains the optimal policy with the fewest samples, which are approximately half as many as required by DDPG and SAC algorithms. The number of samples required to improve the policy is shown in Table 3.7.

Table 3.7 also presents the computational burden of training while utilizing different algorithms to optimize the policy. The MBPO algorithm requires the longest duration to complete the entire training process, taking approximately twice and

Table 3.6: The comparison results of computational burden for time instants k=0, . . . , 100

	NMPC	***DDPG***	***SAC***	***MBPO***
Time (s)	71.34	0.2283	0.4556	0.2232

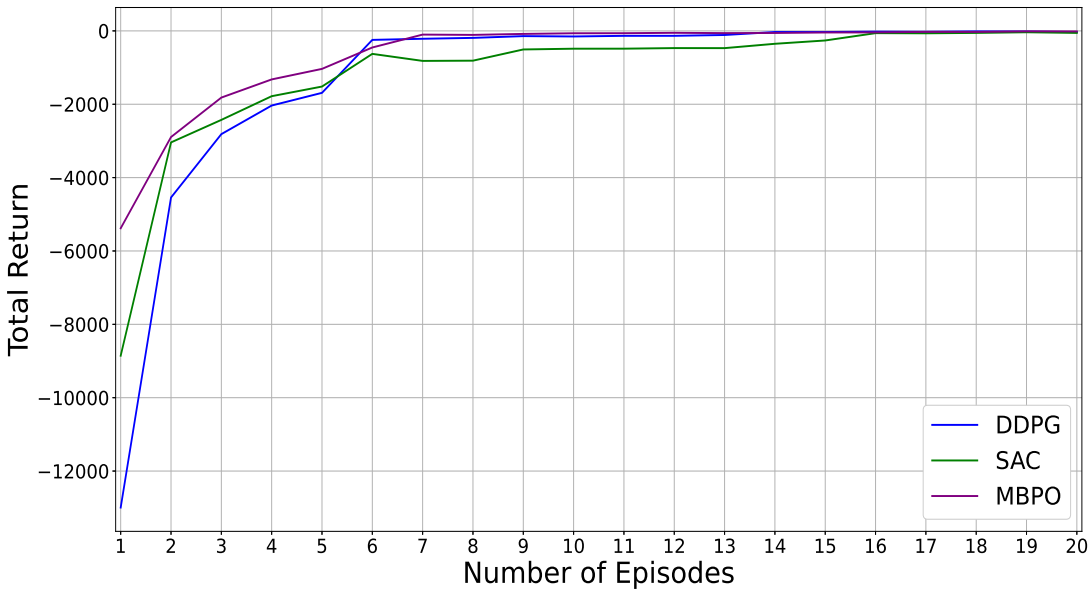

Figure 3.8: Comparison of the total return under NMPC, DDPG, SAC, and MBPO algorithms

thrice as much as SAC algorithm and DDPG algorithm demands. The reason is that the MBPO algorithm not only optimizes the policy, but also trains the learned model and implements the model to plan future states and action sequences. Compared with directly optimizing the policy, training the model may results in more computational complexity, especially for a highly complex environment.

Overall, although the MBPO algorithm has the best performance in terms of the cost, computational burden, and sample efficiency, the main limitation is the training burden.

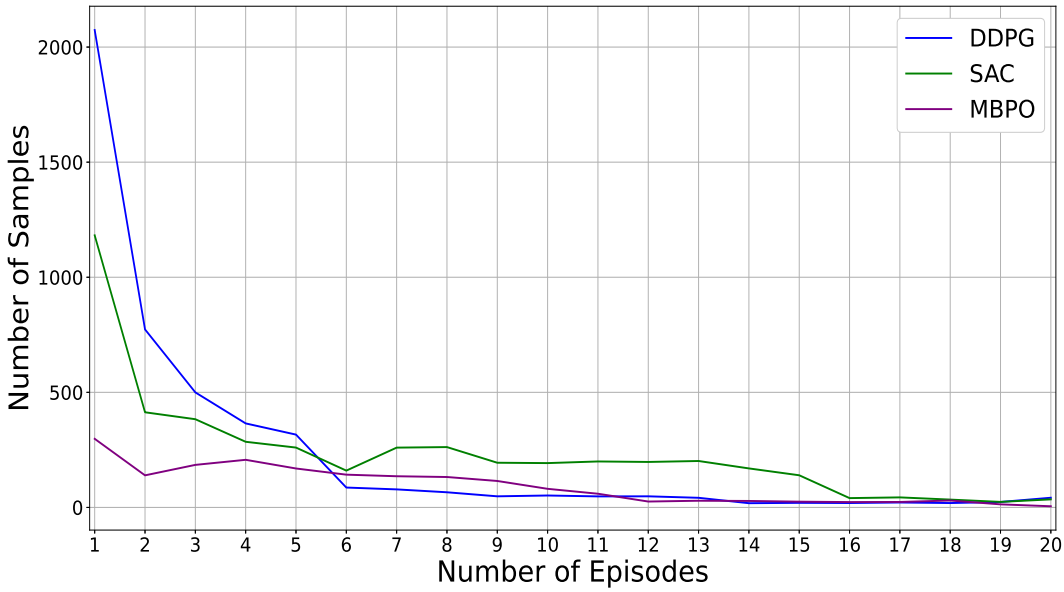

Figure 3.9: Comparison of the sample efficiency under DDPG, SAC, and MBPO algorithms

Table 3.7: The comparison results of sample efficiency and computational burden for episode number i = 1, . . . , 20

	DDPG	***SAC***	***MBPO***
Total samples	3072	3913	1809
Training time (s)	1119.8	1719.5	3511.2

3.4 Case Study: Tracking

In the case study 3.3, we have solved a regulation problem for a 2-DOF manipulator dynamics model by using NMPC, DDPG, SAC, and MBPO algorithms. By comparing simulation results, we conclude that, for a deterministic dynamical system, NMPC, DDPG, and MBPO algorithms successfully achieve the control objective. These three algorithms exhibit distinct strengths and limitations across various performance comparisons. In this case study, we focus on a tracking problem for a 2-DOF manipulator using kinematics model. Since we have introduced DDPG and MBPO

in Section 3.3, we will mainly illustrate the choice of system state and action, and the construction of reward function in this case study.

3.4.1 Problem Formulation

In this case study, the control objective is to control a 2-DOF manipulator system following arbitrary trajectories within the workspace. The secondary objective is to compare the training and control performance, under DDPG and MBPO algorithms.

We consider the kinematics of a 2-DOF robot arm shown in Figure 3.2 with parameters in Table 3.1. At the current time t, the reference joint angles $\widehat{\theta}_1(t), \widehat{\theta}_2(t)$ is obtained by using (3.4), based on the reference end-effector's positions $\widehat{d}_x(t)$ and $\widehat{d}_y(t)$. Consequently, we consider that the system state $\boldsymbol{x}(t)$ consists of two components: the positional error $\boldsymbol{e_d}(t)$ between the reference and current positions, and the angular error $\boldsymbol{e_\theta}(t)$ between the reference and current angles. Then, the system state $\boldsymbol{x}(t)$ can be expressed as

$$\boldsymbol{x}(t) = \begin{bmatrix} \boldsymbol{e_d}(t) \\ \boldsymbol{e_\theta}(t) \end{bmatrix} = \begin{bmatrix} \widehat{d}_x(t) - d_x(t) \\ \widehat{d}_y(t) - d_y(t) \\ \widehat{\theta}_1(t) - \theta_1(t) \\ \widehat{\theta}_2(t) - \theta_2(t) \end{bmatrix}. \tag{3.24}$$

The action is chosen to be angular changes applied on each joint, and shown as

$$\boldsymbol{u}(t) = \begin{bmatrix} \Delta\theta_1(t) \\ \Delta\theta_2(t) \end{bmatrix}. \tag{3.25}$$

The reward function is described as follows:

$$r(t) = \begin{cases} -\|\boldsymbol{e_d}(t)\|, & \boldsymbol{e_d}(t) \neq 0, \\ 1, & \boldsymbol{e_d}(t) = 0. \end{cases} \tag{3.26}$$

However, during the policy training process, we discover that the condition $\boldsymbol{e_d}(t) = 0$ was difficult to be satisfied, which make the policy hardly converge to the optimum. To solve this problem, we adjust the terminal condition by adding a tolerance ζ to $\boldsymbol{e_d}(t)$. As a result, the new reward function is defined as follows:

$$r(t) = -\begin{cases} \|\boldsymbol{e_d}(t)\|, & \boldsymbol{e_d}(t) > \zeta, \\ 1, & \boldsymbol{e_d}(t) \leq \zeta. \end{cases} \tag{3.27}$$

3.4.2 Simulation Results

In this subsection, simulation and comparison results are provided to show the control and training performance of DDPG [41] and MBPO [29] algorithms for a tracking problem with the 2-DOF manipulator system using kinematics model. The system constraints are chosen as follows:

$$\boldsymbol{x}(t) \in \mathbb{X} \triangleq \left\{ \boldsymbol{x} \mid \begin{bmatrix} -0.8 \\ -0.8 \\ -\pi \\ -\pi \end{bmatrix} \leqslant \boldsymbol{x} \leqslant \begin{bmatrix} 0.8 \\ 0.8 \\ \pi \\ \pi \end{bmatrix} \right\}, \boldsymbol{u}(t) \in \mathbb{U} \triangleq \{\boldsymbol{u} \mid \|\boldsymbol{u}\|_\infty \leqslant \frac{\pi}{90}\}.$$

The sampling period is set as $\Delta T = 0.1s$. The parameters for DDPG and MBPO algorithms are shown in Table 3.8. Both algorithms contain 3 hidden layers with ReLU activation functions for each actor and critic neural networks. The initial

Table 3.8: Key parameters adopted in the DDPG and MBPO algorithms

	MBPO	**DDPG**
The actor learning rate	$3e^{-3}$	$1e^{-4}$
The critic learning rate	$3e^{-3}$	$1e^{-4}$
The Learning rate γ	0.9	0.8
The updating rate τ	0.005	0.002
Hidden dimension	64	100
Number of episodes	300	300
Size of buffer β_{env}	10000	10000
Size of buffer β_{model}	1000	N/A
Prediction Horizon N_{R}	1	N/A

point is chosen as $\boldsymbol{x}(t) = [0.16, 0]^\top$, and the reference trajectory is designed as

$$\begin{cases} x_r(t) = 0.16 \cos \dfrac{\pi t}{40}, \\ y_r(t) = 0.16 \sin \dfrac{\pi t}{40}. \end{cases}$$

To achieve the control objectives, during the training process, we randomly select initial points and reference points within the workspace for each episode, with a maximum limit of 150 steps per episode. The terminal condition of each episode is to constantly satisfy the reward condition $\boldsymbol{e_d} \leq \zeta$ with 50 steps. If this terminal condition is achieved within 150 steps, this episode is ended as well. Therefore, the optimal return is the cumulative rewards for achieving this terminal condition with exact 50 steps, which implies that the total return is 50. By using aforementioned setup, the optimal policy is expected to enable the manipulator's end-effector to reach an arbitrary point within the workspace.

Figures 3.10 and 3.11 show the trajectories of total return and sample efficiency during the policy training process, using DDPG and MBPO algorithms. From Figure 3.10, we can see that the total return approaches the optimum around the 250th and 170th episodes, respectively. Thus, the MBPO has better performance of total return.

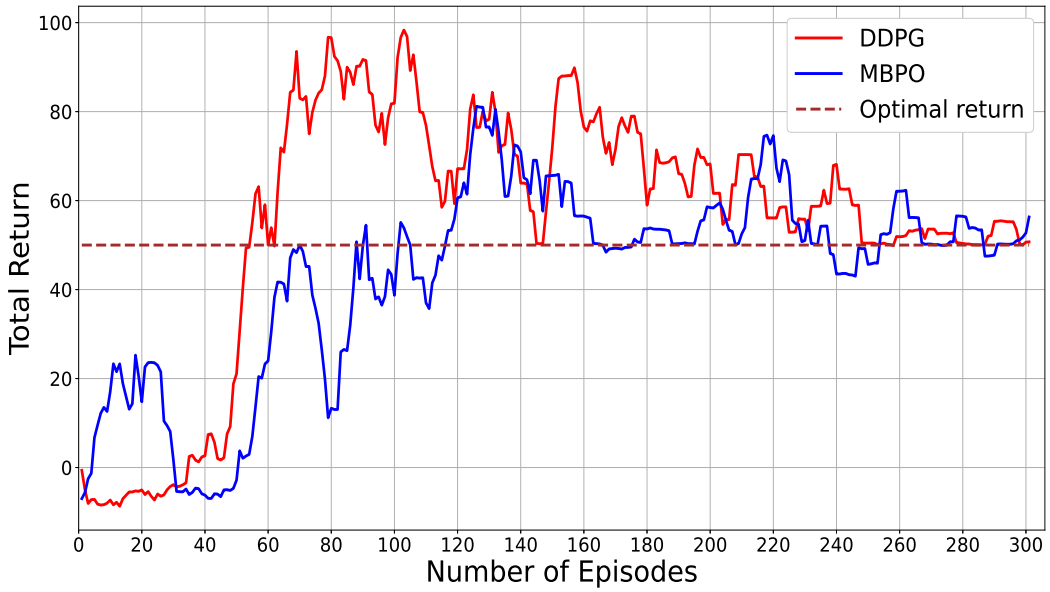

Figure 3.10: Comparison of the total return under DDPG and MBPO algorithms

Figure 3.11 presents the trajectories of the number of samples required by DDPG and MBPO algorithms. In the initial 200 episodes, it is difficult to compare the sample efficiency performance. Between episodes 200 and 300, the MBPO algorithm requires fewer samples to complete one episode. To further compare the sample efficiency, Table 3.9 lists the total samples required to train 300 episodes, and the table also shows that the MBPO algorithm has better a performance of sample efficiency for the tracking problem.

We next compare the computational burden for training, and the comparison results are shown in Table 3.9. Compared with the DDPG algorithm, the MBPO algorithm takes extra time to train the model and predict future states and actions.

Figure 3.12 presents the reference trajectory and trajectories obtained from two policies. We see that both policies trained by MBPO and DDPG algorithms can successfully track the reference trajectory. Therefore, we introduce an index $J = \sum_{t=0}^{40} \|\boldsymbol{e_d}(t)\|$, which describes the total positional error over 40 seconds. The com-

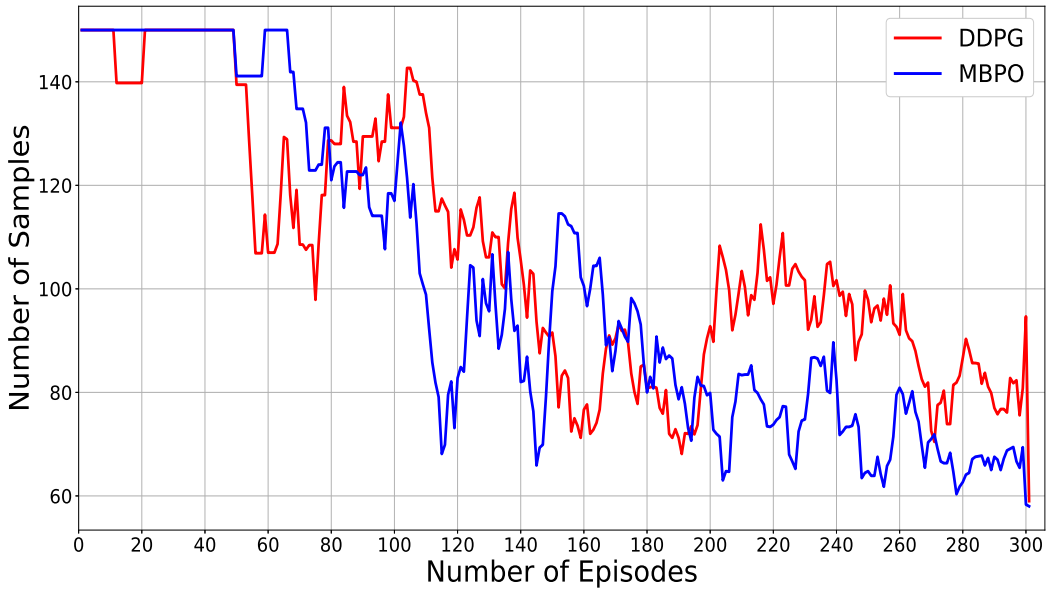

Figure 3.11: Comparison of the sample efficiency under DDPG and MBPO algorithms

Table 3.9: The comparison results of sample efficiency and computational burden for episode number i = 1, . . . , 300

	DDPG	***MBPO***
Index J	1740	1786
Total samples	32405	30506
Training time (s)	3604.3	6049.6

parison result is shown in Table 3.9. We observe that the index J, from DDPG algorithm, is lower than that of the MBPO algorithm, which implies that the DDPG algorithm exhibits better tracking performance compared with the MBPO algorithm.

3.5 Case Study: Regulation Problem of UR10e

In Section 3.3, we have analyzed a regulation problem for a deterministic dynamical system using various algorithms. We find that the MBPO algorithm demonstrates

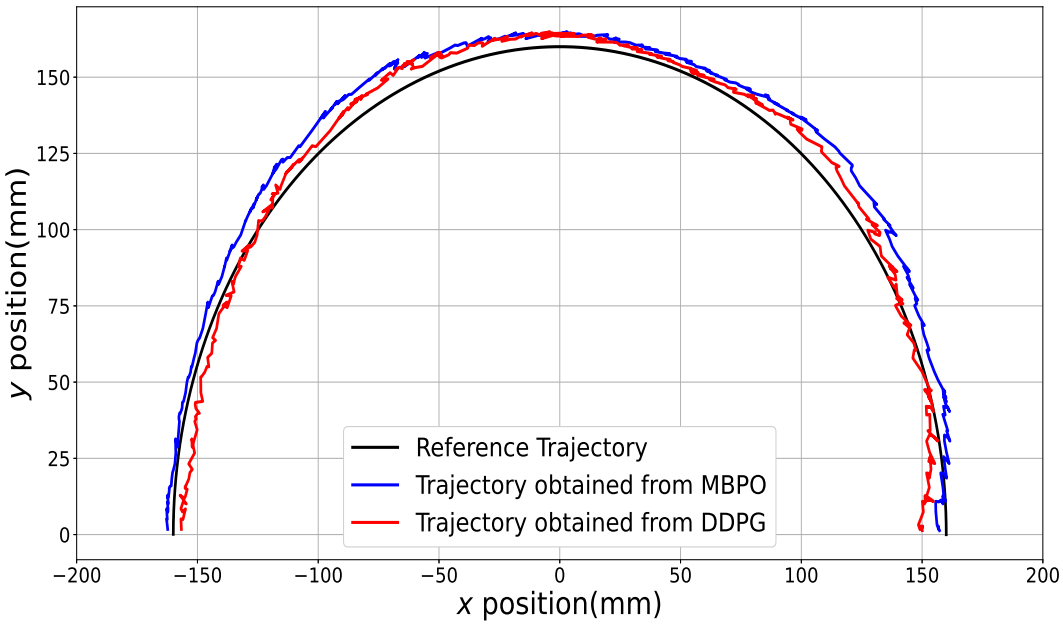

Figure 3.12: Comparison of the tracking performance under DDPG and MBPO algorithms

the best performance during both training and control processes compared with DRL algorithms. The limitation of the MBPO algorithm is the computational burden. In Section 3.4, we increase the complexity of the problem, and the model becomes more difficult to be trained. Thus, the MBPO algorithm exhibits poorer performance in terms of the computational burden and control outcomes, and its sample efficiency was only marginally better than that of DDPG. In this case study, we will further compare DRL and MPC-based RL algorithms by increasing the dimensions of system states and actions. In addition, we will compare the training performance by applying DRL and MBPO algorithms for the kinematics of UR10e.

3.5.1 Problem Formulation

Previous two case studies present that the MBPO algorithm shows poorer performance with the increasing complexity of the problem. Therefore, in this case study, the primary objective is to compare the performance of DDPG, SAC, and MBPO algorithms during training process for the UR10e manipulator system, which has twelve-dimensional states and six-dimensional actions.

In this case study, we consider the kinematics model of UR10e whose state and action are chosen as

$$\boldsymbol{x}(t) = \begin{bmatrix} \boldsymbol{d}(t) \\ \boldsymbol{O}(t) \\ \boldsymbol{\theta}(t) \end{bmatrix} = \begin{bmatrix} d_x(t) \\ d_y(t) \\ d_z(t) \\ O_\phi(t) \\ O_\theta(t) \\ O_\psi(t) \\ \theta_1(t) \\ \theta_2(t) \\ \theta_3(t) \\ \theta_4(t) \\ \theta_5(t) \\ \theta_6(t) \end{bmatrix}, \quad \boldsymbol{u}(t) = \boldsymbol{\Delta\theta}(t) = \begin{bmatrix} \Delta\theta_1(t) \\ \Delta\theta_2(t) \\ \Delta\theta_3(t) \\ \Delta\theta_4(t) \\ \Delta\theta_5(t) \\ \Delta\theta_6(t) \end{bmatrix}, \tag{3.29}$$

where $\boldsymbol{d}(t), \boldsymbol{O}(t)$, and $\boldsymbol{\theta}(t)$ denote the position, orientation and corresponding angles of each joint, respectively. $\boldsymbol{\Delta\theta}(t)$ is the angular change of each joint. Based on the initial position $\boldsymbol{d}(0)$ and orientation $\boldsymbol{O}(0)$, the corresponding initial angle of each joint $\boldsymbol{\theta}(0)$ is calculated by solving the inverse kinematics problems (3.9) and (3.11). After taking the action $\boldsymbol{u(0)}$, the next position $\boldsymbol{d}(\Delta T)$ and orientation $\boldsymbol{O}(\Delta T)$ are

obtained by solving the forward kinematics problem (3.7), where ΔT is the sampling period. The above steps are repeated until the reference position $\boldsymbol{d}_{\mathbf{ref}}$ and orientation $\boldsymbol{O}_{\mathbf{ref}}$ are achieved. Moreover, the reward function is designed to be the error between current and reference position and orientation.

$$r(\boldsymbol{x}_k) = -(\|\boldsymbol{d}_{\mathbf{ref}} - \boldsymbol{d}(t)\| + \|\boldsymbol{O}_{\mathbf{ref}} - \boldsymbol{O}(t)\|). \tag{3.30}$$

3.5.2 Simulation Results

In this subsection, a simulation is given to show the performance of SAC, DDPG, and MBPO algorithms for the kinematics of a UR10e system. The system constraints are considered as

$$\boldsymbol{x} \in \mathbb{X} \triangleq \left\{ \boldsymbol{x} \mid \begin{bmatrix} \boldsymbol{-1.3} \\ \boldsymbol{-2\pi} \\ \boldsymbol{-2\pi} \end{bmatrix} \leqslant \begin{bmatrix} \boldsymbol{d}(t) \\ \boldsymbol{O}(t) \\ \boldsymbol{\theta}(t) \end{bmatrix} \leqslant \begin{bmatrix} \boldsymbol{1.3} \\ \boldsymbol{2\pi} \\ \boldsymbol{2\pi} \end{bmatrix} \right\}, \boldsymbol{u} \in \mathbb{U} \triangleq \{\boldsymbol{u} \mid \|\boldsymbol{u}\|_\infty \leqslant \frac{\pi}{180}\}.$$

The sampling period is chosen as $\Delta = 0.1s$. Table 3.10 shows the parameters used for training the policies. The initial position and orientation are $[\boldsymbol{d}(0), \boldsymbol{O}(0)]^\top = [0, -0.2907, 1.2451, 0, 0, \pi]^\top$. The reference position and orientation are $[\boldsymbol{d}_{\mathbf{ref}}, \boldsymbol{O}_{\mathbf{ref}}]^\top = [0.5, 0.5, 1, -\frac{\pi}{2}, \frac{\pi}{2}, 0]^\top$.

Figures 3.13 and 3.14 show the trajectories of total return and samples during training the policy. From Figure 3.13, the SAC algorithm exhibits the best performance in terms of the total return. The policy updated by SAC converges to the optimum at the sixth episode, while DDPG and MBPO algorithms obtain optimal policies at the fifteenth and sixteenth episodes, respectively. From Figure 3.14, we observe that the MBPO algorithm demonstrates the poorest sample efficiency in this

Table 3.10: Key parameters adopted in the MBPO, DDPG, and SAC algorithms

	MBPO	**DDPG**	**SAC**
The actor learning rate	$3e^{-3}$	$3e^{-3}$	$3e^{-3}$
The critic learning rate	$3e^{-3}$	$3e^{-3}$	$3e^{-3}$
The Learning rate γ	0.9	0.9	0.9
The temperature parameter ξ	N/A	N/A	$3e^{-3}$
The target entropy H_0	N/A	N/A	-1
The updating rate τ	0.005	0.005	0.005
Hidden dimension	64	64	128
Number of episodes	20	20	20
Size of buffer β_{env}	10000	10000	10000
Size of buffer β_{model}	1000	N/A	N/A
Prediction horizon N_{R}	1	N/A	N/A

Table 3.11: The comparison results of sample efficiency and computational for episode number i = 1, . . . , 20

	DDPG	***SAC***	***MBPO***
Total samples	1959	1891	6091
Training time (s)	589.5	493.2	1996.5

case study, requiring approximately three times as many samples as SAC and DDPG to complete training for 20 episodes. Table 3.11 presents the detailed number of total samples used for training the policy and the total training time in seconds. As aforementioned, existing MPC-based RL algorithms utilize a learned model to predict future states and actions that are implemented to improve the policy. Therefore, the computational complexity of training a model for a high-dimensional system is the primary reason that the MBPO algorithm has the poorest performance during training.

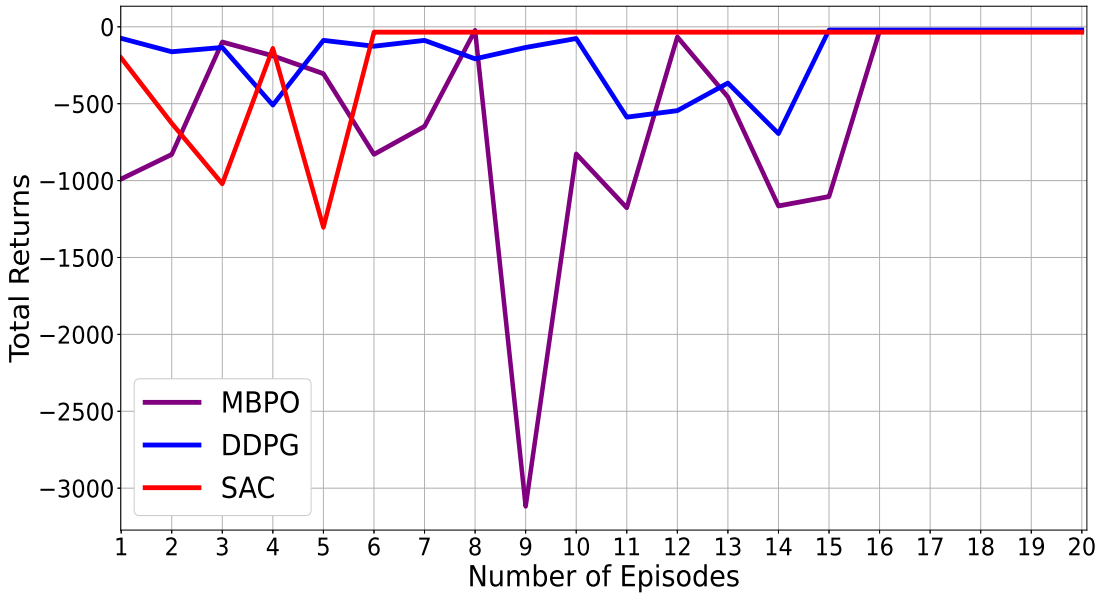

Figure 3.13: Comparison of the total return under DDPG, SAC, and MBPO algorithms

3.6 Conclusion

In this chapter, three cases are studied for different control problems and manipulator systems. Firstly, in Section 3.3, we implement NMPC, DDPG, SAC, and MBPO algorithms to solve a regulation problem involving a 2-DOF manipulator system. Secondly, in Section 3.4, we compare DDPG and MBPO algorithms for a tracking problem of the same 2-DOF manipulator system in Section 3.3. Thirdly, Section 3.5 focuses on the training performance of DDPG, SAC, and MBPO algorithms for a regulation problem with a UR10e system. According to the simulation results, the NMPC has the best control performance. However, there are two primary limitations of NMPC. The first limitation is the requirement of explicit system dynamics for solving the pre-designed optimization problem. The second limitation of NMPC is the computational complexity. We then consider the comparison between model-free DRL algorithms and the MPC-based RL algorithm, both of which require samples

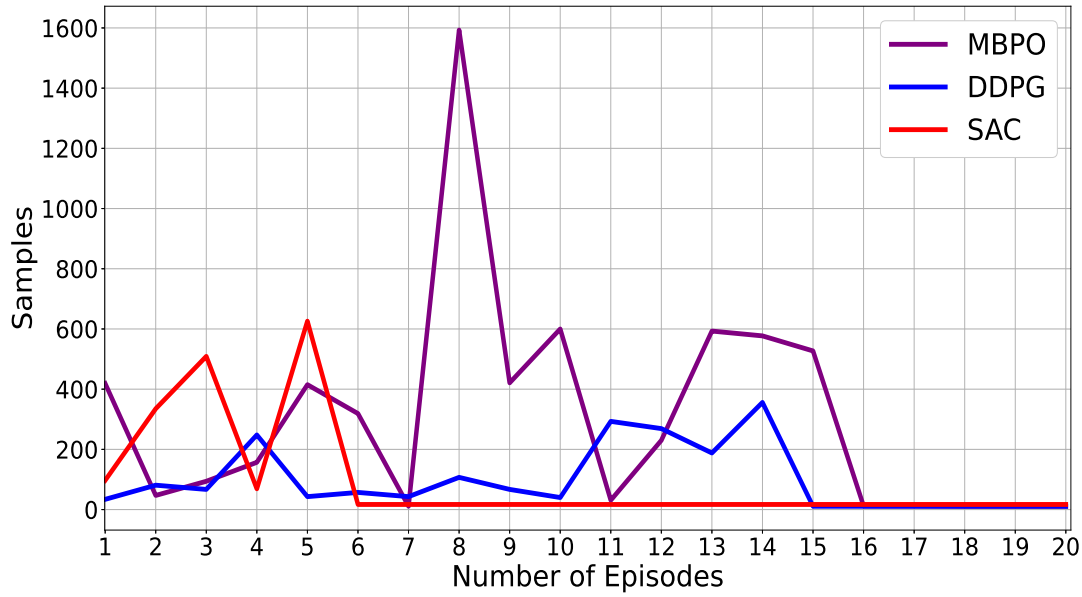

Figure 3.14: Comparison of the sample efficiency under DDPG, SAC, and MBPO algorithms

to optimize the policy. In Section 3.3, compared with model-free algorithms, the MBPO algorithm exhibits the best training and control performance in terms of the total return, sample efficiency, and cost. With the increasing complexity of the problem, the MBPO algorithm shows the enhanced performance of the total return and sample efficiency. While solving a regulation problem with the high-dimensional system in Section 3.5, the MBPO algorithm demonstrates the poorest performance during training. This underperformance is attributed to the challenges of training the learned model with a high-dimensional system.

Chapter 4

Conclusions and Future work

4.1 Conclusions

In this thesis, a literature review is given, regrading the incorporation of MPC and DRL. Then, to tackle the sample efficiency problem, we propose an SAC-RTMPC framework for constrained linear systems with bounded additive disturbances. Further, we investigate the application of three representative methods, i.e., MPC, RL, and MPC-based RL, for robot manipulators.

In Chapter 2, an SAC-RTMPC framework is proposed for constrained linear systems with bounded additive disturbances. In the proposed framework, the RTMPC method is employed to predict future state and action sequences, while respecting system constraints. The predicted sequences are stored in the replay buffer as the form of data pairs. Further, these data pairs and real samples are jointly applied to modify the TD error function and improve the policy by using the SAC algorithm. Finally, numerical simulation and comparison studies are given to demonstrate that our proposed framework enhances control performance, compared with RTMPC.

In Chapter 3, we focus on simulation studies of applying the NMPC method,

the MBPO algorithm, the DDPG algorithm, and the SAC algorithm to robot manipulators. The NMPC method has the best control performance in the first case study. However, the primary limitation of the NMPC method is the computational complexity, making it difficult to satisfy a high-frequency sample period. Moreover, compared with the model-free DRL algorithms, the MBPO algorithm presents the best training and control performance. In the second case study, compared with DDPG, the MBPO exhibits an improved training performance but a poorer tracking performance. In the last case study, we compare the training performance among SAC, DDPG, and MBPO algorithms for the regulation problem with the 6-DOF UR10e manipulator system. Unfortunately, the MBPO algorithm shows the worst training performance. Further, in three simulation studies, the MBPO algorithm shows a high computational burden to optimize the policy and predict future actions.

4.2 Future work

In Chapter 2, we have proposed an MPC-based DRL framework for a constrained linear system with bounded disturbances. The predicted trajectories obtained from RMPC are utilized to improve DRL's policy. However, the proposed framework cannot successfully solve control problems for highly nonlinear systems with unknown disturbances. Thus, how to successfully employ the framework in nonlinear systems with unknown disturbance bounds remains as an open problem.

In Chapter 3, we investigate the application of the NMPC method, the MBPO, DDPG, and SAC algorithms to robot manipulators. For the high-dimensional manipulator system, the MBPO algorithm presents a poorer training performance in terms of sample efficiency and computational burden. It should be mentioned that the current results neglect the safety issues during training the policy, which is a

critical issues in practical robot manipulators. Thus, how to design an MPC-based DRL with an improvement in sample efficiency and safety satisfaction for practical manipulators will be further investigated.

Bibliography

[1] Md Hazrat Ali, K Aizat, K Yerkhan, T Zhandos, and O Anuar. Vision-based robot manipulator for industrial applications. *Procedia Computer Science*, 133:205–212, 2018.

[2] Haider AF Almurib, Haidar Fadhil Al-Qrimli, and Nandha Kumar. A review of application industrial robotic design. In *Proceedings of 2011 International Conference on ICT and Knowledge Engineering*, pages 105–112. IEEE, 2012.

[3] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

[4] Anil Aswani, Humberto Gonzalez, S Shankar Sastry, and Claire Tomlin. Provably safe and robust learning-based model predictive control. *Automatica*, 49(5):1216–1226, 2013.

[5] Isin M Balci, Efstathios Bakolas, Bogdan Vlahov, and Evangelos A Theodorou. Constrained covariance steering based tube-MPPI. In *Proceedings of 2022 American Control Conference*, pages 4197–4202. IEEE, 2022.

[6] André Barreto, Will Dabney, Rémi Munos, Jonathan J Hunt, Tom Schaul, Hado P van Hasselt, and David Silver. Successor features for transfer in re-

inforcement learning. In *Proceedings of 2017 Advances in Neural Information Processing Systems*, 2017.

[7] Richard Bellman. On the theory of dynamic programming. *Proceedings of the National Academy of Sciences*, 38(8):716–719, 1952.

[8] Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[9] Franco Blanchini and Stefano Miani. *Set-Theoretic Methods in Control*, volume 78. Springer, 2008.

[10] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010: 19th International Conference on Computational StatisticsParis France, August 22-27, 2010 Keynote, Invited and Contributed Papers*, pages 177–186. Springer, 2010.

[11] Haitham Bou-Ammar, Holger Voos, and Wolfgang Ertel. Controller design for quadrotor UAVs using reinforcement learning. In *Proceedings of 2010 IEEE International Conference on Control Applications*, pages 2130–2135. IEEE, 2010.

[12] Yassine Bouteraa, Ismail Ben Abdallah, and Jawhar Ghommam. Task-space region-reaching control for medical robot manipulator. *Computers & Electrical Engineering*, 67:629–645, 2018.

[13] Lucian Busoniu, Robert Babuska, Bart De Schutter, and Damien Ernst. *Reinforcement Learning and Dynamic Programming Using Function Approximators*. CRC Press, 2017.

[14] Ilse Cervantes and Jose Alvarez-Ramirez. On the PID tracking control of robot manipulators. *Systems & Control Letters*, 42(1):37–46, 2001.

[15] Hong Chen and Frank Allgöwer. A quasi-infinite horizon nonlinear model predictive control scheme with guaranteed stability. *Automatica*, 34(10):1205–1217, 1998.

[16] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Proceedings of 2018 Advances in Neural Information Processing Systems*, volume 31, 2018.

[17] Li Dai, Yuantao Yu, Di-Hua Zhai, Teng Huang, and Yuanqing Xia. Robust model predictive tracking control for robot manipulators with disturbances. *IEEE Transactions on Industrial Electronics*, 68(5):4288–4297, 2020.

[18] Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on Machine Learning*, pages 465–472, 2011.

[19] Alexey Dosovitskiy and Vladlen Koltun. Learning to act by predicting the future. *arXiv preprint arXiv:1611.01779*, 2016.

[20] Sébastien Gros and Mario Zanon. Data-driven economic NMPC using reinforcement learning. *IEEE Transactions on Automatic Control*, 65(2):636–648, 2019.

[21] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *Proceedings of 2017 IEEE International Conference on Robotics and Automation*, pages 3389–3396. IEEE, 2017.

[22] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep Q-learning with model-based acceleration. In *Proceedings of 2016 International Conference on Machine Learning*, pages 2829–2838. PMLR, 2016.

[23] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of 2018 International Conference on Machine Learning*, pages 1861–1870, 2018.

[24] Wei He, Yuhao Chen, and Zhao Yin. Adaptive neural network control of an uncertain robot with full-state constraints. *IEEE Transactions on Cybernetics*, 46(3):620–629, 2015.

[25] Geoffrey Hinton and Terrence J Sejnowski. *Unsupervised Learning: Foundations of Neural Computation*. MIT Press, 1999.

[26] Eyke Hüllermeier and Willem Waegeman. Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods. *Machine Learning*, 110(3):457–506, 2021.

[27] Julian Ibarz, Jie Tan, Chelsea Finn, Mrinal Kalakrishnan, Peter Pastor, and Sergey Levine. How to train your robot with deep reinforcement learning: lessons we have learned. *The International Journal of Robotics Research*, 40(4-5):698–721, 2021.

[28] Nursultan Imanberdiyev, Changhong Fu, Erdal Kayacan, and I-Ming Chen. Autonomous navigation of UAV by using real-time model-based reinforcement learning. In *Proceedings of 2016 International Conference on Control, Automation, Robotics and Vision*, pages 1–6. IEEE, 2016.

[29] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. In *Proceedings of 2019 Advances in Neural Information Processing Systems*, volume 32, 2019.

[30] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[31] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Scalable deep reinforcement learning for vision-based robotic manipulation. In *Proceedings of 2018 Conference on Robot Learning*, pages 651–673. PMLR, 2018.

[32] Md Abdus Samad Kamal, Masakazu Mukai, Junichi Murata, and Taketoshi Kawabe. Model predictive control of vehicles on urban roads for improved fuel economy. *IEEE Transactions on Control Systems Technology*, 21(3):831–841, 2012.

[33] Sébastien Kleff, Avadesh Meduri, Rohan Budhiraja, Nicolas Mansard, and Ludovic Righetti. High-frequency nonlinear model predictive control of a manipulator. In *Proceedings of 2021 IEEE International Conference on Robotics and Automation*, pages 7330–7336. IEEE, 2021.

[34] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[35] Amit Konar, Indrani Goswami Chakraborty, Sapam Jitu Singh, Lakhmi C Jain, and Atulya K Nagar. A deterministic improved Q-learning for path planning of a mobile robot. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 43(5):1141–1153, 2013.

[36] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. In *Proceedings of 1999 Advances in Neural Information Processing Systems*, pages 1008–1014, 1999.

[37] B. Kouvaritakis and M. Cannon. *Model Predictive Control: Classical, Robust and Stochastic*. Advanced Textbooks in Control and Signal Processing. Springer International Publishing, 2015.

[38] Sergey Levine and Vladlen Koltun. Guided policy search. In *Proceedings of 2013 International Conference on Machine Learning*, pages 1–9. PMLR, 2013.

[39] Huiping Li and Yang Shi. *Robust Receding Horizon Control for Networked and Distributed Nonlinear Systems*. Springer, 2017.

[40] Zhijun Li, Ting Zhao, Fei Chen, Yingbai Hu, Chun-Yi Su, and Toshio Fukuda. Reinforcement learning of manipulation and grasping using dynamical movement primitives for a humanoidlike mobile manipulator. *IEEE/ASME Transactions on Mechatronics*, 23(1):121–131, 2017.

[41] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[42] Junjia Liu, Jiaying Shou, Zhuang Fu, Hangfei Zhou, Rongli Xie, Jun Zhang, Jian Fei, and Yanna Zhao. Efficient reinforcement learning control for continuum robots based on inexplicit prior knowledge. *arXiv preprint arXiv:2002.11573*, 2020.

[43] Ying Liu, Du Jiang, Juntong Yun, Ying Sun, Cuiqiao Li, Guozhang Jiang, Jianyi Kong, Bo Tao, and Zifan Fang. Self-tuning control of manipulator positioning based on fuzzy PID and PSO algorithm. *Frontiers in Bioengineering and Biotechnology*, 9:817723, 2022.

[44] Tianxiang Lu, Kunwu Zhang, and Yang Shi. SARSA-based model predictive control with improved performance and computational complexity. In *Proceedings*

of 2022 IEEE International Conference on Industrial Cyber-Physical Systems, pages 01–06, 2022.

[45] David Q Mayne, James B Rawlings, Christopher V Rao, and Pierre OM Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36(6):789–814, 2000.

[46] David Q Mayne, María M Seron, and SV Raković. Robust model predictive control of constrained linear systems with bounded disturbances. *Automatica*, 41(2):219–224, 2005.

[47] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[48] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[49] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. MIT Press, 2018.

[50] Hoomaan MoradiMaryamnegari, Marco Frego, and Angelika Peer. Data-driven model predictive control using deep double expected Sarsa. In *Proceedings of 2023 International Conference on Control, Decision and Information Technologies*, pages 345–350, 2023.

[51] Andrew S Morgan, Daljeet Nandha, Georgia Chalvatzaki, Carlo D'Eramo, Aaron M Dollar, and Jan Peters. Model predictive actor-critic: Accelerating robot skill acquisition with deep reinforcement learning. In *Proceedings of 2021*

IEEE International Conference on Robotics and Automation, pages 6672–6678. IEEE, 2021.

[52] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *Proceedings of 2018 IEEE International Conference on Robotics and Automation*, pages 7559–7566. IEEE, 2018.

[53] Peng Peng, Ying Wen, Yaodong Yang, Quan Yuan, Zhenkun Tang, Haitao Long, and Jun Wang. Multiagent bidirectionally-coordinated nets: Emergence of human-level coordination in learning to play starcraft combat games. *arXiv preprint arXiv:1703.10069*, 2017.

[54] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *Proceedings of 2018 IEEE International Conference on Robotics and Automation*, pages 3803–3810. IEEE, 2018.

[55] Ryan S Penning, Jinwoo Jung, Justin A Borgstadt, Nicola J Ferrier, and Michael R Zinn. Towards closed loop control of a continuum robotic manipulator for medical applications. In *Proceedings of 2011 IEEE International Conference on Robotics and Automation*, pages 4822–4827. IEEE, 2011.

[56] Ph Poignet and Maxime Gautier. Nonlinear model predictive control of a robot manipulator. In *Proceedings of 6th International Workshop on Advanced Motion Control*, pages 401–406. IEEE, 2000.

[57] Athanasios S Polydoros and Lazaros Nalpantidis. Survey of model-based reinforcement learning: Applications on robotics. *Journal of Intelligent & Robotic Systems*, 86(2):153–173, 2017.

[58] Warren B Powell. *Approximate Dynamic Programming: Solving the curses of dimensionality*, volume 703. John Wiley & Sons, 2007.

[59] Saša V Raković, Basil Kouvaritakis, Rolf Findeisen, and Mark Cannon. Homothetic tube model predictive control. *Automatica*, 48(8):1631–1638, 2012.

[60] Sasa V Raković, William S Levine, and Behçet Açikmese. Elastic tube model predictive control. In *Proceedings of 2016 American Control Conference*, pages 3594–3599. IEEE, 2016.

[61] J Richalet, S Abu el Ata-Doss, Ch Arber, HB Kuntze, A Jacubasch, and W Schill. Predictive functional control-application to fast and accurate robots. *IFAC Proceedings Volumes*, 20(5):251–258, 1987.

[62] Paolo Rocco. Stability of PID control for industrial robot arms. *IEEE Transactions on Robotics and Automation*, 12(4):606–614, 1996.

[63] Loris Roveda, Jeyhoon Maskani, Paolo Franceschi, Arash Abdi, Francesco Braghin, Lorenzo Molinari Tosatti, and Nicola Pedrocchi. Model-based reinforcement learning variable impedance control for human-robot collaboration. *Journal of Intelligent & Robotic Systems*, 100(2):417–433, 2020.

[64] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning Using Connectionist Systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.

[65] Andrei A Rusu, Matej Večerík, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. Sim-to-real robot learning from pixels with progressive nets. In *Proceedings of 2017 Conference on Robot Learning*, pages 262–270. PMLR, 2017.

[66] Tim Schneider, Boris Belousov, Georgia Chalvatzaki, Diego Romeres, Devesh K Jha, and Jan Peters. Active exploration for robotic manipulation. In *Proceedings of 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 9355–9362. IEEE, 2022.

[67] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37, pages 1889–1897. PMLR, 07–09 Jul 2015.

[68] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[69] Chao Shen, Yang Shi, and Brad Buckham. Trajectory tracking control of an autonomous underwater vehicle using Lyapunov-based model predictive control. *IEEE Transactions on Industrial Electronics*, 65(7):5796–5805, 2017.

[70] Yang Shi, Chao Shen, Henglai Wei, and Kunwu Zhang. *Advanced Model Predictive Control for Autonomous Marine Vehicles*. Springer International Publishing, 2023.

[71] Yang Shi and Kunwu Zhang. Advanced model predictive control framework for autonomous intelligent mechatronic systems: A tutorial overview and perspectives. *Annual Reviews in Control*, 52:170–196, 2021.

[72] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[73] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of 2014 International Conference on Machine Learning*, pages 387–395. PMLR, 2014.

[74] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[75] Mark W Spong and Mathukumalli Vidyasagar. *Robot Dynamics and Control*. John Wiley & Sons, 2008.

[76] Richard S Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4):160–163, 1991.

[77] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[78] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, OpenAI Xi Chen, Yan Duan, John Schulman, Filip DeTurck, and Pieter Abbeel. # Exploration: A study of count-based exploration for deep reinforcement learning. In *Proceedings of 2017 Advances in Neural Information Processing Systems*, pages 2753–2762, 2017.

[79] Rong-Jong Wai and Rajkumar Muthusamy. Design of fuzzy-neural-network-inherited backstepping control for robot manipulator including actuator dynamics. *IEEE Transactions on Fuzzy Systems*, 22(4):709–722, 2013.

[80] Steven Lake Waslander, Gabriel M Hoffmann, Jung Soon Jang, and Claire J Tomlin. Multi-agent quadrotor testbed control design: Integral sliding mode vs.

reinforcement learning. In *Proceedings of 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3712–3717. IEEE, 2005.

[81] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.

[82] Grady Williams, Andrew Aldrich, and Evangelos Theodorou. Model predictive path integral control using covariance variable importance sampling. *arXiv preprint arXiv:1509.01149*, 2015.

[83] Grady Williams, Paul Drews, Brian Goldfain, James M Rehg, and Evangelos A Theodorou. Information-theoretic model predictive control: Theory and applications to autonomous driving. *IEEE Transactions on Robotics*, 34(6):1603–1622, 2018.

[84] Grady Williams, Nolan Wagener, Brian Goldfain, Paul Drews, James M Rehg, Byron Boots, and Evangelos A Theodorou. Information theoretic MPC for model-based reinforcement learning. In *Proceedings of 2017 IEEE International Conference on Robotics and Automation*, pages 1714–1721. IEEE, 2017.

[85] Yuxin Wu and Yuandong Tian. Training agent for first-person shooter game with actor-critic curriculum learning. In *Proceedings of 2016 International Conference on Learning Representations*, 2016.

[86] Michael Wunder, Michael L Littman, and Monica Babes. Classes of multiagent Q-learning dynamics with epsilon-greedy exploration. In *Proceedings of 2010 International Conference on Machine Learning*, pages 1167–1174, 2010.

[87] Ji Yin, Zhiyuan Zhang, Evangelos Theodorou, and Panagiotis Tsiotras. Trajectory distribution control for model predictive path integral control using covari-

ance steering. In *Proceedings of 2022 International Conference on Robotics and Automation*, pages 1478–1484. IEEE, 2022.

[88] Mario Zanon and Sébastien Gros. Safe reinforcement learning using robust MPC. *IEEE Transactions on Automatic Control*, 66(8):3638–3652, 2020.

[89] Rui Zeng, Manlu Liu, Junjun Zhang, Xinmao Li, Qijie Zhou, and Yuanchen Jiang. Manipulator control method based on deep reinforcement learning. In *Proceedings of 2020 Chinese Control And Decision Conference*, pages 415–420. IEEE, 2020.

[90] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, Anind K Dey, et al. Maximum entropy inverse reinforcement learning. In *Proceedings of 2008 AAAI Conference on Artificial Intelligence*, volume 8, pages 1433–1438, 2008.