Learning Based Active Rejection of Environmental Disturbances for Underwater Robots

by

Tianming Wang

A thesis submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy

at the
Centre for Autonomous Systems
Faculty of Engineering and Information Technology
University of Technology Sydney

June 2020

Certificate of Original Authorship

I, Tianming Wang declare that this thesis, is submitted in fulfillment of the requirements for the award of Doctor of Philosophy, in the Faculty of Engineering and Information Technology at the University of Technology Sydney.

This thesis is wholly my own work unless otherwise reference or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

This document has not been submitted for qualifications at any other academic institution.

This research is supported by the Australian Government Research Training Program and the University of Technology Sydney.

Signed: Production Note: Signature removed prior to publication.

Date: 02 Jun 2020

Learning Based Active Rejection of Environmental Disturbances for Underwater Robots

by

Tianming Wang

A thesis submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy

Abstract

Underwater robots in shallow waters usually suffer from turbulent flows and strong waves. Such disturbances may frequently exceed the robot's control constraints, thus severely destabilize robot during task operation. Conventional disturbance observer and model predictive control are not particularly effective since they heavily rely on a sufficiently accurate dynamics model. Learning-based controllers are able to alleviate model dependency and achieve high computational efficiency. Learned control policies normally specialize on one dynamics model, and may not directly generalize to other models. Transfer learning offers a pathway to bridge the mismatch between different dynamics models. In this thesis, reinforcement learning algorithms are applied that enables optimal control of underwater robots under unobservable excessive time-correlated disturbances, and transfer learning algorithms are implemented for control policy adaptation under dynamics model mismatch.

History Window Reinforcement Learning (HWRL) and Disturbance Observer Network (DOB-Net) are developed for disturbance rejection control. Both algorithms jointly optimize a disturbance observer and a motion controller, and implicitly learn embedding of disturbance waveforms from motion history of robot. A modular design of learning disturbance rejection controller is also developed. A Generalized Control Policy (GCP) is trained over a wide range of disturbance waveforms, an Online Disturbance Identification

Model (ODI) exploits motion history of robot to predict the disturbance waveforms, which served as input to GCP. Together, GCP-ODI provides robust control across a wide variety of disturbances.

Transfer learning algorithms are applied to address the mismatch between a mathematical model of system dynamics developed from the fundamental principles of dynamics and an empirical model of system dynamics derived from real-world experimental data. Hybrid Policy Adaptation (HPA) is first proposed where learning a model-free policy under the empirical model is accelerated by pre-training a model-based policy with the mathematical model. Transition Mismatch Learning (TML) is then proposed that learns a compensatory policy based on the modular architecture of GCP-ODI through minimizing transition mismatch between the mathematical model and the empirical model.

Numerical simulations on a pose regulation task have demonstrated that HWRL, DOB-Net and GCP-ODI can successfully stabilize the underwater robot across a wide range of disturbance waveforms, and outperform conventional controllers and classical RL policies. Both HPA and TML achieve satisfactory control performance when deployed under the empirical model, with high sample efficiency and avoidance of initial exploratory actions.

Acknowledgements

First and foremost, I would like to express my deep gratitude to my principal supervisor Prof. Dikai Liu, for providing me with an opportunity to do a PhD and introducing me to the field of robotics. He gave me maximum freedom on research so that I can explore different and interesting topics. He always encouraged me to combine theory and practice, his enthusiasm and guidance inspired me to be a good robotics researcher.

I would also like to thank my co-supervisor Dr. Wenjie Lu for his continuous support in both research and life during my entire PhD journey. He always shared with me knowledge and experience in research methodologies and skills, and gave me many helpful feedbacks and suggestions on my projects and papers. He set an example for me on how to be a good researcher.

I would like to thank the excellent researchers I have met during my PhD study. Enormous thanks to Huan Yu, Jonathan Woolfrey, Teng Zhang, Karthick Thiyagarajan, Jiaheng Zhao, Yanhao Zhang, Yongbo Chen, Kanzhi Wu, Fang Bai for many inspiring discussions on research. I would like to thank the research engineers at the Centre for Autonomous Systems, Andrew To, Khoa Le, Thomas Hudson, Li Yang Liu and Brenton Leighton, for their generous help in experiments for my research. I would like to thank the postdoctoral researchers, Zheng Yan and Junyu Xuan, at the Centre for Artificial Intelligence, for the discussions and advice on machine learning.

Besides, I would like to thank my girlfriend for her endless patience listening to me talk about my research and encouragement whenever I got depressed.

Finally, I would like to thank my parents for their unconditional support, both financially and spiritually, throughout my whole student life.

Contents

List of Figures

List of Tables

Acronyms & Abbreviations

UTS	University of Technology Sydney
CAS	Centre for Autonomous Systems
AUV	Autonomous Underwater Vehicle
ROV	Remotely Operated Vehicle
SPIR	Submerged Pile Inspection Robot
IMU	Inertial Measurement Unit
DOF	Degree of Freedom
TCM	Thruster Control Matrix
PWM	Pulse Width Modulation
DOB	Disturbance Observer
DOBC	Disturbance Observer Based Control
RISE	Robust Integral of Sign Error
MPC	Model Predictive Control
LQR	Linear Quadratic Regulator
iLQR	iterative Linear Quadratic Regulator
EKF	Extended Kalman Filter
MDP	Markov Decision Process

POMDP	Partially Observable Markov Decision Process
TD	Temporal Difference
DQN	Deep Q-Network
DDPG	Deep Deterministic Policy Gradient
A2C	Advantage Actor Critic
TRPO	Trust Region Policy Optimization
PPO	Proximal Policy Optimization
SGD	Stochastic Gradient Descent
RNN	Recurrent Neural Network
GRU	Gated Recurrent Unit

Chapter 1

Introduction

1.1 Background

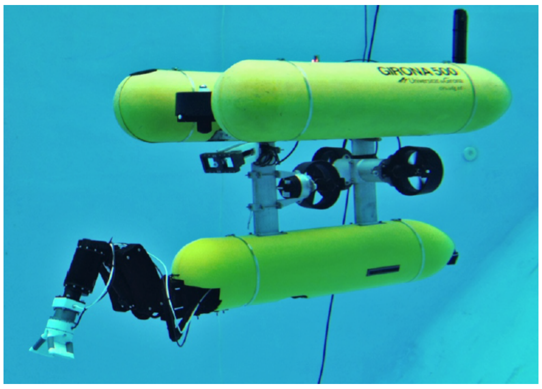
Underwater robotics has attracted an increasing interest from both research and industry in the last few decades. Recently, the applications of Autonomous Underwater Vehicle (AUV) and Remotely Operated Vehicle (ROV) to execute tasks are common, such as sea bottom survey, offshore structures monitoring, pipeline maintenance, biological samples collection and shipwreck search (Antonelli, 2018; Griffiths, 2002). The rising demand for robotic advancements in all of these ocean research and industrial fields predicates the need to better understand the ocean dynamics.

Ocean waves will displace a robot during task execution. The wave forces decay exponentially from the water surface to the seabed, and sufficient depths yield negligible disturbances (Dean and Dalrymple, 1991). Owing to this decay, as well as the considerable size and thrust capabilities of underwater robotic systems (as shown in Figure 1.1(a) and Figure 1.1(b)), the strength and changes of ocean waves are often neglected in robotic motion planning and control in deep water applications (Fernández and Hollinger, 2016). In fact, hydrodynamic drag forces offer damping effects in favor of stabilizing the robot systems.

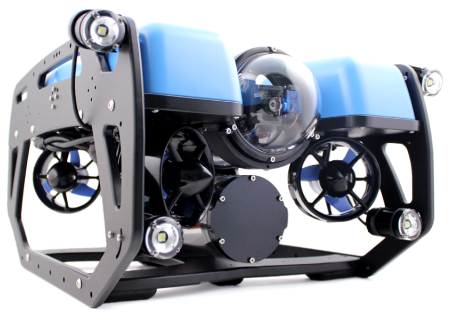
In field applications with low operational depths and turbulent wave climates, like bridge pile inspection (Woolfrey et al., 2016) and sea-ice algae characterization in Antarctica

(a) The Magnum Plus ROV from Oceaneering

(b) The Semi Autonomous Vehicle for Intervention Missions (SAUVIM) from the Autonomous System Laboratory of the University of Hawaii



(c) The GIRONA 500 AUV from Underwater Vision and Robotics Research Centre at the University of Girona



(d) The BlueROV2 from Blue Robotics

FIGURE 1.1: Examples of underwater robotic systems.

(Wang et al., 2018), this assumption can quickly break down, since shallow water environments usually accommodate only small-size robots that have limited thrust capabilities (as shown in Figure 1.1(c) and Figure 1.1(d)), and the disturbances coming from the turbulent flows are time-varying and may frequently exceed robot's thrust capabilities (such wave forces are termed as excessive disturbances throughout the thesis). As a result, increased wave disturbances inevitably hinder the stability and precision of robotic motion control, and may even destabilize the robots (Xie and Guo, 2000; Gao, 2014; Li et al., 2014).

Such problem also arises in many other applications, e.g., aerial quadrotors for surveillance in wind conditions (Waslander and Wang, 2009), and manipulators operating with constantly varying loads. In the cases of robot's actuator failure, the control capabilities may also be lower than the external disturbances.

1.2 Motivation

In the early development of disturbance rejection control, feedback control strategies are used to suppress the unknown disturbances. Examples of feedback controllers include robust control (Skogestad and Postlethwaite, 2007), adaptive control (Åström and Wittenmark, 2013; Lu and Liu, 2017, 2018), optimal control (Bertsekas et al., 1995), sliding mode control (SMC) (Edwards and Spurgeon, 1998), H-infinity control (Doyle et al., 1989), etc. However, these methods often assume that the system deals with bounded disturbances that are sufficiently small, thus fail to guarantee system stability under disturbances exceeding control constraints (Ghafarirad et al., 2014; Gao and Cai, 2016).

One improvement to the above approaches is to add a feedforward compensation term based on disturbance estimation (Yang et al., 2010; Chen et al., 2016). Various disturbance estimation and attenuation methods have been proposed and practiced, such as DOB (Ohishi et al., 1987; Chen et al., 2000; Umeno et al., 1993; Umeno and Hori, 1991), Unknown Input Observer (UIO) in Disturbance Accommodation Control (DAC) (Johnson, 1968, 1971), and Extended State Observer (ESO) (Han, 1995; Gao et al., 2001). DOB (Chen et al., 2000) and related methods have been widely investigated and applied in various industrial processes for decades. The main objective of the use of DOB is to estimate the unknown disturbances from measurable variables, without additional sensors. Then, a control action can be taken, based on the disturbance estimation, to compensate for the influence of the disturbances, which is called Disturbance Observer Based Control (DOBC) (Chen et al., 2016).

However, conventional DOBC does have some limitations when addressing excessive time-correlated disturbances. The first limitation is that DOB normally requires an accurate system model, which might not be readily available for underwater robots due to complex hydrodynamics. In this case, there exist both external disturbances and internal model uncertainties in the control process, and model uncertainties are lumped with external disturbances, then an observation mechanism is adopted to estimate the total disturbances. This leads to the change of the original properties for some disturbances, such as harmonic ones. Secondly, DOB is only capable of dealing with constant or slow time-varying disturbances, as evidenced by its proof of convergence (Chen et al., 2016). While the wave disturbances usually change continuously and rapidly, such case is beyond the capability of conventional DOB. Thirdly, even with a sufficiently accurate estimate of disturbances at current timestep, optimal control solution is still difficult to achieve, since disturbances exceeding control constraints cannot be well rejected through feedback regulation only. Sun and Guo (2016) proposed a neural network approach to construct DOBC. In contrast to conventional DOBC methods, the primary advantage of the proposed method is that the model uncertainties are approximated using a Radial Basis Function Neural Network (RBFNN) technique, and not regarded as part of the disturbances. Then the external disturbances can be separately estimated by a conventional DOB. But they still failed to consider the control constraints, leading to poor control capabilities under excessive disturbances. Thus, in order to achieve optimal control performance under disturbances that frequently exceed the control constraints, the behaviours of an underwater robot need to be optimized over a future time horizon considering time correlation of the disturbance signals.

To this end, Model Predictive Control (MPC) (Garcia et al., 1989; Camacho and Alba, 2013; Fernández and Hollinger, 2016) is often applied due to its constraint handling capability. This method can achieve approximately optimal control performance even under practical constraints, through optimizing system behaviours over a certain time horizon, sometimes even sacrificing instant performance for better overall performance (Gao and Cai, 2016). The MPC requires a prediction model of the system to optimize future behaviours. This model includes not only the robot dynamics, but also the predicted disturbances over next optimization horizon. Thus, researchers have developed a compound control method consisting of a feedforward compensation part based on the DOB and a feedback regulation part based on the MPC (DOB-MPC) (Maeder and Morari, 2010; Yang et al., 2010, 2011; Liu et al., 2012; Yang et al., 2014; Dirscherl et al., 2015). The DOB can provide estimate of disturbances, with Auto-Regressive Moving Average (ARMA) used to predict future disturbances based on past estimations. Then the MPC can be employed based on the given system dynamics and the disturbance prediction. However, the performance of both DOB and MPC relies on the accuracy of given system dynamics model. The requirement for online optimization at each timestep leads to low computational efficiency. Besides, such separated modeling and control optimization process might not be able to produce models and control signals that jointly optimize robot performance, as evidenced in Brahmbhatt and Hays (2017); Karkus et al. (2018).

In contrast, RL has drawn a lot of attention in learning optimal controllers for systems that are difficult to model accurately. RL (Sutton and Barto, 2018) is also known as adaptive dynamic programming and neural computing. It is a trial-and-error approach that allows to find an optimal sequence of commands without any prior assumption about the world and any requirement of an explicit system model, and can naturally adapt to uncertainties and noises in the real system. With recent advances in deep neural networks, RL is now able to solve practical problems. Through using neural networks, RL is able to construct the disturbance estimation and attenuation parts together and achieve joint optimization of both components, instead of conventional separate modules and hand-designed features transmission. In the meantime, neural network controllers also enable high computational efficiency due to their fast forward propagation.

RL is superior in solving a Markov Decision Process (MDP), where the future states of the process depend only on the present state, not on the past states. However, the excessive disturbances are not appropriate to be regarded as noises any more, since the state transition of the robot system is heavily affected by the external disturbances besides the current state and action, leading to a violation of Markov property. Normally, when there are unobservable or hidden states to the agent, the problem will become a Partially Observable Markov Decision Process (POMDP). Thus, the first research question in this thesis is how to find a state space to better represent the disturbances, so that the problem of excessive disturbance rejection can be formulated into a POMDP, then how to learn an effective control policy in POMDP.

RL is able to successfully solve problems when being trained directly on desired dynamics model. Sometimes training samples are expensive to obtain on the desired model. For example, high sample complexity of deep RL algorithms often leads to long training time during their direct application to physical systems. Learning from scratch involves many exploratory actions, which real robots usually cannot withstand and may endanger both the agent and its surroundings. At other times, the desired model may change over time in an unpredictable way during operation. These variations in the dynamics model may include variable friction coefficients, actuator failures, or differences in mass of an object to be manipulated. In such cases, it is difficult to implement pure learning procedures on the desired model. A promising approach is to make an initial guess of the desired model, such as a simulated counterpart of the real robot or a static model of the system before operation, which is easy to get in most cases, then optimize a policy based on this model. However, because of the inaccurate replication of the desired dynamics, initially learned policies usually cannot be directly applied on the desired system model. Transfer learning offers a pathway to bridge the mismatch between different dynamics models. Thus, the second research question in this thesis is how to transfer disturbance rejection control under dynamics model mismatch.

1.3 Contributions

1.3.1 Learning Optimal Control under Excessive Time-Correlated Disturbances

The first research question is how learning algorithms can be developed for an optimal control problem of underwater robots in shallow and turbulent water, where unobservable time-correlated wave forces exist and may frequently exceed the robot control constraints. An initial idea to tackle this problem is to jointly learn a disturbance observer and a motion controller, by virtue of the powerful fitting ability of deep neural network models. Joint learning, also known as end-to-end learning, defines a learning process in which all of the parameters are trained jointly, instead of step by step. A History Window Reinforcement Learning (HWRL) is presented to characterize the disturbed system dynamics model as a multi-order Markov chain. The unobservable disturbance waveforms are assumed to be encoded in the most recent state and action history of the system, and the embedding of disturbance waveforms is learned within policy optimization using DDPG. In contrast to classical RL formulation that chooses action only based on current state, HWRL is trained to generate optimal control actions based on a fixed length of past states and actions along with the current system state.

Due to the difficulty in determining optimal history length through HWRL, RNN is then applied to automatically learn how much past experience should be explored to achieve optimal performance. Learning algorithm called Disturbance Observer Network (DOB-Net) is then proposed. DOB-Net explicitly constructs a sequential structure of a disturbance observer network and a motion controller network. The observer network is built via RNN, through imitating conventional DOB mechanisms. This network surpasses the conventional one in functionality, by encoding unobservable disturbance waveforms into hidden state of RNN, instead of estimating only current disturbance value. Also, the observer network is more robust to the model uncertainties and the time-varying characteristics of the external disturbances, compared with the conventional DOB. Given the encoded disturbance waveforms as input, the controller network is able to actively reject the unobservable disturbances. The observer network and the controller network are jointly optimized by A2C.

This joint learning might achieve an optimized information transmission between the observer and the controller, compared with traditional hand-designed features. In most cases, training such a large network leads to high sample complexity and difficulty of convergence to optimum. Then we seek a modular design of learning policies for disturbance rejection control, we believe that using the decoupled moderate-sized networks instead of a large network trained by RL in an end-to-end mode, might mitigate the learning difficulty and thus improve the sample efficiency. The proposed algorithm is composed of two main parts, namely a Generalized Control Policy (GCP) and an Online Disturbance Identification Model (ODI). The idea is to use a large amount of simulated data to precalculate many possible disturbance waveforms that the robot may encounter during operation, and learn a control policy (i.e., GCP) for each set of waveforms in a parameterized space. Then, given the observed past states and actions of the system, ODI is able to predict the disturbance waveforms in accordance with the current environment. The predicted disturbance waveforms are then provided as input to GCP together with the current system state to produce expected control action. The combined system, GCP-ODI, is a robust control policy that can be executed under a wide variety of disturbances.

The learning algorithms are evaluated on a pose regulation task in simulation, where simulated disturbances are constructed as a superposition of multiple sinusoidal waves.

During training, the amplitudes, frequencies and phases of the disturbances are randomly sampled from given distributions in each episode. As a result, both HWRL and DOB-Net can successfully stabilize the robot under excessive disturbances, and outperform conventional controllers and classical RL policies. Also, DOB-Net achieves even better performance compared with HWRL.

1.3.2 Transferring Disturbance Rejection Policy under Dynamics Model Mismatch

Another focus of the thesis is to apply transfer learning techniques to adapt a learned control policy to the mismatch in dynamics model. In this work, we formulate a transfer learning problem where the source and target tasks differ in system dynamics, the former defines a mathematical model of an underwater robot developed from the fundamental principles of dynamics, the latter applies an empirical model of an underwater robot derived from real-world experimental data.

Hybrid Policy Adaptation (HPA) is introduced, where learning a model-free policy under the empirical model is accelerated by pre-training a model-based policy with the mathematical model, so that this algorithm combines high sample efficiency of model-based learning with high task-specific performance of model-free learning. Specifically, a model-based controller, iterative Linear Quadratic Regulator (iLQR), is applied with the mathematical model to produce successful trajectories. Then supervised learning is used to optimize a control policy to imitate the behaviours of iLQR from these trajectories. Two scenarios of model-based policy optimization are provided. The first scenario excludes simulated disturbances in the mathematical model, then history data is not necessary for decision making and thus the model-based policy only takes the current system state as input. The other one includes the simulated disturbances, then the model-based policy requires the same formulation to process history data with the following model-free learner. The policy adaptation from source task to target task can be implemented by either training a compensatory policy in parallel with the model-based policy, or using the model-based policy to initialize a model-free policy, which is further fine-tuned under the empirical model.

HPA is able to bridge the model mismatch through a small amount of adaptive training under the empirical model. However, this algorithm treats the external disturbances as parts of the system dynamics and deals with them together during adaptation, without taking full advantage of the characteristics of the disturbances. To further improve transfer performance, a learning algorithm, called Transition Mismatch Learning (TML), is developed. This algorithm addresses model discrepancy based on the modular architecture of GCP-ODI. A control policy is first learned using GCP-ODI under the mathematical model. Then an additional compensatory policy is learned through simultaneously maximizing control performance measure and minimizing mismatch of transitions predicted by the mathematical model and the empirical model. These two policies run in parallel and the combined control action is used to control the system under the empirical model. Thus, the empirical model is forced to behave like the mathematical model as if there is no model mismatch. In this way, ODI is able to predict a set of waveforms that best fit the disturbances of the empirical model, although it is trained exclusively on the mathematical model. Then the disturbances are separated from the system dynamics and their mismatch between two dynamics models is nearly eliminated by ODI. Thus the total model mismatch is reduced, leading to reduced learning difficulty and improved sample efficiency for policy adaptation. In addition, middle layer features, instead of final network outputs, of GCP and the compensatory policy are combined to generate control actions under the empirical model, in order to mitigate the limitation to compensation effect under control constraints.

The transfer learning algorithms are evaluated using the mathematical model and the empirical model both in simulation. The goal is to verify the effectiveness of transfer learning algorithms when meeting unknown mismatch in dynamics model, and the successful experiences will be easily extend to real-world environments. As a result, a control policy trained on the mathematical model is still able to successfully reject the disturbances and stabilize the robot on the empirical model through a small amount of adaptive training. In addition, transfer learning using TML achieves better sample efficiency compared with using HPA.

1.4 Thesis Outline

The following content of this thesis is divided in to 7 chapters. Chapter 2 covers a review of related work in the current literature, starting with conventional control theories for anti-disturbance control. Also, a summary of existing reinforcement learning algorithms in POMDP and transfer learning in reinforcement learning under mismatch in system dynamics is presented, their limitations under our problem settings are discussed as well in this thesis. The relationship between proceeding Chapter 3 to Chapter 6 is shown in Figure 1.2. In Chapter 3 and Chapter 4, a detailed problem formulation is provided, including descriptions of system dynamics and control objective. Then, two joint learning algorithms, HWRL and DOB-Net, and one modular learning algorithm, GCP-ODI, for optimal control under excessive time-correlated disturbances are introduced. Their network architectures and training processes are presented as well. The results are demonstrated on a pose regulation task in simulation that validate the effectiveness of these algorithms and the advantages over conventional controllers and classical RL policies. Chapter 5 and Chapter 6 are elaborated on the basis of Chapter 3 and Chapter 4, respectively. A formulation of transfer learning is given first, including the definition of source and target tasks. Then, the details of two transfer learning algorithms, HPA and TML, are explained. The construction process of a neural network dynamics model of a real underwater robot is also described and the model is used to define the target task. The performance of transfer

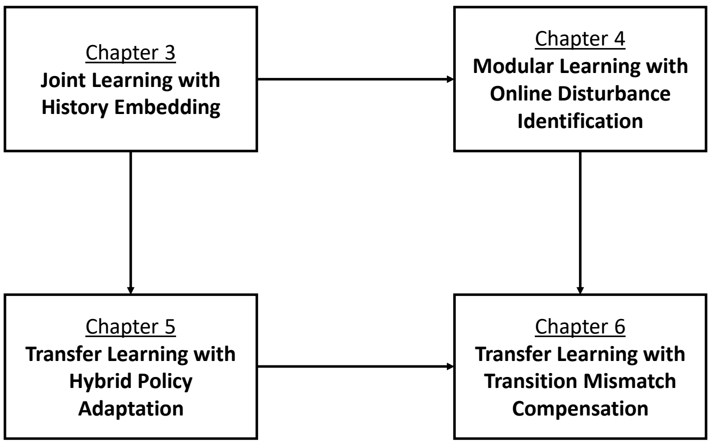

Chapter 3
Joint Learning with History Embedding
Chapter 4
Modular Learning with Online Disturbance Identification
Chapter 5
Transfer Learning with Hybrid Policy Adaptation
Chapter 6
Transfer Learning with Transition Mismatch Compensation

FIGURE 1.2: The flow of ideas in this thesis.

learning algorithms are tested again on the pose regulation task, and shows their advantages over learning from scratch without transfer. Chapter 7 concludes the whole thesis and discusses existing limitations of proposed algorithms and potential improvements for further research.

1.5 Publications

- **Tianming Wang**, Wenjie Lu, Huan Yu and Dikai Liu, "Transfer Learning with Online Identification and Rejection of Excessive Time-Correlated Disturbances," ready to submit to *IEEE Transactions on Industrial Electronics (TIE).*
- **Tianming Wang**, Wenjie Lu, Zheng Yan and Dikai Liu, "DOB-Net: Actively Rejecting Unknown Excessive Time-Varying Disturbances," *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.
- **Tianming Wang**, Wenjie Lu and Dikai Liu, "Excessive Disturbance Rejection Control of Autonomous Underwater Vehicle using Reinforcement Learning," *Australasian Conference on Robotics and Automation (ACRA)*, 2018.
- **Tianming Wang**, Wenjie Lu and Dikai Liu, "A Case Study: Modeling of A Passive Flexible Link on A Floating Platform for Intervention Tasks," *IEEE World Congress on Intelligent Control and Automation (WCICA)*, 2018, pp. 187-193.

Chapter 2

Review of Related Work

2.1 Anti-Disturbance Control

Since disturbances are inevitable in most practical processes, the research problem of anti-disturbance control has always been a hot topic in the field of control theory. Such control techniques can be mainly divided into disturbance attenuation schemes and disturbance rejection methods.

2.1.1 Disturbance Attenuation

Two branches of the disturbance attenuation approaches are stochastic control theory (Åström, 2012; Guo and Wang, 2010) and robust control theory (Zhou and Doyle, 1998). In stochastic control, Gaussian control theory is proposed, such as minimum variance control and Kalman filter, to control the output variance to optimize system behaviours, where the disturbances are considered to be Gaussian noises. When the system under consideration has non-Gaussian noises, model uncertainties or other types of disturbances, the performance of the Gaussian control methods may be reduced. According to statistical information set or probability density function, a class of stochastic distribution control methods is presented for the problem of non-Gaussian stochastic control (Guo and Wang, 2010). The stochastic control theory is based on the assumptions of known statistical characteristics of the noises, but it is usually not satisfied in the actual processes. To overcome these shortcomings, the robust control problems have been widely investigated in the last few decades. The robust control does not require the assumptions of statistical characteristics about the noises, in which the norm-bounded uncertainties and disturbances can be handled. Amongst the robust control approaches, H2 and H-infinity control have widespread applications in attenuating the influence of disturbances on the reference output at a desired level (Zhou and Doyle, 1998; Van Der Schaft, 1992; Lin and Lin, 2001). Nevertheless, the robust control approaches are conservative for the disturbances represented by norm-bounded variables.

2.1.2 Disturbance Rejection

The canonical disturbance rejection methods include disturbance observer based control (DOBC) (Chen et al., 2016), active disturbance rejection control (ADRC) (Han, 2009), output regulation theory (Huang, 2004), internal model control (Garcia and Morari, 1982), and embedded model control (Canuto and Musso, 2006). The internal model control methods are applied to compensate for disturbance inputs under neutral stability conditions for linear systems. The classic output regulation methods can be used for the disturbances represented by exogenous models (Isidori and Byrnes, 1990; Marino and Tomei, 1996; Byrnes et al., 2012). But when the models or parameters of the exogenous models are unknown, it is difficult to obtain viable disturbance compensation approaches or output regulation results. In terms of nonlinear systems with uncertainties, the internal model control then applies adaptive control techniques to solve the problem of output regulation (Huang and Chen, 2004; Xian et al., 2003; Xi and Ding, 2007). In addition, to deal with the disturbances with unknown parameters or model uncertainties, a disturbance rejection method is developed based on adaptive estimation (Nikiforov, 2001; Bodson et al., 2001). The output regulation theory in the nonlinear cases employs a Francis–Isidori–Byrnes nonlinear partial differential equations (PDE) to get a control solution. For the ADRC, although such methods have better industrial applications prospects, their stabilities are in lack of strict theoretical support (Han, 2009).

2.2 Reinforcement Learning in Partially Observable Markov Decision Processes

Besides the traditional control strategies, we also look at the data-driven methods like RL for some inspirations. In recent years, with the powerful representation learning and function approximation capabilities of deep neural networks, deep learning has seen a rise and then accelerated the development of RL. Deep RL algorithms based on Q-learning (Mnih et al., 2015; Oh et al., 2016; Gu et al., 2016a), policy gradients (Schulman et al., 2015a; Gu et al., 2016b), and actor-critic methods (Lillicrap et al., 2015; Mnih et al., 2016; Schulman et al., 2015b) have been shown to learn complex skills in high-dimensional state and action spaces, including video game playing, quadruped robot locomotion, autonomous driving, and dexterous manipulation.

However, real-world control problems rarely feature the full state information of the system. That is to say, the Markov property is often invalid in real-world environments. Then a POMDP better describes the dynamics of the real-world environments through explicitly recognizing that the agent only observes partial glimpses of the underlying system state. Existing solutions to a POMDP typically maintain a belief state over the world state given the observations so far. This method has shortcomings in model dependency and computational cost during belief update (Kaelbling et al., 1998; Shani et al., 2013). A more frequently applied method is RNN (Wierstra et al., 2010; Hausknecht and Stone, 2015; Heess et al., 2015; Mnih et al., 2016; Oh et al., 2016).

2.2.1 Recurrent Neural Networks in Reinforcement Learning

Using RNN to represent policies is a popular approach to handle partial observability (Hausknecht and Stone, 2015; Wierstra et al., 2007). The idea being that the RNN is able to retain information from observations further back in time, and incorporate this information into predicting better actions and value functions, thus performing better on tasks that require long term planning. Particularly, at each timestep, a RNN policy takes as input a current state and a hidden vector, then produces an action and the next hidden vector. The hidden vector is then received as input to the network at the next timestep.

Since the hidden vector contains past processed information, the action is a function that depends on all of the previous states.

One of the earliest works is to apply RNN in Deep Q-Network (DQN) framework, which enables the policy to better handle POMDP by learning long-term dependencies. Hausknecht and Stone (2015) introduced a deep recurrent Q-network (DRQN) that was capable of successfully estimating velocities in training video game player, where recurrent connections create an effective way to conditionally operate on previous observations that are far away in time. Then attention mechanism was introduced for further improvements, leading to a deep attention recurrent Q-network (DARQN) (Sorokin et al., 2015), that builds additional connections between recurrent units and lower layers. Attention mechanism allows the network to focus on the most important part of the next input, so that DARQN outperforms DRQN and DQN on the video games that require long-term planning. But for the games that require quick responses, DQN is better than DRQN and DARQN, since Q-values may fluctuate faster.

These policies are able to solve tasks that require memory by loading sequence of states (Wierstra et al., 2007). However, most of them only considered the state-only history, which, for example, has been used for estimating velocities in training video game player (Mnih et al., 2015), and modeling pedestrians' kinematics (Alahi et al., 2016). While this is not sufficient to observe hidden information in terms of action. Sutskever et al. (2014) presented a novel usage of RNN, where the input to the RNN comes from the previously predicted output. This architecture gives rise to an idea of using both past states and actions in the recurrent networks, where the policy produces an action, given both current system state as well as previously executed action as the input to RNN. Such approaches encode additional dynamics information besides kinematics one, thus enable the capability of observing complete action information. The successfully applications have been developed based on policy gradients (Wierstra et al., 2010) and actor-critic methods (Heess et al., 2015; Mnih et al., 2016).

2.2.2 Meta-Reinforcement Learning

Meta-Reinforcement Learning (Meta-RL) (Finn et al., 2017; Nagabandi et al., 2018a; Rakelly et al., 2019) defines a framework which leverages data from previous tasks to acquire a learning procedure that can quickly adapt to new tasks using only a small amount of experience. The tasks are considered as MDP or POMDP drawn from a task distribution with varying transition functions or varying reward functions. Nagabandi et al. (2018a) and Rakelly et al. (2019) both considered adaptation to current task setting using past transitions or context. Rakelly et al. (2019) considered adaptation at test time in meta-RL as a special case of RL in a POMDP by including the task as the unobserved part of the state, and learned a probabilistic latent representation of prior experience to capture uncertainty over the task. Nagabandi et al. (2018a) defined a more fluid notion of task, where each segment of a trajectory can be regarded as a different task, since variations in system dynamics or environmental factors may occur at any time. Their meta-RL approach achieved online model adaptation in dynamic environments using M past timesteps to predict next K timesteps.

However, these works either assume each task is an invariable MDP or POMDP (Finn et al., 2017; Rakelly et al., 2019), or assume the environment is locally consistent during a rollout (Nagabandi et al., 2018a), which may not be suitable for our problem due to the rapid time-varying characteristic of the wave forces. Also, these disturbances are better described as functions of time, instead of just uncertainties in the transition function. In that case, the external disturbances (or variations in transition function) for two consecutive samples are not independent and identically distributed (i.i.d.). Current meta-RL formulation has not clearly indicated an appropriate solution for this time-correlated variations in the transition function, it is also not clear what kind of state space is appropriate to formulate this specific problem in POMDP. A potential idea is to characterize the ever-changing transition function as a multi-order Markov chain, where the enlarged state space may contain more information about the underlying state in POMDP. This naturally leads us to the history window formulation and recurrent networks in RL, and this is exactly one of the focuses and contributions of our work.

2.3 Transfer Learning in Reinforcement Learning

Although the RL algorithms can learn complex tasks in high-dimensional state and action spaces, it is almost impossible to directly deploy a RL agent on real-world systems, due to the high sample complexity. There are a number of RL-based approaches focusing on improving sample efficiency. It is usually effective to engage in reward shaping (Laud, 2004; Grzes and Kudenko, 2008), the problem is that this requires heavy domain-specific engineering. Researchers also seek to learn good latent representations to construct the states with meaningful information, especially when the original set of features is too big (Yarats et al., 2019; Du et al., 2019). In the case of off-policy learning, not all samples are useful in that they are not part of the distribution that the agent is interested in. Importance sampling (Tang and Abbeel, 2010; Wang et al., 2016) is a technique to filter these samples. It has been applied in Trust Region Policy Optimization (TRPO) (Schulman et al., 2015a) and Proximal Policy Optimization (PPO) (Schulman et al., 2017) to achieve better sample efficiency.

Furthermore, the knowledge previously obtained from related tasks can be used to accelerate learning process of RL, such methods include: transfer learning (Pan and Yang, 2009), curriculum learning (Bengio et al., 2009), multitask learning (Caruana, 1997), etc.. Among them, transferring policy from one task to another (Taylor and Stone, 2009), especially from learning in physical simulators to adapting on real robots, has aroused great interest. Compared with other approaches, transfer RL provides a more simple but effective way that directly moves most of the training burden to other different but related tasks. Thus, the required training samples on target task can be reduced, without considering about optimizing the total amount of the training samples. The most simple idea is to use identical network architecture for both simulation and real environment (Zhu et al., 2017). More sophisticated learning process adds new layers when transferring to the new task, in the meantime freezes old layers, thus avoids the problem of catastrophically forgetting (Rusu et al., 2016a,b). There are also other methods including domain adaptation that learns aligned visual representations between synthetic and real-world images (Tzeng et al., 2020; Bousmalis et al., 2018).

2.3.1 Model-Based Reinforcement Learning

Much of the previous work has focused on classical system identification (Lennart, 1999; Giri and Bai, 2010), which provides a framework for finding accurate system models, then simplifying the design of robot controllers in the real world. In the context of RL, these methods are often referred to as model-based RL (Deisenroth et al., 2013), that is, data from actual policy execution is used to fit a transition model and then used to learn a control policy without directly interacting with the real-world scene. Such methods reduce the number of samples compared to purely model-free RL, which generally requires tremendous data to encode the objective function and the transition model. The trained policy can be directly applied or adapted to the real world. Thus, the model-based RL can be regarded as one kind of transfer learning where the learned dynamics model defines the source task and the real-world scene defines the target task.

Various model-based RL methods have been proposed (Deisenroth and Rasmussen, 2011; Kuvayev and Sutton, 1996; Forbes and Andre, 2002; Hester and Stone, 2017; Jong and Stone, 2007; Sutton, 1991), and applied successfully on both simulated and real-world robots, such as inverted pendulums (Deisenroth and Rasmussen, 2011), manipulators (Brauer, 2012), and legged robots (Schmidt and Lipson, 2009). There are also various model-based RL algorithms presented for learning dynamics model and control policies using visual information (Oh et al., 2015; Watter et al., 2015; Wahlström et al., 2015a), the learned dynamics model is able to encode high-dimensional visual observations into low-dimensional features with the power of autoencoders (Wahlström et al., 2015b). After getting a sufficiently precise dynamics model of the system, it is then easy to implement robotic control using raw image data (Finn et al., 2016). In addition, the neural network dynamics models also enable explorations of the vision-based agent to be conducted without interactions with the real environments (Stadie et al., 2015).

In fact, system identification is usually interleaved with policy optimization (Abbeel and Ng, 2005; Gevers et al., 2006; Bongard and Lipson, 2005; Nagabandi et al., 2018b). In other words, additional policy execution data is collected through alternating between collecting data with the current model and retraining the model with the aggregated data. This data aggregation procedures improves performance by alleviating the mismatch between the distribution of the trajectory data and that of the model-based controller, and such an iterative process is able to converge to the optimal policy.

However, their performance could be unsatisfactory if the models are not accurate enough. Thus, combinations of model-based and model-free RL have been proposed, such that the systems can quickly achieve moderately proficient behaviours, and then slowly achieve near-optimal performance. Kumar et al. (2018) and Koryakovskiy et al. (2018) both proposed to use model-free RL to learn a compensatory control signal on top of a model-based controller. The model-based controller can speed up learning of model-free RL and avoid risky exploratory actions, and the model-free learner can enhance the control performance by compensating the model-plant mismatch. The model-based controllers, such as MPC or Linear Quadratic Regulator (LQR), may involve solving optimization problems, which is much slower than the forward propagation of neural network policies. Nagabandi et al. (2018b) also used a model-based controller, but they adopted supervised learning to train an imitation policy to mimic the model-based controller, and then used this imitation policy as an initialization for the model-free learner.

A number of prior works have also sought to incorporate dynamics models into model-free value estimation. Gu et al. (2016a) proposed model-based acceleration (MBA), where imagination rollouts are generated from the learned model, and then used as additional training data for a parameterized value network. Feinberg et al. (2018) presented model-based value expansion (MVE) that uses a dynamics model to simulate a short-term horizon when estimating Q-value target, instead of using bootstrapping. The improved value estimate provides enough signal for faster learning for actor-critic methods. Moreover, some works have further studied the connection between model-free and model-based RL. Boyan (1999) showed that one can extract a model from a value function when using a tabular representation of the transition function. Pong et al. (2018) introduced a goal-conditioned value function called temporal difference model (TDM), that can be trained efficiently with model-free learning and can then be used with an MPC-like method to accomplish desired tasks.

2.3.2 Domain Randomization

Another area of research is domain randomization, where the differences between the source and target tasks are modeled as the variabilities in the source task. Therefore, the source task can be designed to be as diverse as possible in the simulation in order to better generalize the trained policy to unfamiliar system dynamics or environmental factors in the real world. Randomization in the field of vision has been used to transfer vision-based policies directly from simulation to real-world, and does not require real images during training (Sadeghi and Levine, 2016; Tobin et al., 2017). Their systems use only low-fidelity rendering and randomizes scene attributes (such as lights, textures, and camera placement) to simulate the differences in visual appearance. Furthermore, randomized dynamics have also been used to design controllers that are robust to model uncertainties in system dynamics. Peng et al. (2018) proposed to learn a policy in simulation through randomizing the physical parameters of the environment, then transfer the learned policy to a real robot for a puck pushing task. Andrychowicz et al. (2020) proposed to train dexterous manipulation skills of a robotic hand using randomization of physical properties and object appearance in simulation.

However, sometimes it can be difficult for transfer learning using domain randomization, since this technique usually needs tedious manual fine-tuning and a significant expertise to design the distributions of simulation parameters (Chebotar et al., 2019). Thus, system identification and domain randomization have also been combined (Tan et al., 2018; Lowrey et al., 2018; Antonova et al., 2017). These approaches address the problem of automatically learning the distributions of simulated parameters to avoid manually design procedures, thus improving the transfer learning of policies. Rajeswaran et al. (2016) and Chebotar et al. (2019) applied this framework to iteratively learn a policy over an ensemble model and used data from the target task to adjust model distributions. The results showed that policies can be successfully transferred with only a few iterations of simulation updates using a small number of real robot trials.

2.4 Summary

It can be found from the literature review that there exist rich research works in many of the related fields to the anti-disturbance control and the transfer RL. In spite of this, there are still evident limitations and gaps with the existing methodologies.

In terms of the anti-disturbance control problem, current methods can be divided into control theory and reinforcement learning. In addition to providing satisfactory stability, the required anti-disturbance controller is also supposed to have a simple structure, fast computation speed, and robustness to unmodeled dynamics, parameter changes, and process noises. Some of the existing control methods posses simple design, but might not achieve desirable system performance. In contrast, other methods may provide perfect stability, but the controller structure is too complicated or computationally intensive to be applied online in practical processes. Furthermore, their applications for different systems may be limited by some strict assumptions of the models. Using RL to solve this control problem leads to a formulation of POMDP, but the presence of external disturbances causes some differences from the generic POMDP. That is, even given the full observability of the external disturbances at the current timestep, it is still difficult to formulate a transition function to predict the next disturbances from the current system state, external disturbances as well as control action, since the variation in the external disturbances is uncertain. Existing learning algorithms have not addressed this problem.

For transferring policy between different transition dynamics, model-based RL and domain randomization approaches have been widely investigated. Although neural-network-based dynamics model can make reasonable predictions over a future time horizon without physical interaction (Chiappa et al., 2017), the success of model-based RL still heavily depends on the quality and quantity of the real-world experimental data. For contact-rich or highly dynamic tasks, it typically requires large amounts of high quality real-world data to learn an accurate dynamics model and thus a control policy. In order to promote the adoption of neural network models in model-based RL, finding strategies to improve their sample efficiency is necessary. As for domain randomization, this kind of techniques generally restricted to only low-dimensional dynamics models. When the real-world dynamics become more complicated, the selected parameterization of the dynamics model might not well represent it. Thus domain randomization can be difficult to apply in sim-to-real transfer in most cases.

In conclusion, most existing methodologies, including both control theory and reinforcement learning, have not considered a situation where the external disturbances possess excessive strengths with respect to the control constraints, in the meantime are time-correlated in behaviours. Accordingly, these methods have not provided specific and effective solutions for such problems. In this thesis, we propose learning-based algorithms to address the research questions regarding both stability control under excessive disturbances and policy adaptation under model uncertainties. On one hand, we adopt learning-based controllers that make use of the time-correlated characteristic of wave forces and explore the underlying behaviours of disturbances for optimizing system behaviours over a future time horizon. In the meantime, these controllers also achieve high computational efficiency and model independence (Chapter 3 and Chapter 4). On the other hand, we employ transfer learning algorithms that the learned policy for disturbance rejection control can be efficiently adapted to model variations either in internal system dynamics or in external disturbances (Chapter 5 and Chapter 6).

Chapter 3

Joint Learning of Disturbance Rejection Control with History Embedding

The wave forces in shallow water environments present great challenges for stabilization control of underwater robots due to their excessive strengths. When using RL to deal with such unobservable disturbances, the problem cannot be defined as a MDP, since the state transition does not only depend on the current state and action, but also heavily on the external disturbances. Due to the strong time correlation within the wave forces, the behaviours of disturbances might be learned for the prediction of future disturbances and then for better control.

Thus, we propose RL algorithms for control under excessive time-correlated disturbances, through constructing an observation mechanism for the disturbance behaviours, then determining a stability controller based on this information. Two learning schemes are considered, joint learning and modular learning. The joint learning scheme seeks to learn a disturbance observer and a motion controller in an end-to-end mode, in order to achieve an optimized information transmission between modules. While the modular learning scheme explicitly decouples these two modules and trains them separately, so that attaining reduced learning difficulty and improved sample efficiency.

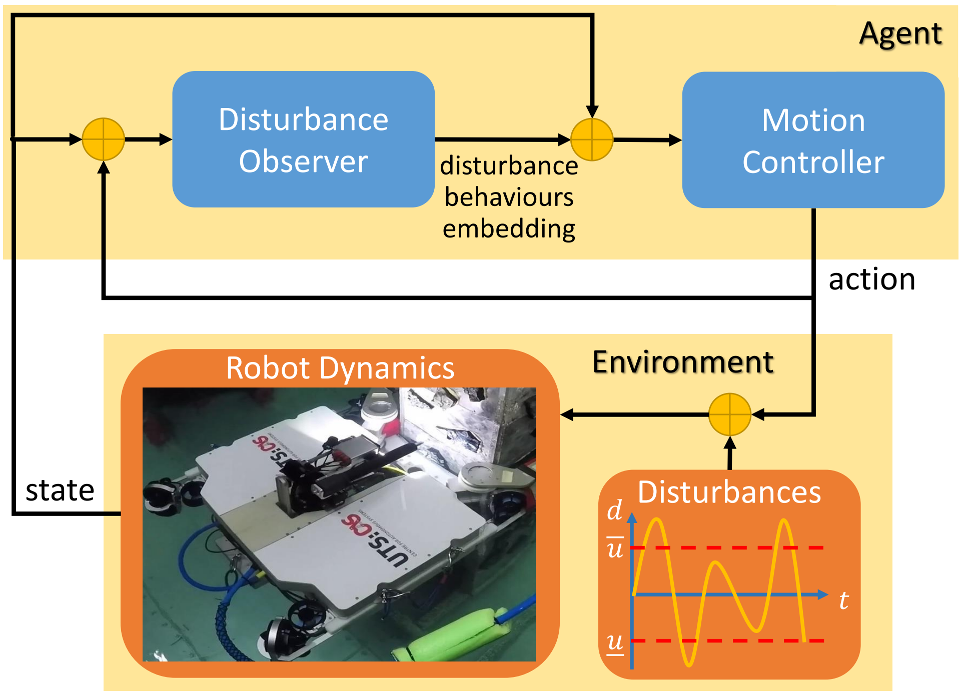


FIGURE 3.1: Working flow of joint learning algorithms for disturbance rejection control ($\overline{\boldsymbol{u}}$ and $\underline{\boldsymbol{u}}$ are the robot control constraints).

In this chapter, the joint learning algorithms, HWRL and DOB-Net, are introduced, along with their network architectures and training process. A general working flow of these algorithms are shown in Figure 3.1. The evaluation results are demonstrated on a pose regulation task in simulation. Modular learning design will be discussed in Chapter 4.

3.1 Problem Formulation

3.1.1 System Dynamics

Throughout this thesis, SPIR is used as the robot system for algorithm development and testing. Consider an nonlinear time-invariant system to represent the dynamics of an underwater robot in the form of

$$\dot{\boldsymbol{x}}\left(t\right)=f\left(\boldsymbol{x}\left(t\right),\boldsymbol{u}\left(t\right)\right), \tag{3.1}$$

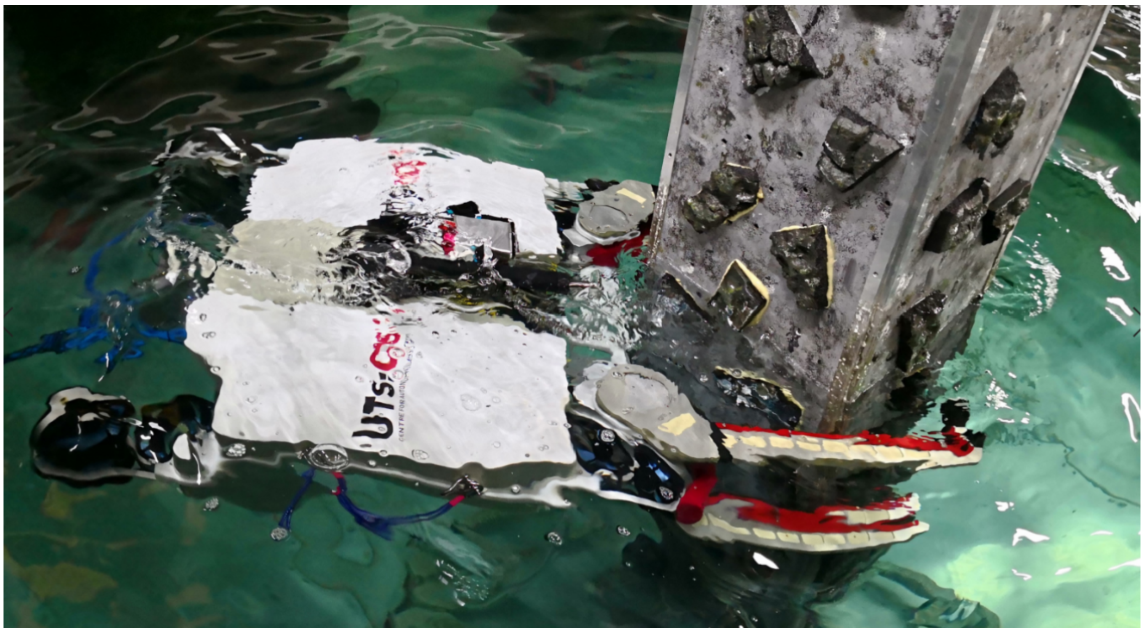


FIGURE 3.2: SPIR is designed at CAS of UTS, to operate in shallow bathymetry to clean infrastructure.

with the detailed version as

$$\begin{aligned} \boldsymbol{M}\dot{\boldsymbol{\nu}} + \boldsymbol{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{D}_{RB}(\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{g}_{RB}(\boldsymbol{\eta}) &= \boldsymbol{u} + \boldsymbol{d} + \boldsymbol{\xi}, \\ \dot{\boldsymbol{\eta}} &= \boldsymbol{J}\boldsymbol{\nu}, \end{aligned} \tag{3.2}$$

where $\boldsymbol{\eta} \in \mathbb{R}^6$ is the robot pose in a earth-fixed reference frame (global frame) $\Sigma_G : O_G - X_G Y_G Z_G$, $\dot{\boldsymbol{\eta}} \in \mathbb{R}^6$ is the derivative of the robot pose in the global frame, $\boldsymbol{\nu} \in \mathbb{R}^6$ is the velocity of the body-fixed frame $\Sigma_B : O_B - X_B Y_B Z_B$ with respect to the global frame expressed in the body-fixed frame (from now on: body-fixed velocity), $\dot{\boldsymbol{\nu}} \in \mathbb{R}^6$ is the derivative of the body-fixed velocity, i.e. body-fixed acceleration, $\boldsymbol{x} = \left[\boldsymbol{\eta}^T \ \boldsymbol{\nu}^T\right]^T \in \mathcal{X} \in \mathbb{R}^{12}$ is the system state, $\mathcal{X}$ represents the state space, $\dot{\boldsymbol{x}} = \left[\dot{\boldsymbol{\eta}}^T \ \dot{\boldsymbol{\nu}}^T\right]^T \in \mathbb{R}^{12}$ is the derivative of the system state, $\boldsymbol{J} \in \mathbb{R}^{6\times 6}$ is the system's Jacobian matrix, $\boldsymbol{M} \in \mathbb{R}^{6\times 6}$ is the inertia matrix, $\boldsymbol{C}(\boldsymbol{\nu}) \in \mathbb{R}^{6\times 6}$ is the matrix of Coriolis and centripetal terms, $\boldsymbol{M} = \boldsymbol{M}_{RB} + \boldsymbol{M}_A$ and $\boldsymbol{C} = \boldsymbol{C}_{RB} + \boldsymbol{C}_A$ include also added mass terms, $\boldsymbol{D}_{RB}(\boldsymbol{\nu}) \in \mathbb{R}^{6\times 6}$ is the matrix of drag forces, $\boldsymbol{g}_{RB}(\boldsymbol{\eta}) \in \mathbb{R}^6$ is the vector of gravity and buoyancy forces, $\boldsymbol{u} \in \mathcal{U} \in \mathbb{R}^6$ is the control vector applied to the system, $\mathcal{U}$ represents the action space, $\boldsymbol{d} \in \mathbb{R}^6$ represents unknown wave forces, and $\boldsymbol{\xi} \in \mathbb{R}^6$ represents unknown model uncertainties. The wave forces $\boldsymbol{d}$ are considered as external disturbances to the system, and can also be regarded as environmental uncertainties. They are assumed to be time-correlated signals following specific waveforms, and may frequently exceed the control constraints of the robot $\overline{\boldsymbol{u}}, \underline{\boldsymbol{u}} \in \mathbb{R}^6$. The model uncertainties $\boldsymbol{\xi}$ include both parametric and structural uncertainties in this work. The presence of model uncertainties causes the behaviour mismatch between policies trained on the mathematical model and the real system. The sources of uncertainties may include complex hydrodynamics effect, system control latency, and so on. Also notice that the parameters of the dynamics model are fixed but unknown to the learning algorithms.

3.1.2 Control Objective

The goal of controller is to navigate an underwater robot from a start position with an initial velocity to a target position then stabilize itself around the target, under external disturbances that can be described as time-correlated signals and may frequently exceed the control constraints of robot's thrusters.

In this work, we consider discretization of the continuous-time system in Equation 3.1, modeled by a generic function:

$$\boldsymbol{x}_{t+1} = f\left(\boldsymbol{x}_t, \boldsymbol{u}_t\right), \tag{3.3}$$

which describes the evolution from time t to $t+1$ of the state $\boldsymbol{x} \in \mathcal{X} \in \mathbb{R}^{n_x}$, given the action $\boldsymbol{u} \in \mathcal{U} \in \mathbb{R}^{n_u}$, where $\mathcal{X}$ and $\mathcal{U}$ represent state space and action space respectively. In an open-loop optimal control problem, a model of the dynamics is used to make predictions, which are then used for action selection. A trajectory $(\boldsymbol{X}, \boldsymbol{U})$ is a sequence of controls $\boldsymbol{U} = (\boldsymbol{u}_0, \boldsymbol{u}_1, \cdots, \boldsymbol{u}_{T-1})$, and corresponding state sequence $\boldsymbol{X} = (\boldsymbol{x}_0, \boldsymbol{x}_1, \cdots, \boldsymbol{x}_{T-1}, \boldsymbol{x}_T)$ satisfying Equation 3.3.

The objective function denoted by J is the sum of costs l, incurred when the system starts from initial state $\boldsymbol{x}_0$ and is controlled by the control sequence $\boldsymbol{U}$ until the horizon T is reached:

$$J\left(\boldsymbol{x}_0, \boldsymbol{U}\right) = \sum_{t=0}^{T-1} l\left(\boldsymbol{x}_t, \boldsymbol{u}_t\right), \tag{3.4}$$

the cost function is given as:

$$l\left(\boldsymbol{x}_t, \boldsymbol{u}_t\right) = \boldsymbol{x}_t^T \boldsymbol{Q} \boldsymbol{x}_t + \boldsymbol{u}_t^T \boldsymbol{R} \boldsymbol{u}_t, \tag{3.5}$$

where $\boldsymbol{Q} \in \mathbb{R}^{n_x \times n_x}$ and $\boldsymbol{R} \in \mathbb{R}^{n_u \times n_u}$ are weight matrices. The cost on state is used to minimize the Euler distance between the current position and target position, and minimize the current velocity of the robot, in order to stabilize the robot at the target position; the cost on control is used to minimize the control output, in order to perform the task with minimal power consumption. The trajectory is represented implicitly using only the controls $\boldsymbol{U}$. The state sequence $\boldsymbol{X}$ is recovered by integration of Equation 3.3 from the initial state $\boldsymbol{x}_0$. The solution of the optimal control problem is the control sequence corresponding to the minimized the objective function J:

$$\boldsymbol{U}^*\left(\boldsymbol{x}_0\right) = \underset{\boldsymbol{U}}{\operatorname{argmin}} J\left(\boldsymbol{x}_0, \boldsymbol{U}\right). \tag{3.6}$$

Note the controller aims to find $\boldsymbol{U}^*\left(\boldsymbol{x}_0\right)$ for a particular $\boldsymbol{x}_0$, rather than for all possible initial states. In addition, the control constraints need to be considered when optimizing the control sequence (Tassa et al., 2014). We consider the control constraints of the form:

$$\underline{\boldsymbol{u}} \leq \boldsymbol{u} \leq \overline{\boldsymbol{u}}, \tag{3.7}$$

with element-wise inequality and $\underline{\boldsymbol{u}}$, $\overline{\boldsymbol{u}}$ represent the respective lower and upper bounds.

3.1.3 Reinforcement Learning

RL problem is typically framed as a MDP, where at each timestep t, the system makes a transition from the state variable $\boldsymbol{x}_t$ to $\boldsymbol{x}_{t+1}$ in response to a control signal $\boldsymbol{u}_t$ chose from some policy π under a dynamics model f, meanwhile collecting a scalar reward r_t according to a reward function r. RL generally considers stochastic systems of the form (Sæmundsson et al., 2018):

$$\boldsymbol{x}_{t+1} = f\left(\boldsymbol{x}_t, \boldsymbol{u}_t\right) + \boldsymbol{\epsilon}, \tag{3.8}$$

with independent and identically distributed (i.i.d.) system noise marginalized over time $\boldsymbol{\epsilon} \sim \mathcal{N}\left(\mathbf{0}, \boldsymbol{E}\right)$, where $\boldsymbol{E} = \operatorname{diag}\left\{\sigma_1^2, \cdots, \sigma_{n_x}^2\right\}$. While the wave forces should be regarded as functions of time instead of random noises, due to their large strengths and time-correlated characteristics. The reward function is chosen to be the negative cost:

$$r\left(\boldsymbol{x}_t, \boldsymbol{u}_t\right) = -l\left(\boldsymbol{x}_t, \boldsymbol{u}_t\right). \tag{3.9}$$

The goal of RL is to learn a policy $\boldsymbol{u}_t = \pi\left(\boldsymbol{x}_t\right) + \boldsymbol{n}, \boldsymbol{n} \sim \mathcal{N}$ that maximizes the value function, i.e. the expected value of discounted sum of future rewards:

$$V\left(\boldsymbol{x}_t\right) = \mathbb{E}\left[\sum_{k=t}^{T} \gamma^{k-t} r\left(\boldsymbol{x}_k, \boldsymbol{u}_k\right)\right], \tag{3.10}$$

where $\gamma \in [0, 1]$ is a discount factor that prioritizes near-term rewards and T is the horizon of each episode. Usually, RL requires a number of episodes to correctly estimate the value function Equation 3.10. These training samples are obtained by adding exploration noise $\boldsymbol{n}$ to the control signal $\boldsymbol{u}_t$ at each timestep. The outcome of this exploratory policy is not known in advance and therefore may be damaging to the system or its environment.

3.2 History Window Reinforcement Learning

HWRL is proposed to be combat the excessive wave forces, this algorithm characterizes the disturbed system dynamics model as a multi-order Markov chain, and assumes the unobservable time-correlated disturbances and their predictions over next planning horizon are encoded in the most recent state-action history of the robot $\boldsymbol{H}_t = (\boldsymbol{x}_{t-T_H}, \boldsymbol{u}_{t-T_H}, \cdots, \boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1})$, where T_H represents the length of the history. The policy is then trained to produce control signals based on a fixed length of state-action history along with the current system state, and the embedding of disturbance behaviours can be learned within policy optimization. Note that control actions $\boldsymbol{u}$ are also included in the history queue $\boldsymbol{H}$ to encode the disturbance behaviours in contrast to state-only history, which, for example, has been used for estimating velocities in training video game player (Mnih et al., 2015).

Before using this state-action history to train a control policy, it is required to verify the rationality of multi-order Markov chain hypothesis, through validating the existence of a transition function $\boldsymbol{x}_{t+1} = f_{\boldsymbol{H}}\left(\boldsymbol{H}_t, \boldsymbol{x}_t, \boldsymbol{u}_t\right)$.

3.2.1 Validation of Multi-Order Markov Chain

In this part of work, we implemented a simpler setting to validate our assumption of multi-order Markov chain using inverted pendulum swing-up problem ($\boldsymbol{X} \in \mathbb{R}^3, \boldsymbol{U} \in \mathbb{R}^1$), as shown in Figure 3.3. The inverted pendulum swing-up problem is a classic problem in the control literature. In this problem, the inverted pendulum starts with a random angle, and the goal is to swing it up and keep it upright. The disturbances act as an additional torque along with the control torque.

The transition function $\hat{f}_{\boldsymbol{H}}(\boldsymbol{h}_t, \boldsymbol{x}_t, \boldsymbol{u}_t; \theta_f)$ is represented by a deep neural network, where the parameters θ_f represent the network weights. An intuitive parameterization for $\hat{f}_{\boldsymbol{H}}(\boldsymbol{h}_t, \boldsymbol{x}_t, \boldsymbol{u}_t; \theta_f)$ proceeds to the next state $\boldsymbol{x}_{t+1}$, given the most recent history $\boldsymbol{H}_t$, the current state $\boldsymbol{x}_t$ and action $\boldsymbol{u}_t$ as input. However, when there is a very short time interval Δt between two consecutive states, the next state $\boldsymbol{x}_{t+1}$ will become quite similar with the current state $\boldsymbol{x}_t$. Thus the transition function will be difficult to learn since the state difference may not well indicate the underlying dynamics (Nagabandi et al., 2018b). This problem can be solved by learning a transition function that predicts a state change during one timestep Δt. Then the next state is now given by: $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \hat{f}_{\boldsymbol{H}}(\boldsymbol{H}_t, \boldsymbol{x}_t, \boldsymbol{u}_t; \theta_f)$.

The training data is collected by sampling starting configuration $\boldsymbol{x}_0 \sim p(\boldsymbol{x}_0)$, generating random disturbance parameters, performing random actions at each timestep, and recording the resultant trajectories $\boldsymbol{\tau} = (\boldsymbol{x}_0, \boldsymbol{u}_0, \cdots, \boldsymbol{x}_{T-1}, \boldsymbol{u}_{T-1}, \boldsymbol{x}_T)$ of length T.

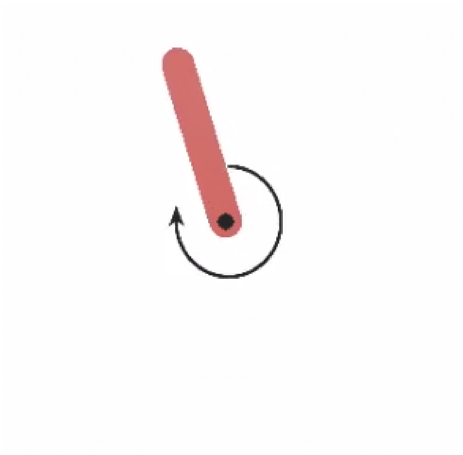


FIGURE 3.3: Simulated inverted pendulum.

The trajectories $\{\boldsymbol{\tau}_i\}$ are then decomposed into training inputs $(\boldsymbol{H}_t, \boldsymbol{x}_t, \boldsymbol{u}_t)$ and corresponding labels $\boldsymbol{x}_{t+1} - \boldsymbol{x}_t$. The useful training data should begin at $t = T_H$ in each episode, since the agent starts to observe the full length of history at this time. The training data is then normalized and stored in the dataset $\mathcal{D}_{train}$.

The transition model $\hat{f}_{\boldsymbol{H}}(\boldsymbol{h}_t, \boldsymbol{x}_t, \boldsymbol{u}_t; \theta_f)$ is trained by minimizing the loss:

$$L(\theta_f) = \frac{1}{|\mathcal{D}_{train}|} \sum_{(\boldsymbol{H}_t, \boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{x}_{t+1}) \in \mathcal{D}_{train}} \left\| (\boldsymbol{x}_{t+1} - \boldsymbol{x}_t) - \hat{f}_{\boldsymbol{H}}(\boldsymbol{H}_t, \boldsymbol{x}_t, \boldsymbol{u}_t; \theta_f) \right\|, \tag{3.11}$$

using Stochastic Gradient Descent (SGD), where $\|\cdot\|$ represents L2-norm. While training on the dataset $\mathcal{D}_{train}$, we also evaluate the loss in Equation 3.11 on a evaluation dataset $\mathcal{D}_{val}$, including different trajectory data from the training dataset.

Through several experiments using the simulated inverted pendulum, we found that the error between the learned model and the actual transition is always less than 2%, which proves the existence of the transition model and thus the rationality of the multi-order Markov chain hypothesis to some extent.

3.2.2 Learning History Window Policy

The rationality of multi-order Markov chain hypothesis ensures that HWRL is able to learn a satisfactory policy $\pi(\boldsymbol{u}|\boldsymbol{h}, \boldsymbol{x}; \theta_\pi)$. In our implementations, DDPG (Lillicrap et al., 2015) is used to train the neural network policy. DDPG is a model-free, actor-critic algorithm based on deterministic policy gradient that deals with continuous action spaces. As shown in Figure 3.4, HWRL algorithm consists of an actor network and a critic network. The actor network acts as a control policy, which takes in the fixed length of state-action history as well as the current state to generate action, the critic network is used to evaluate action-value function (discounted sum of future rewards) based on the state-action history, the current state and the selected action. The action-value function and Temporal Difference (TD) error are used respectively to update the parameters of the actor network θ_π and the critic network θ_Q.

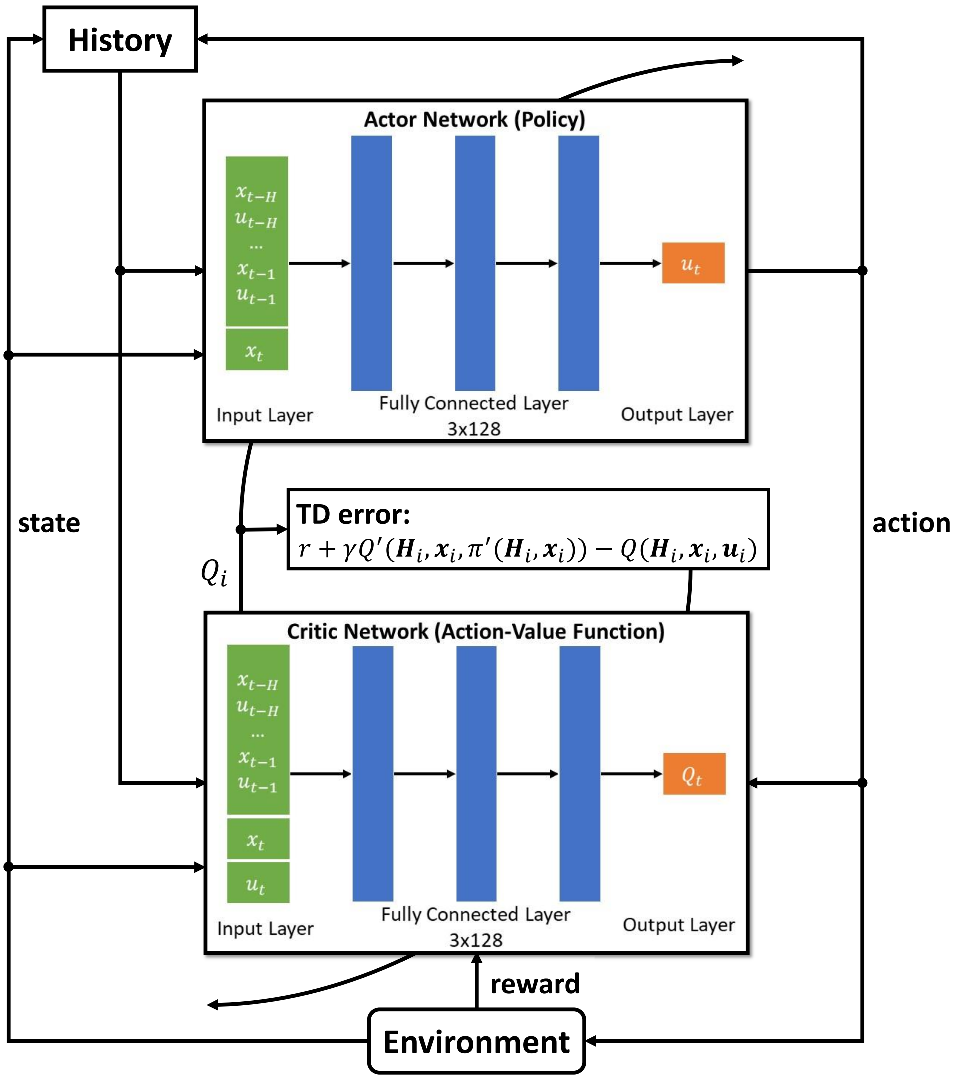

History
Actor Network (Policy)
x_{t-H}
u_{t-H}
...
x_{t-1}
u_{t-1}
x_t
u_t
Input Layer
Fully Connected Layer 3x128
Output Layer
state
action
TD error:
$r + \gamma Q'(\boldsymbol{H}_i, \boldsymbol{x}_i, \pi'(\boldsymbol{H}_i, \boldsymbol{x}_i)) - Q(\boldsymbol{H}_i, \boldsymbol{x}_i, \boldsymbol{u}_i)$
Q_i
Critic Network (Action-Value Function)
x_{t-H}
u_{t-H}
...
x_{t-1}
u_{t-1}
x_t
u_t
Q_t
Input Layer
Fully Connected Layer 3x128
Output Layer
reward
Environment

FIGURE 3.4: Network architecture of HWRL.

The algorithm details are shown in Algorithm 1. During training, our purpose is to enable the trained policy to deal with unknown time-correlated disturbances, thus we randomly generate parameters of disturbances (amplitudes, frequencies and phases) in each episode. Furthermore, in each episode, when the number of timesteps does not reach the history length T_H, the algorithm will randomly choose action, and add current state and action into the history queue $\boldsymbol{H}$. When the number of timesteps exceeds the history length T_H, the algorithm will choose action based on the current deterministic policy π, then update the history queue (delete the oldest state-action pair and add the latest one). The transition $(\boldsymbol{H}_t, \boldsymbol{x}_t, \boldsymbol{u}_t, r_t, \boldsymbol{H}_{t+1}, \boldsymbol{x}_{t+1})$ for each timestep is saved to a replay memory R. The training begins when the replay memory is full, a batch of N transitions is grabbed

Algorithm 1: Learning HWRL

```
Randomly initialize critic network $Q\left(\boldsymbol{H}, \boldsymbol{x}, \boldsymbol{u}; \theta_Q\right)$ and actor network $\pi\left(\boldsymbol{H}, \boldsymbol{x}; \theta_\pi\right)$
  with weights $\theta_Q$ and $\theta_\pi$
Initialize target network $Q'$ and $\pi'$ with weights $\theta_{Q'} \leftarrow \theta_Q$, $\theta_{\pi'} \leftarrow \theta_\pi$
Initialize replay memory $R$
for $episode = 1 : M$ do
    Reset initial state $\boldsymbol{x}_0 \sim p\left(\boldsymbol{x}_0\right)$
    Initialize a random process $\mathcal{N}$ for action exploration
    for $t = 0 : T - 1$ do
        if $t \leq T_H$ then
            Select action randomly within control constraints: $\boldsymbol{u}_t \in [\underline{\boldsymbol{u}}, \overline{\boldsymbol{u}}]$
            Execute action $\boldsymbol{u}_t$, then observe reward $r_t$ and next state $\boldsymbol{x}_{t+1}$
            Add $\boldsymbol{x}_t$ and $\boldsymbol{u}_t$ into history queue $\boldsymbol{H}_{t+1}$
        end
        else if $t > T_H$ then
            Select action according to current policy and exploration noise:
              $\boldsymbol{u}_t \sim \pi\left(\boldsymbol{H}_t, \boldsymbol{x}_t; \theta_\pi\right) + \mathcal{N}_t$
            Execute action $\boldsymbol{u}_t$, then observe reward $r_t$ and next steta $\boldsymbol{x}_{t+1}$
            Update history queue $\boldsymbol{H}_t$ to $\boldsymbol{H}_{t+1}$ by deleting $\boldsymbol{x}_{t-T_H}$ and $\boldsymbol{u}_{t-T_H}$ and
              adding $\boldsymbol{x}_t$ and $\boldsymbol{u}_t$
            Store transition $(\boldsymbol{H}_t; \boldsymbol{x}_t; \boldsymbol{u}_t; r_t; \boldsymbol{H}_{t+1}; \boldsymbol{x}_{t+1})$ into $R$
            if $R$ is full then
                Sample a random minibatch of $N$ transitions $(\boldsymbol{H}_i; \boldsymbol{x}_i; \boldsymbol{u}_i; r_i; \boldsymbol{H}_{i+1}; \boldsymbol{x}_{i+1})$
                  from $R$
                Set target action-value function:
                  $y_i = r_i + \gamma Q'\left(\boldsymbol{H}_{i+1}, \boldsymbol{x}_{i+1}, \pi'\left(\boldsymbol{H}_{i+1}, \boldsymbol{x}_{i+1}; \theta_{\pi'}\right); \theta_{Q'}\right)$
                Update critic network by minimizing the loss function:
                  $L_c = \frac{1}{N}\sum_i \left(y_i - Q\left(\boldsymbol{H}_i, \boldsymbol{x}_i, \boldsymbol{u}_i; \theta_Q\right)\right)^2$
                Update actor network by minimizing the loss function:
                  $L_a = -\frac{1}{N}\sum_i Q\left(\boldsymbol{H}_i, \boldsymbol{x}_i, \pi\left(\boldsymbol{H}_i, \boldsymbol{x}_i; \theta_\pi\right); \theta_Q\right)$
                Update the target network:
                  $\theta_{Q'} \leftarrow \beta\theta_Q + (1-\beta)\theta_{Q'}, \quad \theta_{\pi'} \leftarrow \beta\theta_\pi + (1-\beta)\theta_{\pi'}, \quad \beta \ll 1$
            end
        end
        Update state: $\boldsymbol{x}_t \leftarrow \boldsymbol{x}_{t+1}$
        Update history queue: $\boldsymbol{H}_t \leftarrow \boldsymbol{H}_{t+1}$
    end
end
```

from the replay memory and used to train the actor and critic network at each timestep through minimizing actor loss function L_a and critic loss function L_c. We also need to note that the disturbance behaviour is encoded in the state-action history, thus during the training of the policy, the embedding of disturbance behaviour are also learned.

3.3 Disturbance Observer Network

HWRL attempts to resolve the environmental disturbances by making the selected action depend not only on the current state, but also on a fixed number of the most recent states and actions. However, it is difficult to determine an optimal length of the applied history data. Shorter history may not provide sufficient information about disturbances, longer history will make the training difficult. HWRL only considered the history length as a hyperparameter that was statistically optimized through trail-and-error during training.

Due to the difficulty in determining optimal history length through HWRL, RNN is then utilized to automatically learn how much past experience should be explored to achieve optimal performance. This section introduces another joint learning algorithm called DOB-Net. A conventional DOB is first compared with a GRU, the results show some similarities in the structure of processing hidden information. Thus, an enhanced observer network for excessive time-correlated disturbances is designed using GRU, encoding the disturbance waveforms into the hidden state of GRU. A controller network is then built upon this encoding in order to generate optimal controls.

3.3.1 Conventional Disturbance Observer

The main objective of the use of DOB (Chen et al., 2000, 1999) is to deduce the unknown or uncertain disturbances (or the influence of the disturbances) from measurable variables, without the use of additional sensors. Then, a control signal can be taken, based on the disturbance estimation, to compensate for the influence of the disturbances, which is called DOBC (Chen et al., 2016).

The basic idea of conventional DOB is to estimate disturbances based on robot state and executed control, its formulation is proposed as:

$$\begin{aligned}
&\boldsymbol{G}(\boldsymbol{\eta},\boldsymbol{\nu}) = \boldsymbol{C}(\boldsymbol{\nu})\,\boldsymbol{\nu} + \boldsymbol{D}_{RB}(\boldsymbol{\nu})\,\boldsymbol{\nu} + \boldsymbol{g}_{RB}(\boldsymbol{\eta}),\\
&\dot{\boldsymbol{y}} = -\boldsymbol{L}(\boldsymbol{\eta},\boldsymbol{\nu})\,\boldsymbol{y} + \boldsymbol{L}(\boldsymbol{\eta},\boldsymbol{\nu})\left(\boldsymbol{G}(\boldsymbol{\eta},\boldsymbol{\nu}) - \boldsymbol{p}(\boldsymbol{\eta},\boldsymbol{\nu}) - \boldsymbol{u}\right),\\
&\hat{\boldsymbol{d}} = \boldsymbol{y} + \boldsymbol{p}(\boldsymbol{\eta},\boldsymbol{\nu}),
\end{aligned} \tag{3.12}$$

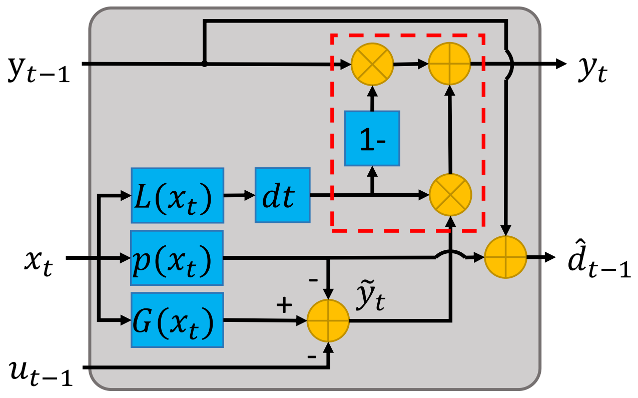


FIGURE 3.5: Architecture of DOB.

where $\hat{\boldsymbol{d}}$ is the estimated disturbances, $\boldsymbol{y}$ is the internal state of the nonlinear observer and $\boldsymbol{p}(\boldsymbol{\eta}, \boldsymbol{\nu})$ is the nonlinear function to be designed. The DOB gain $\boldsymbol{L}(\boldsymbol{\eta}, \boldsymbol{\nu})$ is determined by the following nonlinear function:

$$\boldsymbol{L}(\boldsymbol{\eta}, \boldsymbol{\nu}) \boldsymbol{M}\dot{\boldsymbol{\nu}} = \begin{bmatrix} \frac{\partial \boldsymbol{p}(\boldsymbol{\eta},\boldsymbol{\nu})}{\partial \boldsymbol{\eta}} & \frac{\partial \boldsymbol{p}(\boldsymbol{\eta},\boldsymbol{\nu})}{\partial \boldsymbol{\nu}} \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{\eta}} \\ \dot{\boldsymbol{\nu}} \end{bmatrix}. \tag{3.13}$$

It has been shown in (Chen et al., 2000) that DOB is globally asymptotically stable by choosing:

$$\boldsymbol{L}(\boldsymbol{\eta}, \boldsymbol{\nu}) = \operatorname{diag}\{c, \cdots, c\}, \tag{3.14}$$

where $c > 0$. More specifically, the exponential convergence rate can be specified by choosing c. The convergence and the performance of the DOB have been established for slowly time-varying disturbances and disturbances with bounded rate in (Li et al., 2016). A discrete version of DOB is also provided (illustrated in Figure 3.5):

$$\begin{aligned} \tilde{\boldsymbol{y}}_t &= \boldsymbol{G}(\boldsymbol{x}_t) - \boldsymbol{p}(\boldsymbol{x}_t) - \boldsymbol{u}_{t-1}, \\ \boldsymbol{y}_t &= (1 - \boldsymbol{L}(\boldsymbol{x}_t)\Delta t)\, \boldsymbol{y}_{t-1} + \boldsymbol{L}(\boldsymbol{x}_t)\Delta t \tilde{\boldsymbol{y}}_t, \\ \hat{\boldsymbol{d}}_{t-1} &= \boldsymbol{y}_{t-1} + \boldsymbol{p}(\boldsymbol{x}_t). \end{aligned} \tag{3.15}$$

3.3.2 Gated Recurrent Unit

The architecture of GRU (Cho et al., 2014) is shown in Figure 3.6. The formulations are given below:

$$\begin{aligned}
\boldsymbol{z}_t &= \sigma\left(\boldsymbol{W}_z\left[\boldsymbol{h}_{t-1}, \boldsymbol{s}_t\right] + \boldsymbol{b}_z\right), \\
\boldsymbol{r}_t &= \sigma\left(\boldsymbol{W}_r\left[\boldsymbol{h}_{t-1}, \boldsymbol{s}_t\right] + \boldsymbol{b}_r\right), \\
\tilde{\boldsymbol{h}}_t &= \tanh\left(\boldsymbol{W}_h\left[\boldsymbol{r}_t \circ \boldsymbol{h}_{t-1}, \boldsymbol{s}_t\right] + \boldsymbol{b}_h\right), \\
\boldsymbol{h}_t &= (1-\boldsymbol{z}_t) \circ \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \circ \tilde{\boldsymbol{h}}_t,
\end{aligned} \tag{3.16}$$

where $\boldsymbol{s}_t$ is the input vector, $\boldsymbol{h}_t$ is the output vector, $\boldsymbol{z}_t$ is the update gate vector, $\boldsymbol{r}_t$ is the reset gate vector, $\boldsymbol{W}$ and $\boldsymbol{b}$ are the weight matrices and bias vectors, σ and *tanh* are the activation functions (sigmoid function and hyperbolic tangent). The operator $\circ$ denotes the Hadamard product.

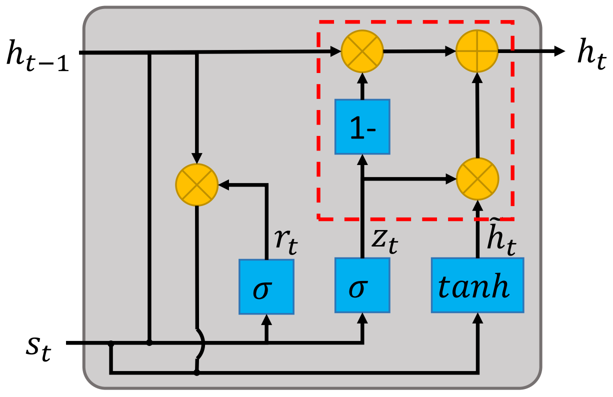


FIGURE 3.6: Architecture of GRU.

3.3.3 Learning Disturbance Observer Network

Network Architecture: DOB-Net is developed based on classical actor-critic architecture. As described in Figure 3.5 and Figure 3.6, the DOB and the GRU have a similar architecture, especially the part in red box. $\boldsymbol{y}_t$ of DOB acts as the hidden state, similar to the role of $\boldsymbol{h}_t$ in GRU, which preserves past processed information. DOB-Net is developed based on the similarity between the DOB and the GRU. Figure 3.7 shows the network architecture of DOB-Net in details. In order to imitate the function of DOB, a GRU is first employed to process the same input as DOB, which is the latest state-action

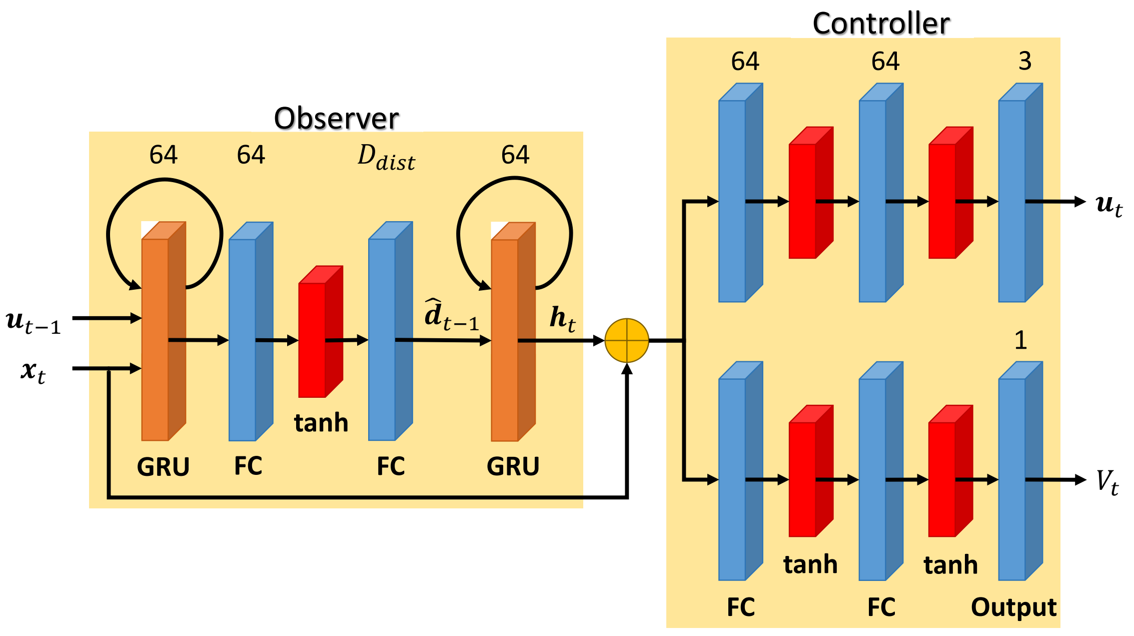


FIGURE 3.7: Network architecture of DOB-Net.

pair $(\boldsymbol{x}_t, \boldsymbol{u}_{t-1})$. Then, fully connected layers are required to further extract embedding of estimated disturbances $\hat{\boldsymbol{d}}_{t-1}$. After disturbance estimation, the observer network can be further enhanced through feeding this embedding $\hat{\boldsymbol{d}}_{t-1}$ into another GRU, in order to encode a sequence of past estimated disturbances. The hidden state of the second GRU $\boldsymbol{h}_t$ is supposed to represent the disturbance waveforms. It can then be combined with the current state $\boldsymbol{x}_t$, becoming the actual input of the following controller network. One design parameter of DOB-Net is the embedding dimension of $\hat{\boldsymbol{d}}_{t-1}$. In this section, 4-dimension (the dimension of disturbances) and 64-dimension (the dimension of GRU hidden state) are chosen and compared in simulation. Such comparison shows the flexibility of neural networks after building the observer from GRU.

Training: A2C (Mnih et al., 2016) is a lightweight and conceptually simple framework for RL that applies synchronous gradient descent to optimize neural network controllers. In this algorithm, multiple agents synchronously execute in parallel on multiple copies of the environment. This parallel mechanism decorrelates the agents' behaviours into a more stable process, since the parallel agents will be in various different states at any given timestep. In addition to the parallel learning mechanism, the algorithm also adopts a shared and slowly-changing target network when calculating loss function, and applies the

Algorithm 2: Learning DOB-Net - pseudocode for each thread

```
Assume global shared parameter vectors $\theta_\pi$ and $\theta_V$
Assume thread-specific parameter vectors $\theta'_\pi$ and $\theta'_V$
Initialize global shared counter $T = 0$
Initialize thread step counter $t = 1$
repeat
    Reset gradients: $d\theta_\pi \leftarrow 0$ and $d\theta_V \leftarrow 0$
    Synchronize thread-specific parameters $\theta'_\pi = \theta_\pi$ and $\theta'_V = \theta_V$
    $t_{start} = t$
    Get state $\boldsymbol{x}_t$, last action $\boldsymbol{u}_{t-1}$ and last hidden state $\boldsymbol{h}_{t-1}$
    repeat
        Sample action $\boldsymbol{u}_t$ according to policy $\pi\left(\boldsymbol{u}_t|\boldsymbol{x}_t, \boldsymbol{u}_{t-1}, \boldsymbol{h}_{t-1}; \theta'_\pi\right)$
        Receive new hidden state $\boldsymbol{h}_t$
        Perform $\boldsymbol{u}_t$, receive reward $r_t$ and new state $\boldsymbol{x}_{t+1}$
        $t \leftarrow t + 1$
        $T \leftarrow T + 1$
    until terminal $\boldsymbol{x}_t$ or $t - t_{start} == t_{max}$;
    $R = \begin{cases} 0 & \text{for terminal } \boldsymbol{x}_t \\ V\left(\boldsymbol{x}_t, \boldsymbol{u}_{t-1}, \boldsymbol{h}_{t-1}; \theta'_v\right) & \text{for non-terminal } \boldsymbol{x}_t \end{cases}$
    for $i \in \{t-1, \cdots, t_{start}\}$ do
        $R \leftarrow r_i + \gamma R$
        Accumulate gradients w.r.t. $\theta'_\pi$:
          $d\theta_\pi \leftarrow d\theta_\pi + \nabla_{\theta'_\pi} \log \pi\left(\boldsymbol{u}_i|\boldsymbol{x}_i, \boldsymbol{u}_{i-1}, \boldsymbol{h}_{i-1}; \theta'_\pi\right)\left(R - V\left(\boldsymbol{x}_i, \boldsymbol{u}_{i-1}, \boldsymbol{h}_{i-1}; \theta'_V\right)\right)$
        Accumulate gradients w.r.t. $\theta'_V$:
          $d\theta_V \leftarrow d\theta_V + \partial\left(R - V\left(\boldsymbol{x}_i, \boldsymbol{u}_{i-1}, \boldsymbol{h}_{i-1}; \theta'_V\right)\right)^2 / \partial\theta'_V$
    end
    Perform synchronous update of $\theta_\pi$ using $d\theta_\pi$ and of $\theta_V$ using $d\theta_V$
until $T > T_{\max}$;
```

gradients after accumulating then over multiple timesteps to stabilize the learning process.

The algorithm is developed in A2C style. Algorithm 2 shows the pseudocode of DOB-Net, where each thread learns the policy through interacting with its own instance of the environment. The disturbance waveforms are also different in each thread, where their parameters (amplitudes, frequencies and phases) are randomly sampled. We found this setting helps accelerate the convergence of learning and improve performance, through comparison with using the same disturbance waveform through all threads during numerical simulations. The update performed by the algorithm can be seen as:

$$\nabla_{\theta'_\pi} \log \pi\left(\boldsymbol{u}_t|\boldsymbol{x}_t, \boldsymbol{u}_{t-1}, \boldsymbol{h}_{t-1}; \theta'_\pi\right) A\left(\boldsymbol{x}_t, \boldsymbol{u}_{t-1}, \boldsymbol{h}_{t-1}, \boldsymbol{u}_t; \theta_\pi, \theta_V\right), \tag{3.17}$$

where $A\left(\boldsymbol{x}_t, \boldsymbol{u}_{t-1}, \boldsymbol{h}_{t-1}, \boldsymbol{u}_t; \theta_\pi, \theta_V\right)$ is an estimation of the advantage function given as:

$$\sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V\left(\boldsymbol{x}_{t+k}, \boldsymbol{u}_{t+k-1}, \boldsymbol{h}_{t+k-1}; \theta_V\right) - V\left(\boldsymbol{x}_t, \boldsymbol{u}_{t-1}, \boldsymbol{h}_{t-1}; \theta_V\right), \tag{3.18}$$

where k can vary from different state and is upper-bounded by t_{max}.

3.4 Simulation

3.4.1 Simulation Setup

In the simulations, a pose regulation task is simulated to test the control performance of the proposed algorithms under excessive disturbances. Each episode contains 200 timesteps with $0.05s$ per timestep. In each episode, the robot starts at a random pose with a random velocity, and the goal is to navigate to a target pose, which is set to be zero in simulated coordinates, and stabilize around the target. The simulated underwater robot has the mass of $60kg$ with the geometrical size of $0.68 \times 0.75 \times 0.19m^3$, as shown in Figure 3.8. The state space contains the robot's pose and velocity; the action space consists of the control forces and torques applied to the system. We only consider the motion and control in the directions of X, Y, Z position and yaw angle, thus the robot has an 8-dimensional

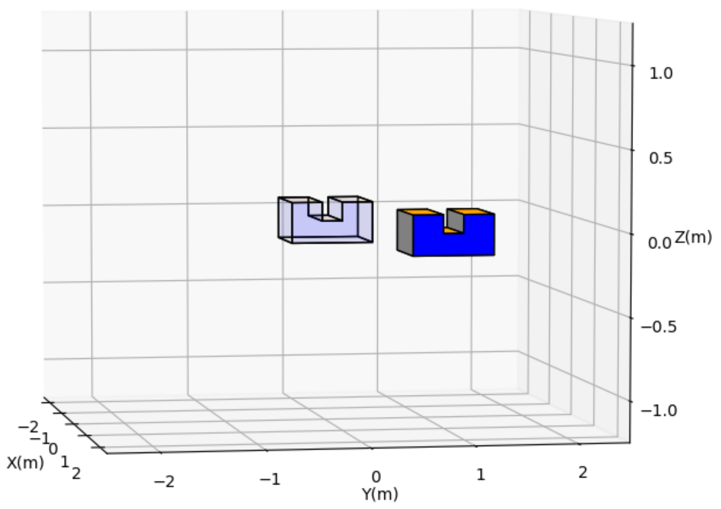


FIGURE 3.8: Simulated underwater robot (solid: current pose, transparent: target pose).

state space $\mathcal{X} \in \mathcal{R}^8$ and a 4-dimensional action space $\mathcal{U} \in \mathcal{R}^4$. The control constraints are given as $|\overline{\boldsymbol{u}}| = |\underline{\boldsymbol{u}}| = [112N\ 112N\ 94N\ 82Nm]^T$.

In these experiments, the simulated disturbances are exerted on all four directions (X, Y, Z and yaw) in inertial frame. They are constructed as a superposition of multiple sinusoidal waves (three in this work) with different amplitudes A, frequencies ω and phases ϕ. The distributions of these disturbance parameters are given in Table 3.1. Three different types of disturbances are applied, the first one (Type I) has small amplitudes that are within the system control constraints (around 50-80% of the control constraints); the second one (Type II) has moderate amplitudes that are close to or slightly exceed the control constraints (around 100-130% of the control constraints); the last one (Type III) has large amplitudes that exceed the control constraints a lot (around 130-160% of the control constraints). An example of Type III simulated disturbances is shown in Figure 3.9. Our goal is to enable the trained control policy to deal with unknown time-correlated wave disturbances, and the policy is expected to adapt to a wide range of waveforms, instead of a fixed one. Thus, the amplitudes, frequencies and phases of the disturbances are randomly sampled from the given distributions in each training episode.

In order to further validate the efficacy of the proposed algorithms, we also collected real-world wave data in open water. The data collection process is implemented through connecting a force/torque sensor between an underwater robot and a metal pole, the metal pole is fixed to a bridge over turbulent water flows, and the robot is deployed in the water and remains unactuated. Then, the readings of the force/torque sensor are regarded as the external wave forces and torques. The wave data has been collected through the

Table 3.1: Distributions of disturbance parameters used in simulation.

Component Wave		1	2	3
Type I	Amplitude w.r.t. $\lvert\overline{\boldsymbol{u}}\rvert$	$50 \sim 80\%$	$20 \sim 40\%$	$10 \sim 20\%$
	Period (s)	$2 \sim 4$	$1 \sim 2$	$0.5 \sim 1$
	Phase (rad)	$-\pi \sim \pi$	$-\pi \sim \pi$	$-\pi \sim \pi$
Type II	Amplitude w.r.t. $\lvert\overline{\boldsymbol{u}}\rvert$	$100 \sim 130\%$	$20 \sim 40\%$	$10 \sim 20\%$
	Period (s)	$2 \sim 4$	$1 \sim 2$	$0.5 \sim 1$
	Phase (rad)	$-\pi \sim \pi$	$-\pi \sim \pi$	$-\pi \sim \pi$
Type III	Amplitude w.r.t. $\lvert\overline{\boldsymbol{u}}\rvert$	$130 \sim 160\%$	$20 \sim 40\%$	$10 \sim 20\%$
	Period (s)	$2 \sim 4$	$1 \sim 2$	$0.5 \sim 1$
	Phase (rad)	$-\pi \sim \pi$	$-\pi \sim \pi$	$-\pi \sim \pi$

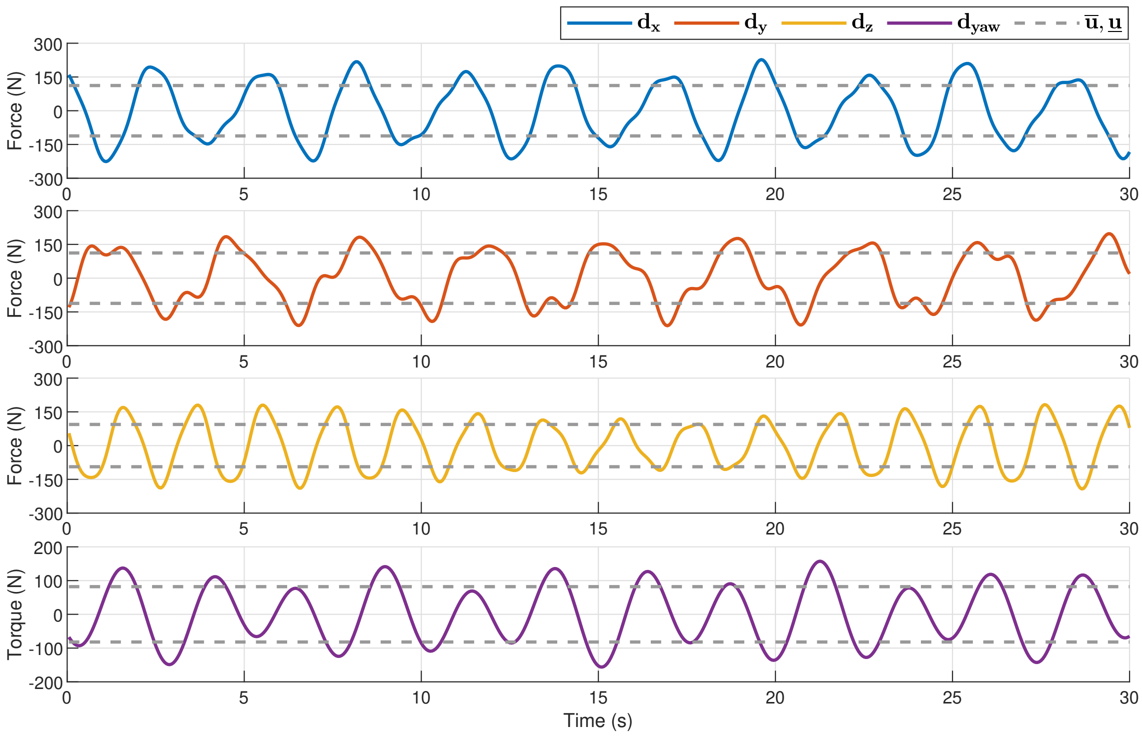


FIGURE 3.9: Example simulated wave disturbances (Type III) in X, Y, Z and yaw directions.

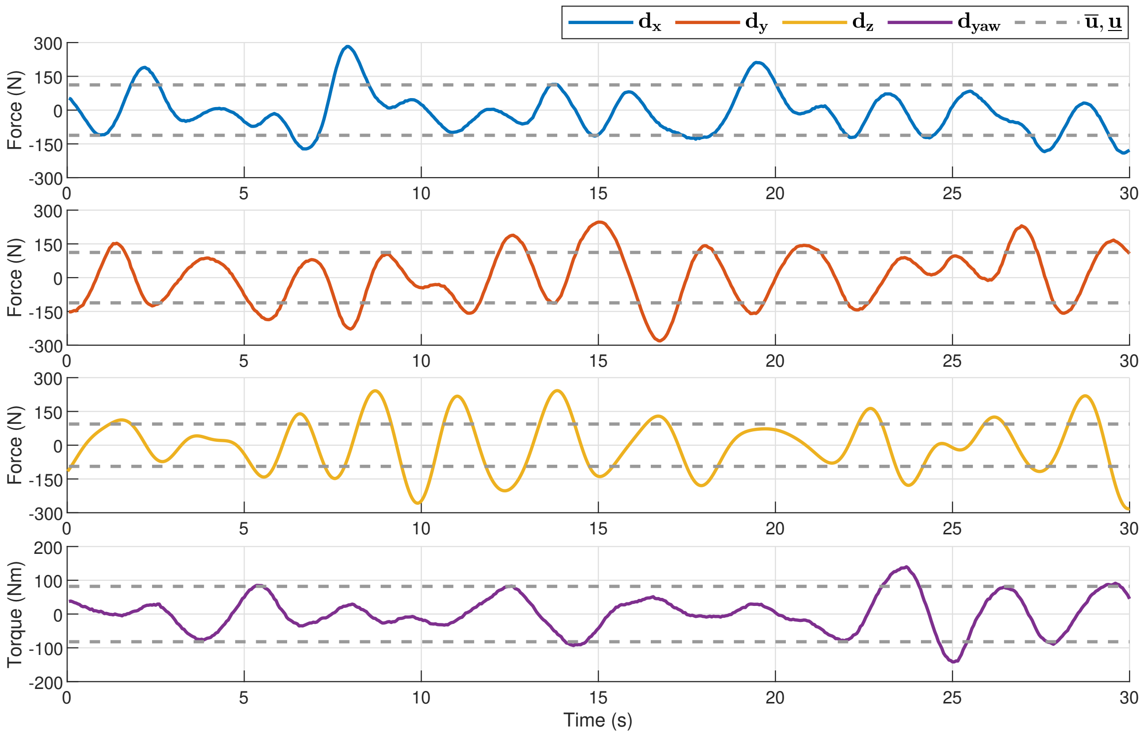


FIGURE 3.10: Example real wave disturbances in X, Y, Z and yaw directions.

force/torque sensor over a long time period (around two hours). The collected wave data is then sliced into a number of segments, and used for training or testing in episodic style. Figure 3.10 shows a small segment of the collected wave data. We found that the real-world disturbances have widely varying amplitudes, which are not constrained within the ranges seen during training, leading to a more challenging control scenario. Note that, in this part of the experiments, the RL algorithms are trained exclusively on the simulated disturbances, the real-world wave data is applied only for test purpose to verify the generalization of the RL controllers, no additional training or fine-tuning is conducted on these collected data.

In the remaining parts of this section, the training process and the control performance of both HWRL and DOB-Net are evaluated. Note that the resultant cumulative rewards during training are averaged over several training trials, which have randomly initialized parameters of neural networks and different random seeds. This setting holds for all results of training rewards throughout the thesis.

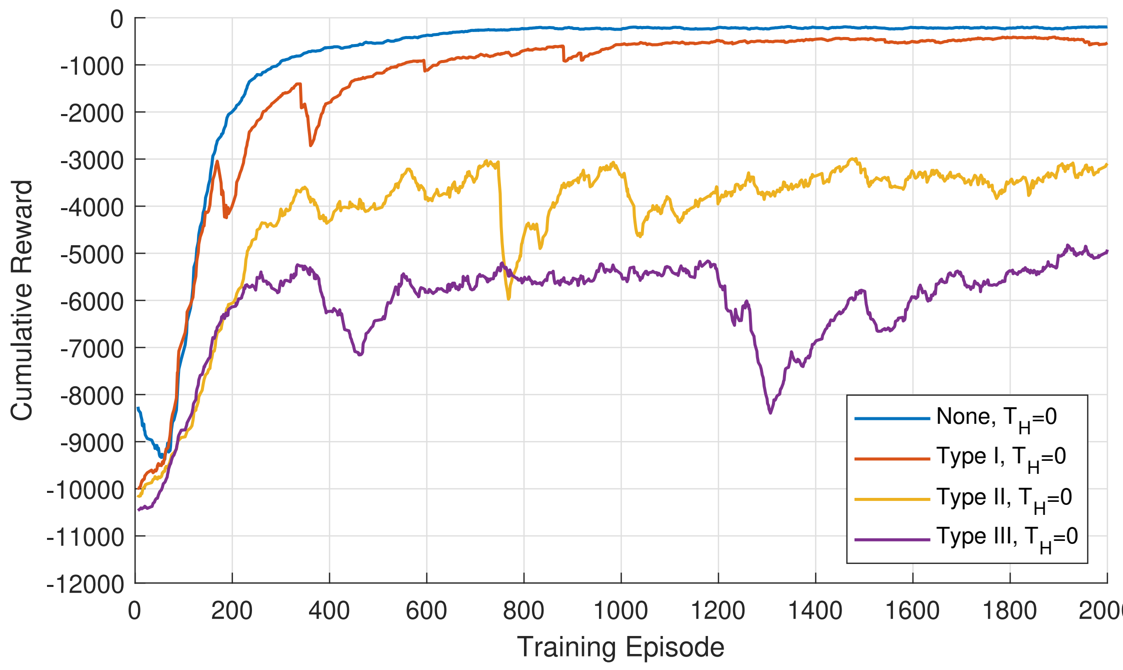


FIGURE 3.11: Training reward: comparison of different disturbance strengths for DDPG.

3.4.2 Results of History Window Reinforcement Learning

HWRL algorithm is applied to handle the unobservable disturbances through taking the state-action history $\boldsymbol{H}_t$ as additional input to the control policy along with the current state $\boldsymbol{x}_t$. Different strengths of the disturbances and different lengths of the history queue would affect the disturbance rejection performance. We first evaluate various disturbance strengths and history lengths for RL using empirical evaluations, then we compare the history window policy with a conventional feedback controller (RISE control) and a classical RL policy (DDPG) for their control performance subject to excessive disturbances.

Figure 3.11 presents the training process of a classical RL policy with four different disturbance scenarios (including the situation without disturbances), showing that stronger disturbances lead to slower convergence and lower final cumulative reward, the results accord with our preconception. The figure also shows that, the control performance is not much affected by the disturbances when they are within the robot control constraints (Type I disturbances). Once the disturbances start to exceed the control constraints, the performance then dramatically decreases. Such phenomenon illustrates that control actions

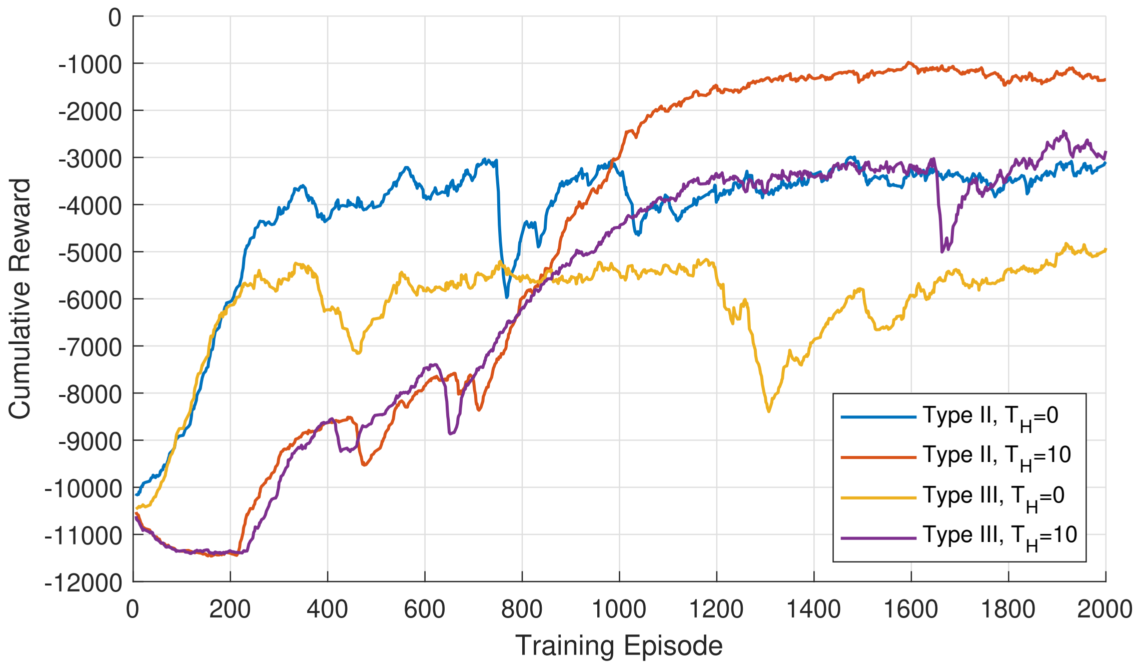


FIGURE 3.12: Training reward: comparison of using history or not for HWRL.

based on current observation is not able to well stabilize the system under disturbances exceeding the control constraints.

We then focus on the excessive disturbance scenarios (Type II and Type III disturbances). In order to mitigate the impact of the disturbances, we then enable the control policy to make better decision through using the history information. When taking 10-step history ($T_H = 10$) into consideration, as shown in Figure 3.12, both policies trained on Type II and Type III disturbances have improved control performance. This result shows that, the generated control actions based on past motion data does improve the disturbance rejection capability of the robot system. But the convergence speed apparently becomes lower, this might because the history information enlarges the state space, making the training process more difficult.

We then use Type III disturbances to test and compare the results using different lengths of the history queue. From Figure 3.13, we can see that using shorter history length leads to faster convergence, it means that the convergence speed is inversely related to the dimension of the state space. However, the relation between the history length and the cumulative reward is not monotonic. In this comparison, 10-step history gives the

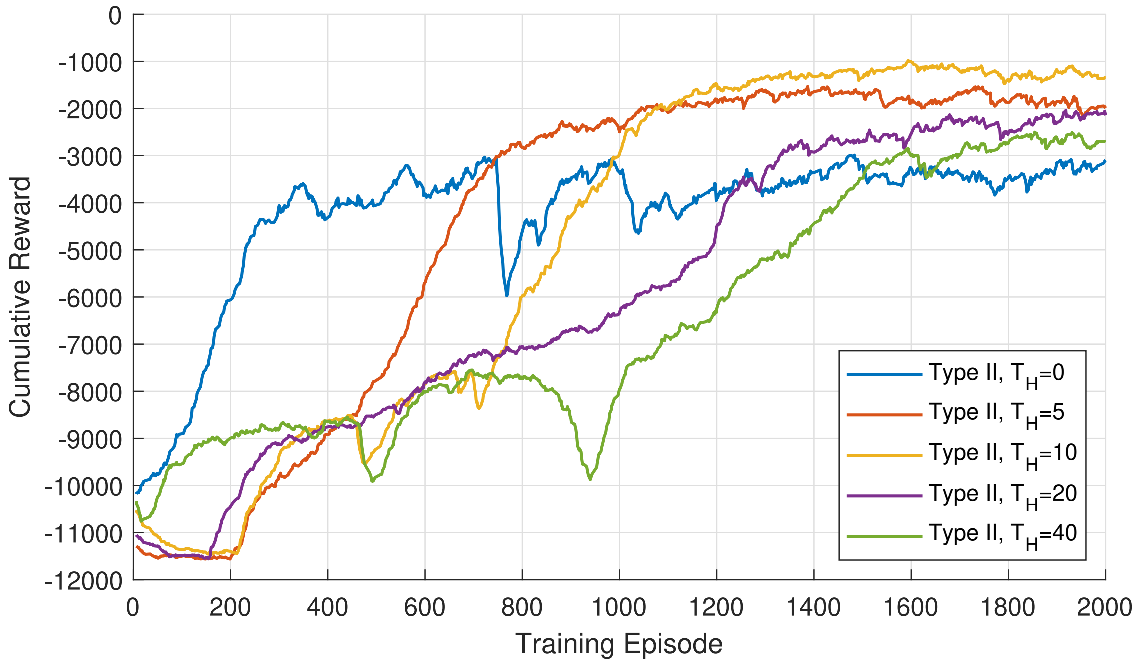


FIGURE 3.13: Training reward: comparison of different history lengths for HWRL.

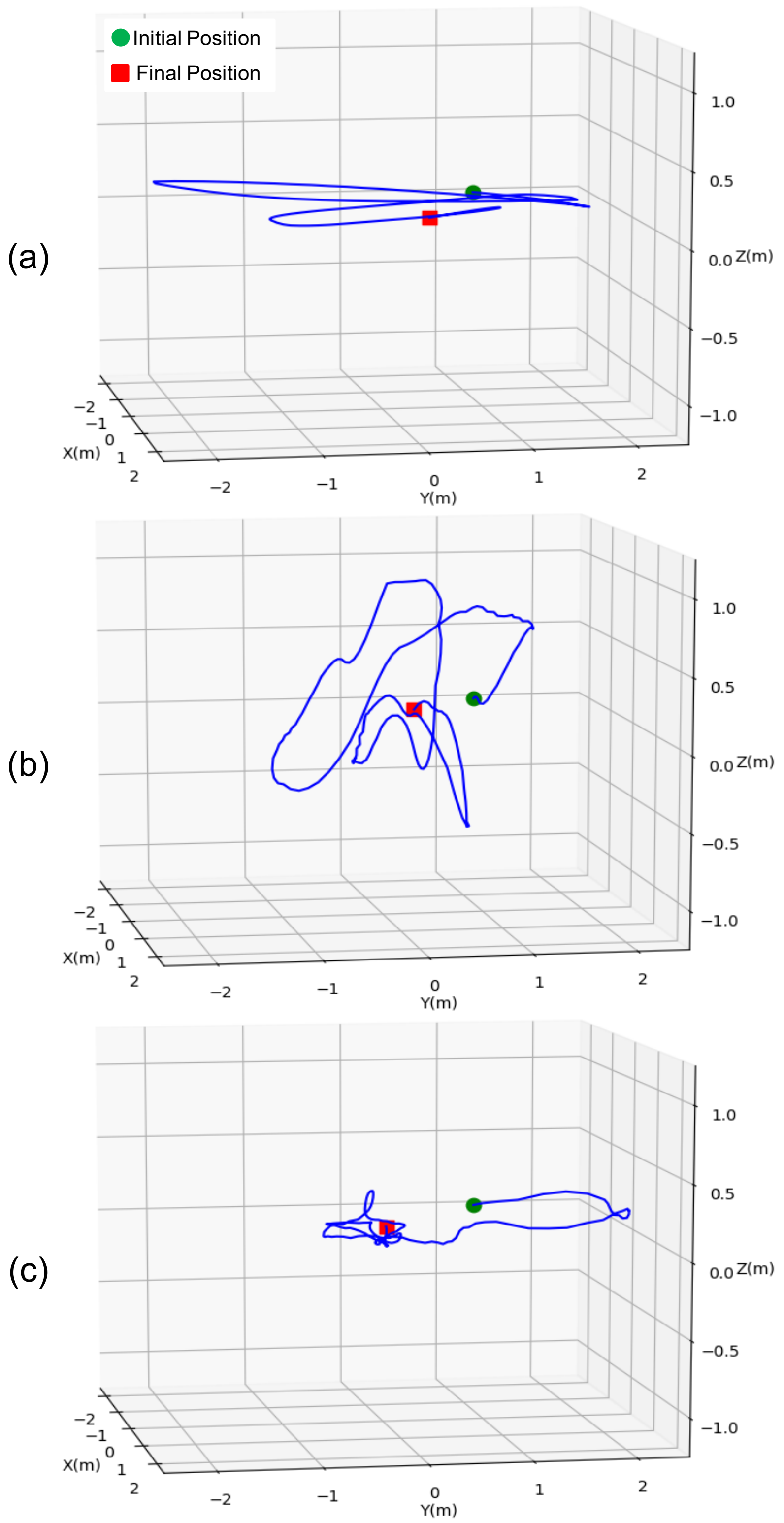


FIGURE 3.14: Comparison of 3D trajectories: (a) RISE controller; (b) DDPG policy; (c) HWRL policy.

best result, both being longer than or being shorter than this length makes the result worse. Thus, we believe there always exists an extremum for the control performance with respect to the history length. But we cannot ensure our result (10-step history) to be the optimal one, due to the sparse sampling of the history length in this comparison, and it will absolutely take a great effort to find an optimal value through trail-and-error. This part of knowledge still requires further investigation.

To further validate the effectiveness of HWRL, we test and compare the control performance of RISE controller (Fischer et al., 2014), conventional DDPG policy and HWRL policy, and plot their 3D trajectories in Figure 3.14 when performing the position regulation task. It can be seen that the robot is difficult to stabilize itself using either RISE controller or DDPG policy. While considering 10-step state-action history as additional input to the control policy (HWRL policy), the robot can quickly navigate to the target position and stabilize itself in a relatively small region thereafter.

3.4.3 Results of Disturbance Observer Network

We have already tested HWRL policy and showed its superior control capability for excessive time-correlated disturbances over conventional feedback controller and classical RL policy. Then, we look at DOB-Net and conduct more comprehensive experiments and comparison among the ten algorithms in Table 3.2.

TABLE 3.2: Different algorithms for disturbance rejection control.

Code Name	**Algorithm**
M1	RISE Control (Fischer et al., 2014)
M2	DOBC
M3	A2C
M4	History Window A2C with state history (HWA2C-x)
M5	History Window A2C with state-action history (HWA2C-xu)
M6	Recurrent A2C with state history (RA2C-x)
M7	Recurrent A2C with state-action history (RA2C-xu)
M8	DOB-Net ($D_{dist} = 4$)
M9	DOB-Net ($D_{dist} = 64$)
M10	Trajectory Optimization **(given the true disturbance waveforms in advance)**

Notice that, among these algorithms, the trajectory optimization assumes that the full knowledge of the disturbances over the whole episode is given in advance, while all other algorithms deal with unknown disturbances. The comparison is obviously not fair, the trajectory optimization is used only to provide optimal performance under ideal conditions, it can be considered as an informal upper bound for all the algorithms. Our goal is to narrow the gap between the performance of the proposed algorithm and that of the

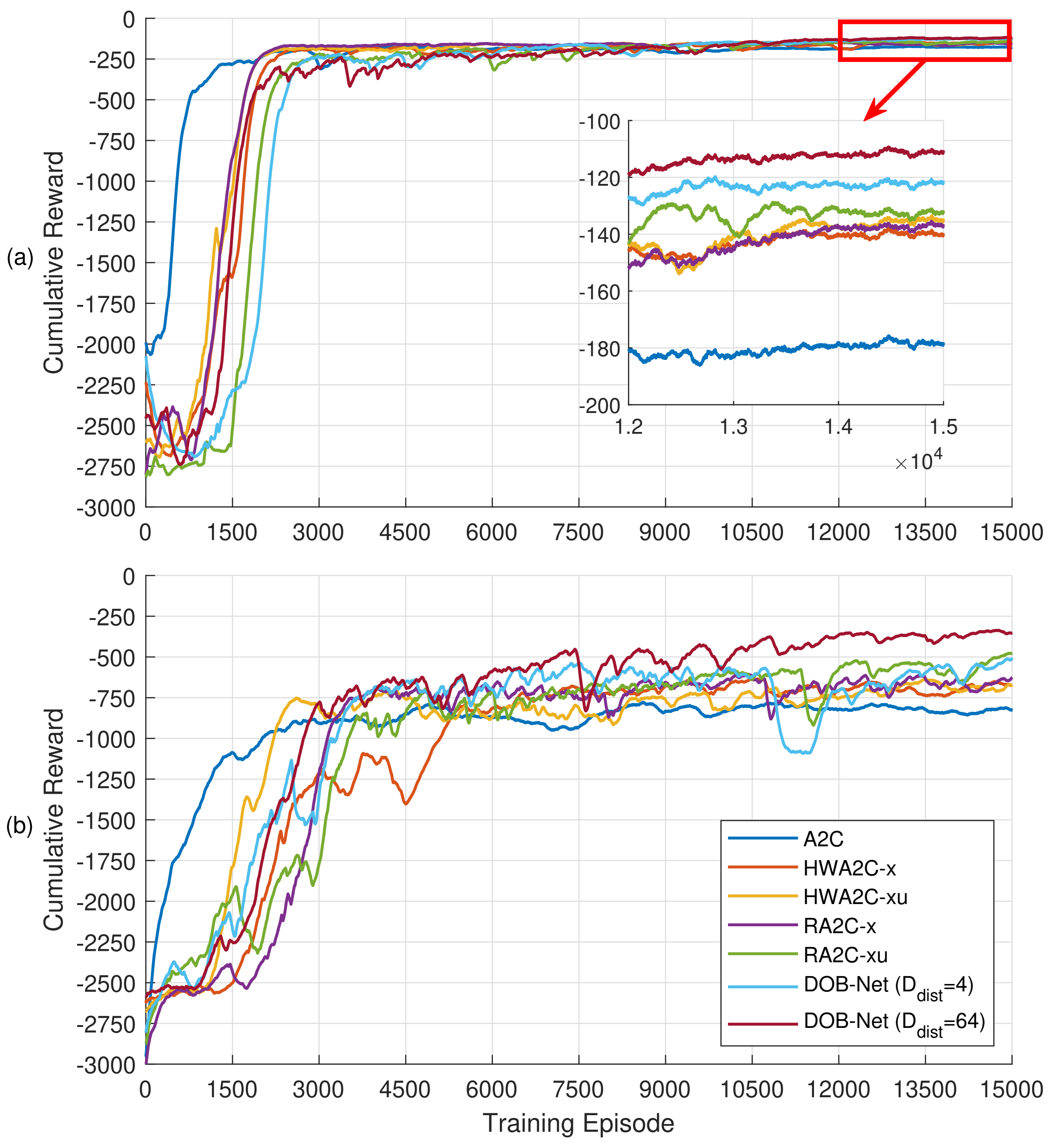


FIGURE 3.15: Training rewards: (a) Type II simulated disturbances; (b) Type III simulated disturbances.

trajectory optimization solution. In this work, the trajectory optimization is implemented by iLQR. RISE control is a conventional feedback controller; DOBC is an implementation of the work presented by Chen et al. (2000), considering control constraints; HWA2C applies the history window formulation into A2C framework, the used window length is 10 timesteps, which is 0.5s in our simulation setup; and RA2C employs RNN to deal with the past states and actions. The applied A2C framework employs a parallel training mode, 16 agents are used in the meantime, the equivalent real-world training time for each agent is 41.67 hours. The RL algorithms accumulate gradients over 20 timesteps, its reward discount factor is 0.99. The used optimizer for RL is RMSprop, where the learning rate is 7e-4 and the max norm of gradients is 0.5. In the remaining part of this section, we first evaluate the training process of different algorithms, then test and compare the control performance among them using both the Type II and Type III simulated disturbances.

Figure 3.15 shows the change of cumulative reward over time during training. But the training reward is not sufficient to compare the performance among different algorithms, we are also interested in state distribution and bounded response of the robot disturbed by underwater waves. As shown in Figure 3.16 and Figure 3.17, we performed 100 episodes using the controllers and trained policies, and recorded the motion data. The box plot is used to represent the distribution of the distance between the robot and the target at each timestep. On each box, the central mark indicates the median, the bottom and top edges

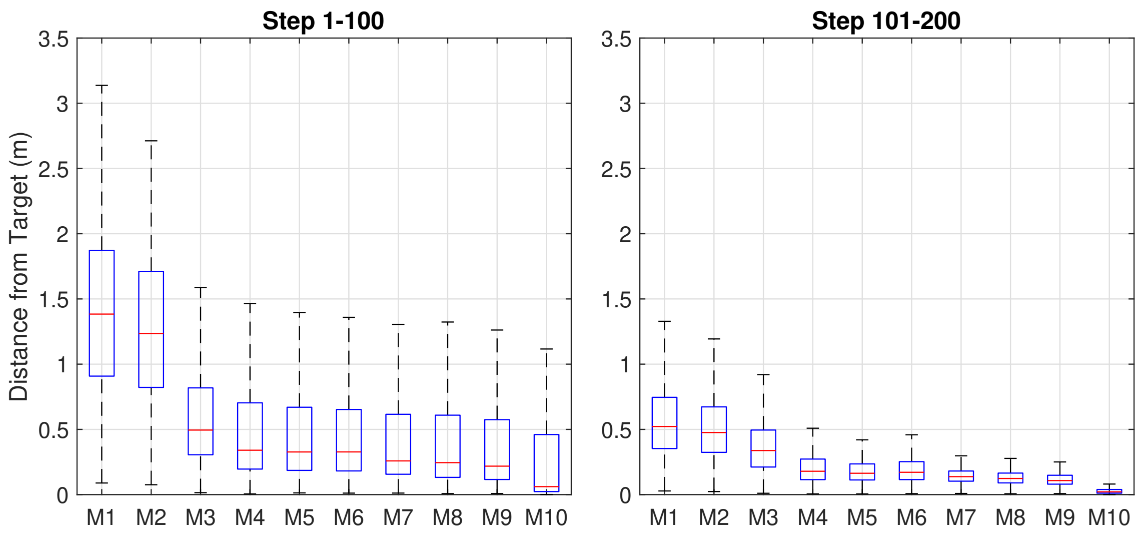


FIGURE 3.16: Distribution of the distance between the robot and the target with Type II simulated disturbances, the algorithms for comparison are given in Table 3.2.

of the box indicate the 25th and 75th percentiles, respectively, and the whiskers extend to the most extreme data points. The results are compared between the first half (step 1-100) and the second half (step 101-200) of each episode, as well as among different algorithms. From the results, we can tell that the distance ranges of the second half episode are smaller than those of the first half, since the robot first observes the disturbance dynamics and then tries to stabilize itself. The results also demonstrate that all these algorithms do stabilize the robot to a certain extend.

Again notice that, the trajectory optimization provides an optimal solution in the case the disturbances through the entire episode are given in advance. Our goal is to narrow the gap between our algorithm and the optimal solution in ideal case. If we focus on the second half of the episode, it can be seen that both the history window policies and the recurrent policies perform better than the classical RL policy (A2C) and the conventional controllers (RISE and DOBC), which means considering history information does improve the disturbance rejection capability. When only moderate disturbances occur (Type II disturbances), different approaches to use the history information have nearly the same results. When the disturbances become larger (Type III disturbances), the recurrent policies achieve better results compared with the history window policies, proving that RNN is able to utilize the history information more efficiently than naively combining past states and actions into the input space of the control policy. Furthermore, including past actions

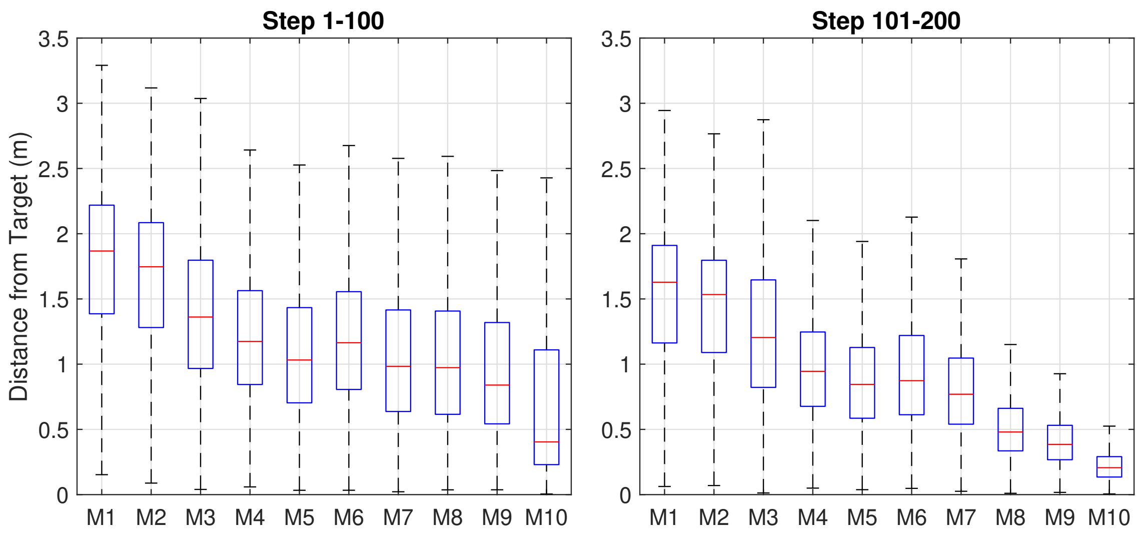
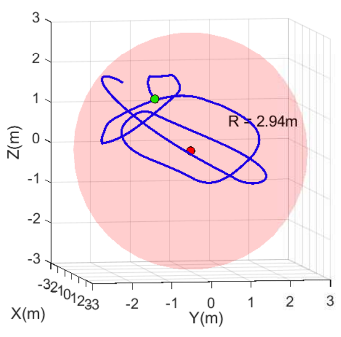
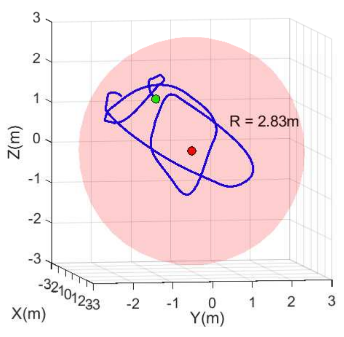


FIGURE 3.17: Distribution of the distance between the robot and the target with Type III simulated disturbances, the algorithms for comparison are given in Table 3.2.

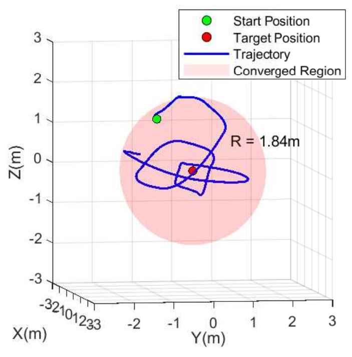
as additional input besides states yields performance improvement for both HWA2C and RA2C. DOB-Net achieves the best performance among all these algorithms, using larger embedding dimension ($D_{dist} = 64$) of disturbance estimate produces higher cumulative reward and smaller movement range during the position regulation task. We believe that enlarging the embedding dimension of disturbance estimate can provide better representation of disturbance waveforms. Transforming this embedding from a 64-dimensional variable to a 4-dimensional variable may cause loss of information. However, even using



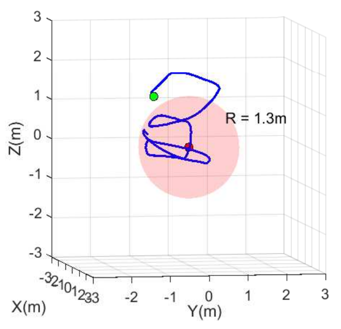
(a) RISE Control



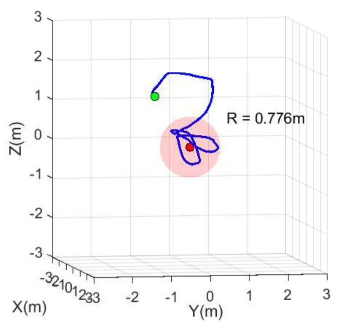
(b) DOBC



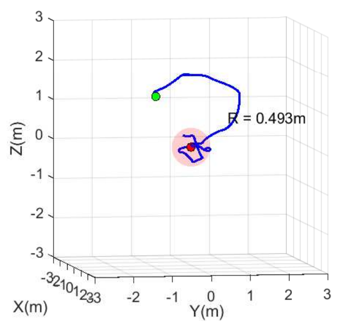
(c) A2C



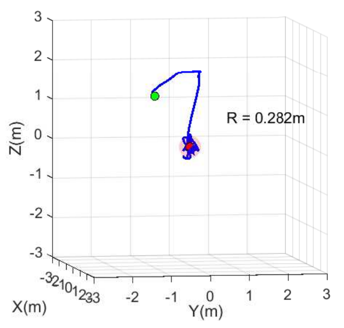
(d) HWA2C-xu



(e) RA2C-xu



(f) DOB-Net ($D_{dist} = 64$)



(g) Trajectory Optimization

FIGURE 3.18: 3D trajectories of the underwater robot with Type III simulated disturbances (R is the radius of the converged region). Note the trajectory optimization assumes that the disturbance dynamics is known in advance, thus provides the ideal performance.

the best RL algorithms we mentioned so far, the control performance still has a large gap from the trajectory optimization solution. There is still room for further improvements.

In addition, it is obvious that stronger disturbances lead to worse performance. But we also found larger amplitude range of disturbances gives a more similar results between DOB-Net and the trajectory optimization (the ratio of medians between DOB-Net and the trajectory optimization is 502.31% and 186.65% respectively for Type II and Type III simulated disturbances). This phenomenon might result from that, for disturbances with larger amplitudes, the optimal controls for different waveforms tend to be more similar. Thus, it is easier for the trained RL policy to generalize over different waveforms under larger disturbance amplitudes.

The 3D trajectories of the robot subject to Type III simulated disturbances are compared among these algorithms in Figure 3.18. The red ball region represents the robot's maximum distance from the target during the last 50 timesteps, called converged region. According to this region, we can see that the robot is difficult to achieve satisfactory bounded response using either conventional controllers (RISE control and DOBC) or classical RL policy (A2C). While the proposed DOB-Net can significantly reduce the converged region. Using DOB-Net, the robot can quickly navigate to the target and stabilize itself within a

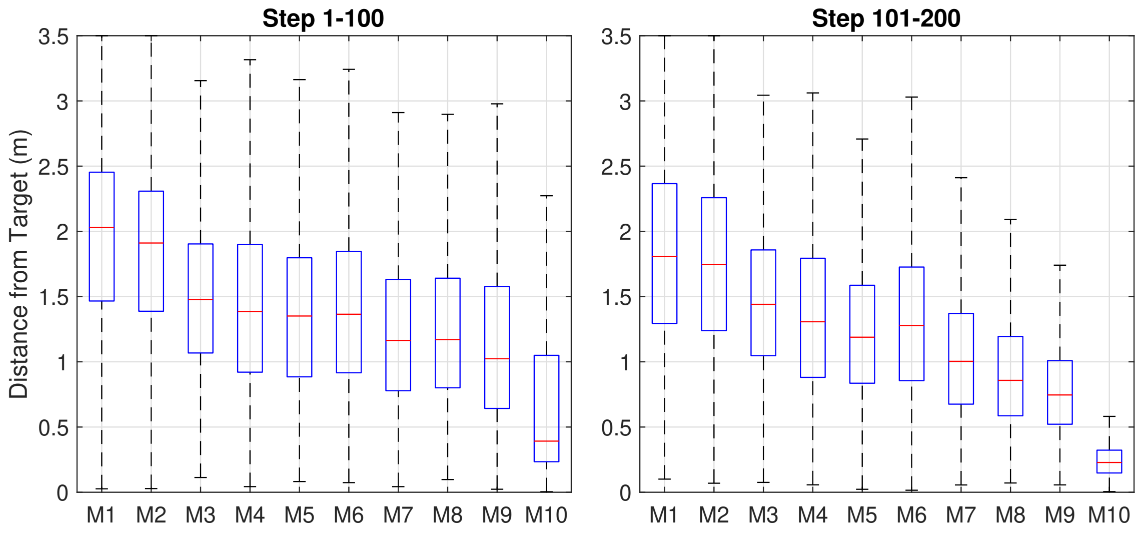


FIGURE 3.19: Distribution of the distance between the robot and the target with real disturbances, the algorithms for comparison are given in Table 3.2.

distance of $0.493m$ from the target thereafter. However, there is still an obvious difference between DOB-Net results and the optimal trajectories under ideal conditions.

3.4.4 Results on Real-World Wave Data

Besides the simulated disturbances, we also use the collected real-world wave data (as shown in Figure 3.10) for test the robustness and generalization of DOB-Net. Note the real disturbances are only used for test purpose, no retraining or fine-tuning is required on such data. As compared in Figure 3.19 and Figure 3.20, DOB-Net still outperforms all the other algorithms on the real-world waveforms, which proves the performance improvement of DOB-Net. But the control stability of these algorithms is apparently worse compared to the test results with the simulated disturbances. Even the trajectory performed by DOB-Net cannot be regarded as satisfactory performance.

There might be multiple reasons why this phenomenon occurs. Firstly, there might not be sufficient training on the simulated disturbances, thus the agent might not fully explore the entire distributions of simulated disturbance parameters when optimizing the control policy. Secondly, the real-world wave data might have more diverse and complicated waveforms, and have a wider range of disturbance parameters (mismatched distributions

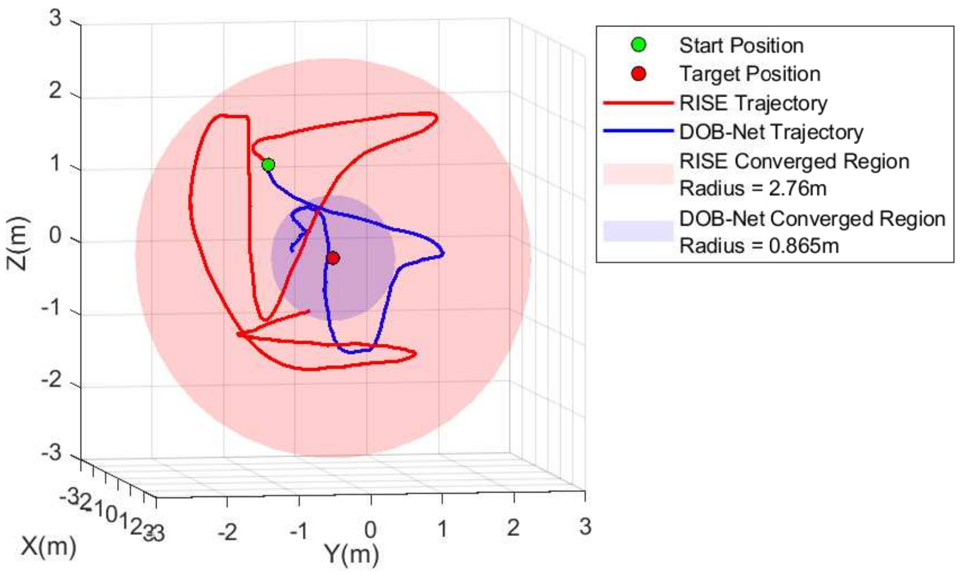


Figure 3.20: 3D trajectories of the underwater with real disturbances (R is the radius of the converged region).

of parameters) compared with the simulated disturbances. DOB-Net might not be capable of handling such unseen disturbance waveforms.

There are several possible ways to address this issue. In regard to insufficient training, we can generate more training samples on the simulated disturbances, or even construct more different distributions of simulated parameters (more than three types). As for mismatched distributions of parameters, a simple idea is to collect real-world wave data that is similar to the simulated disturbances used in training. This can be difficult to implement due to the lack of knowledge of real-world waves. Another way is to adopt transfer learning techniques (Pan and Yang, 2009), which can be used for generalization to real-world data with a wide range of parameters, based on training on simulated data with limited range of parameters.

Chapter 4

Modular Learning of Disturbance Rejection Control with Online Disturbance Identification

Chapter 3 has demonstrated that RNN can directly learn a policy to control a dynamics system with unobservable disturbances in an end-to-end mode, where the past motion history is mapped to the control action. While, inspired by (Yu et al., 2017), this chapter applies modular learning procedures, that explicitly decouple the process into disturbance identification and motion control, instead of joint optimization of both modules.

4.1 Learning a Generalized Control Policy with Online Disturbance Identification

The learning algorithm for disturbance rejection control proposed in this chapter is composed of two main modules, namely a generalized control policy (GCP) and an online disturbance identification model (ODI), as shown in Figure 4.1. Firstly, we build a RL framework to train GCP, $\boldsymbol{u}_t = \pi\left(\boldsymbol{x}_t, \boldsymbol{d}_{t:t+n}\right)$, under a system dynamics model, $\boldsymbol{x}_{t+1} = f\left(\boldsymbol{x}_t, \boldsymbol{u}_t; \boldsymbol{\mu}\right)$, parameterized using the disturbance parameters $\boldsymbol{\mu}$. Unlike classical RL policies where a mapping between states and controls is established, GCP explicitly

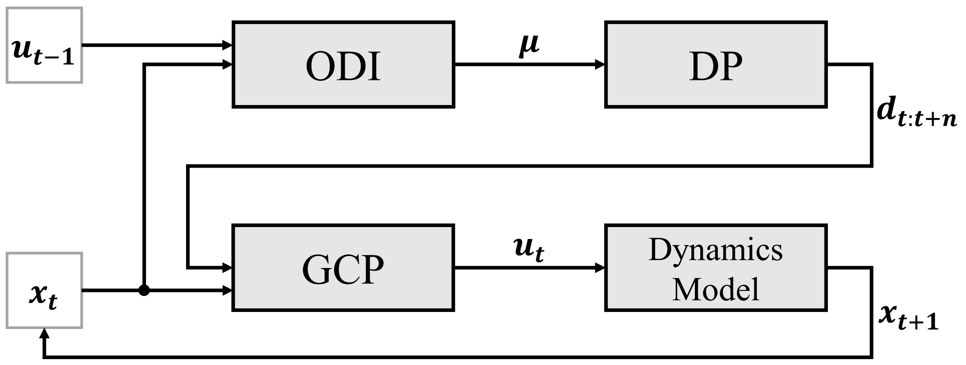


Figure 4.1: Diagram of GCP-ODI. The online disturbance identification model (ODI) employs RNN to identify the disturbance parameters $\boldsymbol{\mu} = (A_1, \omega_1, \phi_1, \cdots, A_k, \omega_k, \phi_k)$ from the past states and actions, and the disturbance prediction model (DP) formulates a sequence of future disturbances $\boldsymbol{d}_{t:t+n}$ from these parameters, where $\boldsymbol{d}_t = A_1 \sin(\omega_1 t + \phi_1) + \cdots + A_k \sin(\omega_k t + \phi_k)$. The generalized control policy (GCP) then takes the predicted future disturbances $\boldsymbol{d}_{t:t+n}$ along with the current state $\boldsymbol{x}_t$ to compute the control action $\boldsymbol{u}_t$.

takes a sequence of future disturbances (i.e. $\boldsymbol{d}_{t:t+n}$) as input besides the current state, and outputs a control signal. This additional input allows the policy to specialize at each set of disturbance waveforms, which might improve the control performance. Secondly, we employ RNN to construct ODI, $(\boldsymbol{\mu}, \boldsymbol{h}_t) = \psi(\boldsymbol{x}_t, \boldsymbol{u}_{t-1}, \boldsymbol{h}_{t-1})$, and train it following a supervised learning style. ODI is able to predict the disturbance parameters $\boldsymbol{\mu}$, given the most recent state-action pair $(\boldsymbol{x}_t, \boldsymbol{u}_{t-1})$ and a hidden state vector $\boldsymbol{h}_{t-1}$ from the last timestep, which encodes the information of past states and actions.

When combining GCP and ODI together, the complete workflow is to use ODI to predict the disturbance parameters $\boldsymbol{\mu}$ based on the current system state $\boldsymbol{x}_t$, the previously executed action $\boldsymbol{u}_{t-1}$ of the robot, and the previous hidden state $\boldsymbol{h}_{t-1}$ as well. The disturbance prediction model (DP) uses these parameters to produce a sequence of future disturbances $\boldsymbol{d}_{t:t+n}$, which are then fed into GCP, with the current system state $\boldsymbol{x}_t$, to generate the control action $\boldsymbol{u}_t$. The action $\boldsymbol{u}_t$ is executed on the robot and the system dynamics gives an updated state $\boldsymbol{x}_{t+1}$, then the algorithm proceeds to the next timestep (Figure 4.1).

We speculate that using the decoupled moderate-sized networks instead of a large network trained by RL in an end-to-end mode, might mitigate the learning difficulty and thus improve the sample efficiency. The modular learning procedures can also benefit the policy transfer under dynamics model mismatch, since only the controller network needs to be modified for adaptation to target dynamics model, yet the identification model can remain fixed, thus reducing the transfer difficulty.

Both GCP and ODI are parameterized as deep neural networks. During training, GCP is trained first to cover all of the possible disturbance waveforms that ODI might explore during following optimization, where there is no need for any initial network or prior knowledge for both modules.

4.1.1 Learning Generalized Control Policy

The disturbance rejection controller is expected to be generalized over a range of disturbance waveforms. The intuition for this problem is to directly train one unified control policy that adapts to all different waveforms. While our initial attempt showed that, the policies change a lot when optimized over different waveforms, and the trained unified policy can hardly succeed at the given task. The reason behind is that the wave forces are too strong to be considered as random noises any more, and the disturbed system dynamics under different waveforms tends to be drastically varied with each other. Thus, training one single policy to deal with a wide variety of dynamics usually leads to poor generalization ability.

Our algorithm in this work is to train a "generalized" policy to specialize at each set of waveforms through constructing the policy network parameterized by $\boldsymbol{\mu}$. After sufficient training, the generalized policy can achieve high cumulative rewards over the space of $\boldsymbol{\mu}$, with comparable performance to the policies trained for a specific $\boldsymbol{\mu}$. However, the disturbance parameters $\boldsymbol{\mu}$ are composed of frequency domain signals $(A_1, \omega_1, \phi_1, \cdots, A_k, \omega_k, \phi_k)$, simply appending these signals to the input state cannot yield a well-performed control policy (see Section 4.2). The reason behind this phenomenon is that the frequency domain signals only provide a description of the waveform, but do not explicitly indicate time information, so that the algorithm cannot figure out where it is along the waveform at each timestep. One improvement is to formulate time domain signals from the frequency domain signals, and use multi-step future disturbances $\boldsymbol{d}_{t:t+n}$ as additional input for the policy, where $\boldsymbol{d}_t = A_1 \sin(\omega_1 t + \phi_1) + \cdots + A_k \sin(\omega_k t + \phi_k)$. Although such formulation

Algorithm 3: Learning Generalized Control Policy

```
Randomly initialize policy network $\pi$
Initialize rollout buffer $R$
Initialize episode step counter $t = 0$
Sample state $\boldsymbol{x}_t \sim p(\boldsymbol{x}_0)$
Sample disturbance parameters $\boldsymbol{\mu} \sim p(\boldsymbol{\mu})$
Initialize disturbance sequence $\boldsymbol{d}_{t:t+n}$ from $\boldsymbol{\mu}$
Insert $(\boldsymbol{x}_t, \boldsymbol{d}_{t:t+n})$ into $R$
while $step \leq MaxStep$ do
    while $R$ is not full do
        Calculate disturbances $\boldsymbol{d}_t$ from $\boldsymbol{\mu}$
        Perform action $\boldsymbol{u}_t = \pi(\boldsymbol{x}_t, \boldsymbol{d}_{t:t+n})$
        Receive next state $\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t; \boldsymbol{\mu})$
        Receive reward $r_t = r(\boldsymbol{x}_t, \boldsymbol{u}_t)$
        Update $\boldsymbol{d}_{t+1:t+1+n}$
        Insert $(\boldsymbol{x}_{t+1}, \boldsymbol{d}_{t+1:t+1+n}, \boldsymbol{u}_t, r_t)$ into $R$
        Update $t \leftarrow t + 1$
        if episode is terminated then
            Reset $t = 0$
            Sample $\boldsymbol{x}_t \sim p(\boldsymbol{x}_0)$
            Sample $\boldsymbol{\mu} \sim p(\boldsymbol{\mu})$
        end
    end
    Update $\pi$ using data in $R$
end
```

does not contain complete information of the disturbances, and only achieving suboptimal solution, it is still proved attaining better control performance.

A2C (Mnih et al., 2016) is used to train GCP. During each episode, the algorithm (Algorithm 3) samples disturbance parameters $\boldsymbol{\mu}_i$ from a uniform distribution $p(\boldsymbol{\mu})$, and constructs a sequence of future disturbances in the time domain $\boldsymbol{d}_{t:t+n}$, then performs the trajectory under the policy $\pi(\boldsymbol{x}_t, \boldsymbol{d}_{t:t+n})$ and the dynamics model $f(\boldsymbol{x}_t, \boldsymbol{u}_t; \boldsymbol{\mu}_i)$. Once the trajectory data are collected, the policy network π will be updated following A2C rules.

4.1.2 Learning Online Disturbance Identification Model

Even though GCP is capable of performing optimal control under different waveforms, the policy can only succeed with the correct disturbance parameters, but this information is usually not easy to obtain. This issue can be addressed by learning an online disturbance identification model (ODI), $\boldsymbol{\mu} = \psi\left(\boldsymbol{x}_t, \boldsymbol{u}_{t-1}, \boldsymbol{h}_{t-1}\right)$, that continuously identifies the correct disturbance parameters for GCP.

The training of ODI can be framed as a supervised learning problem, where the model aims to predict the disturbance parameters $\boldsymbol{\mu}$ with the input being the most recent state-action pair $\left(\boldsymbol{x}_t, \boldsymbol{u}_{t-1}\right)$ and the hidden state $\boldsymbol{h}_{t-1}$ from the last timestep. Note that $\boldsymbol{h}_{t-1}$ is neither the input nor the output of the network, but only an intermediate variable representing the past processed information, in order to enable the network to preserve a memory of the past inputs. The optimization is conducted through minimizing the objective function using SGD:

$$\theta_\psi^* = \underset{\theta_\psi}{\operatorname{argmin}} \sum_{(\boldsymbol{\tau}_i, \boldsymbol{\mu}_i) \in B} \sum_{t=0}^{T-1} \left\|\boldsymbol{\mu}_i - \psi\left(\boldsymbol{x}_t, \boldsymbol{u}_{t-1}, \boldsymbol{h}_{t-1}; \theta_\psi\right)\right\|^2, \tag{4.1}$$

where $\boldsymbol{\tau} = \left(\boldsymbol{x}_0, \boldsymbol{u}_0, \cdots, \boldsymbol{x}_{T-1}, \boldsymbol{u}_{T-1}, \boldsymbol{x}_T\right)$ is a trajectory of states and actions in each episode, and θ_ψ represents the parameters of ODI network ψ. During training, we randomly sample the space of $\boldsymbol{\mu}$ for K times, and we simulate N episodes for each sampled $\bar{\boldsymbol{\mu}}_i$, with the policy $\pi\left(\boldsymbol{x}_t, \bar{\boldsymbol{d}}_{t:t+n}\right)$ and the dynamics $f\left(\boldsymbol{x}_t, \boldsymbol{u}_t; \bar{\boldsymbol{\mu}}_i\right)$. The trajectory data is then stored in the training buffer B and used to update ODI network ψ using Equation 4.1 (Algorithm 4).

After optimizing ODI ψ using Equation 4.1, we noticed that the performance of the combined algorithm, GCP-ODI, was much worse than directly feeding the true disturbance parameters $\bar{\boldsymbol{\mu}}$ to GCP (see Section 4.2). This result is not surprising since the trajectories used for training ODI are generated by performing a control policy $\pi\left(\boldsymbol{x}_t, \bar{\boldsymbol{d}}_{t:t+n}\right)$ under a dynamics model $f\left(\boldsymbol{x}_t, \boldsymbol{u}_t; \bar{\boldsymbol{\mu}}\right)$, where they both use the same disturbance parameters $\bar{\boldsymbol{\mu}}$ in their formulation. However, when ODI is deployed with GCP, due to the lack of information at the initial timestep and the regression error in the neural network, ODI will inevitably make some error in the prediction. This error tends to be more and more aggravated over time because the next state-action pair fed to ODI is generated by the control policy $\pi(\boldsymbol{x}_t, \hat{\boldsymbol{d}}_{t:t+n})$ using an incorrectly predicted disturbance parameters $\hat{\boldsymbol{\mu}}$, under the dynamics model with the true disturbance parameters $f\left(\boldsymbol{x}_t, \boldsymbol{u}_t; \bar{\boldsymbol{\mu}}\right)$.

Algorithm 4: Learning Online Disturbance Identification Model

```
Randomly initialize ODI netowrk $\psi$
Initialize training buffer $B$
Initialize iteration counter $iter = 1$
while $\psi$ is not converged do
    for $i = 1 : K$ do
        Sample $\bar{\boldsymbol{\mu}} \sim p(\boldsymbol{\mu})$
        for $j = 1 : N$ do
            Sample $\boldsymbol{x}_0 \sim p(\boldsymbol{x}_0)$
            for $t = 0 : T - 1$ do
                Calculate $\bar{\boldsymbol{d}}_t$ from $\bar{\boldsymbol{\mu}}$
                if $iter == 1$ then
                    Calculate $\bar{\boldsymbol{d}}_{t:t+n}$ from $\bar{\boldsymbol{\mu}}$
                    Perform $\boldsymbol{u}_t = \pi\left(\boldsymbol{x}_t, \bar{\boldsymbol{d}}_{t:t+n}\right)$
                end
                else if $iter > 1$ then
                    Predict $(\hat{\boldsymbol{\mu}}, \boldsymbol{h}_t) = \psi(\boldsymbol{x}_t, \boldsymbol{u}_{t-1}, \boldsymbol{h}_{t-1})$
                    Calculate $\hat{\boldsymbol{d}}_{t:t+n}$ from $\hat{\boldsymbol{\mu}}$
                    Perform $\boldsymbol{u}_t = \pi(\boldsymbol{x}_t, \hat{\boldsymbol{d}}_{t:t+n})$
                end
                Receive $\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t; \bar{\boldsymbol{\mu}})$
            end
            Form trajectory $\boldsymbol{\tau} = (\boldsymbol{u}_0, \boldsymbol{x}_1, \cdots, \boldsymbol{u}_{T-1}, \boldsymbol{x}_T)$
            Store $(\boldsymbol{\tau}, \bar{\boldsymbol{\mu}})$ in $B$
        end
    end
    Optimize $\psi$ using data in $B$
    Update $iter \leftarrow iter + 1$
end
```

In order to solve this issue, one possible idea is to iteratively train ODI by using mismatched disturbance parameters for the control policy $\pi(\boldsymbol{x}_t, \hat{\boldsymbol{d}}_{t:t+n})$ and the system dynamics $f(\boldsymbol{x}_t, \boldsymbol{u}_t; \bar{\boldsymbol{\mu}})$. In each iteration, more training examples are generated following the previous procedures, but the disturbances used by GCP now come from the predicted parameters of ODI $\hat{\boldsymbol{\mu}}$, rather than the true disturbance parameters $\bar{\boldsymbol{\mu}}$ used in the system dynamics. Then, ODI is trained again through combining the mismatched training samples with previously gathered ones according to Equation 4.1. After a small number of iterations, the combined system, GCP-ODI, achieves close performance with GCP that is fed with the true disturbance parameters $\bar{\boldsymbol{\mu}}$.

4.2 Simulation

4.2.1 Simulation Setup

Normally, an underwater robot has six Degree of Freedom (DOF). In this research, only the horizontal motion and control (X, Y, yaw) of the robot are considered. We build a mathematical model of the underwater robot that enables large scale training in simulation. The model is assumed to has the mass of $60kg$ and a simplified cuboid shape with the dimension of $0.68 \times 0.75 \times 0.19m^3$. Both hydrodynamics and control latency are excluded. Due to the horizontal movement, the robot has a 6-dimensional state space $\boldsymbol{X} \in \mathcal{R}^6$ and a 3-dimensional action space $\boldsymbol{U} \in \mathcal{R}^3$. The control constraints are given as $|\overline{\boldsymbol{u}}| = |\underline{\boldsymbol{u}}| = [112N\ 112N\ 82Nm]^T$.

The performance of the designed algorithms is tested through a pose regulation task, where the robot starts with random pose and velocity in each episode, the goal is to control the robot to reach a target pose then stabilize itself under the external wave disturbances. The wave disturbances used in the simulation are constructed as a superposition of multiple sinusoidal waves with different amplitudes, frequencies and phases. This chapter applies

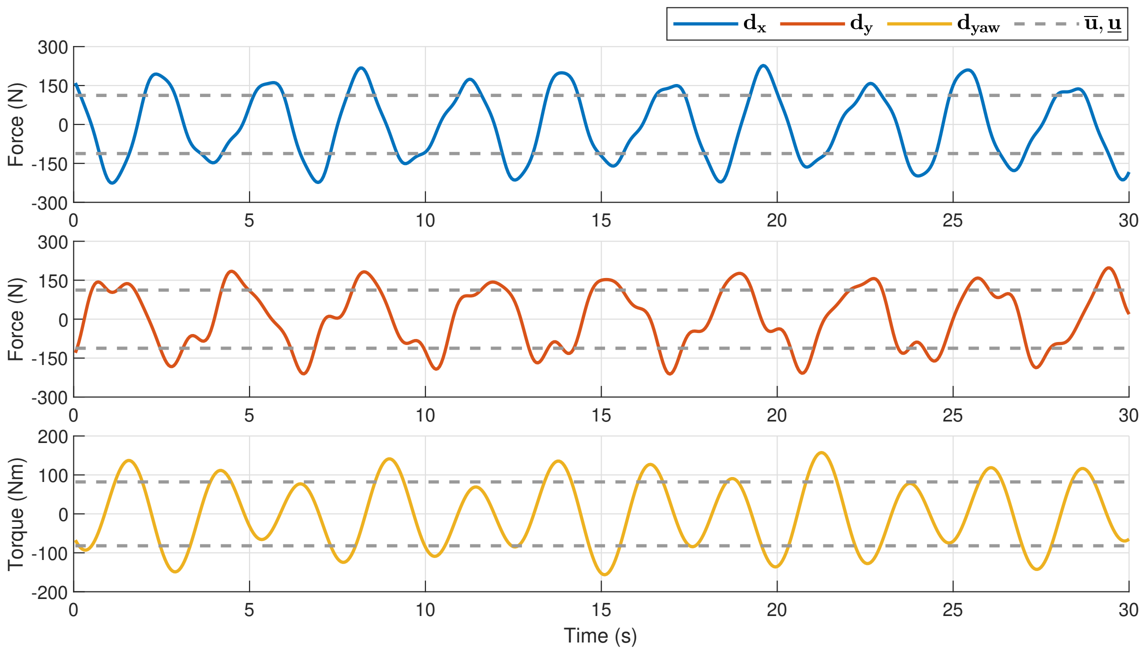


FIGURE 4.2: Example simulated wave disturbances (Type III) in X, Y and yaw directions.

the Type III simulated disturbances, whose parameter distributions and waveforms are shown in Table 3.1 and Figure 4.2, with X, Y and yaw directions being considered. The policy is trained to actively reject the unknown time-correlated wave disturbances, thus their amplitudes, frequencies and phases are randomly sampled from the given distributions in each episode. During training, each episode contains 200 timesteps with 0.05s per timestep. The adopted A2C framework employs a parallel training mode through using 16 agents synchronously, the equivalent real-world training time for each agent is 16.67 hours.

4.2.2 Results

As shown in Figure 4.3, it can be seen that training a RL policy with disturbances $\boldsymbol{d}_{t:t+n}$ or disturbance parameters $\boldsymbol{\mu}$ as additional input can achieve better performance, compared

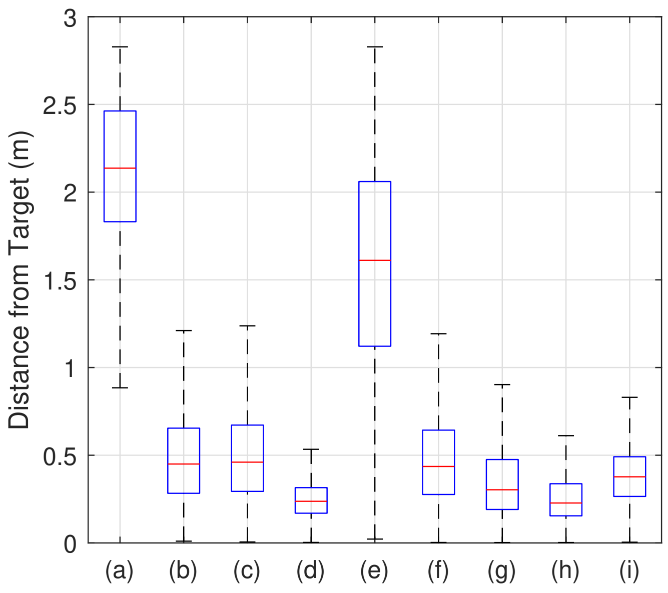


FIGURE 4.3: Distribution of the distance between the robot and the target during last 100 timesteps for GCP-ODI: (a) A2C; (b) GCP-true using as input the frequency domain signals $(A_1, \omega_1, \phi_1, \cdots, A_k, \omega_k, \phi_k)$; (c) GCP-true using as input the processed frequency domain signals $(A_1, \omega_1, \omega_1 t + \phi_1, \cdots, A_k, \omega_k, \omega_k t + \phi_k)$; (d) GCP-true using as input the time domain signals $\boldsymbol{d}_{t:t+n}$; (e) GCP-ODI at the 1st iteration; (f) GCP-ODI at the 2nd iteration; (g) GCP-ODI at the 3rd iteration; (h) GCP-ODI at the 4th iteration; (i) DOB-Net.

with conventional RL policy (A2C). In terms of the part of policy input space representing disturbances, we found that using time domain signals outperforms using frequency domain signals, even though the time information is combined with the frequency domain parameters $\boldsymbol{\mu} = (A_1, \omega_1, \omega_1 t + \phi_1, \cdots, A_k, \omega_k, \omega_k t + \phi_k)$. This result indicates that it might be difficult for the RL to find an effective mapping between the frequency domain and the time domain. Thus, although the time domain signals only contain partial information of the disturbances (i.e. only n timesteps of future disturbances), adopting these variables still perform well due to a more direct representation.

We also found that, the robot hardly stabilize itself near the target when ODI is only trained on the initially collected data (1st iteration). This is because these data is generated by a control policy and a dynamics model with consistent disturbance parameters $\bar{\boldsymbol{\mu}}$, but there are actually mismatched disturbance parameters $\bar{\boldsymbol{\mu}} \neq \hat{\boldsymbol{\mu}}$ during the online operation of GCP-ODI. This situation can be mitigated through iteratively alternating between gathering more data with the current ODI and GCP, and retraining ODI using the aggregated data. After several iterations (4 in our case), the disturbance rejection capability of the robot reaches a relatively high level, approaching the performance of

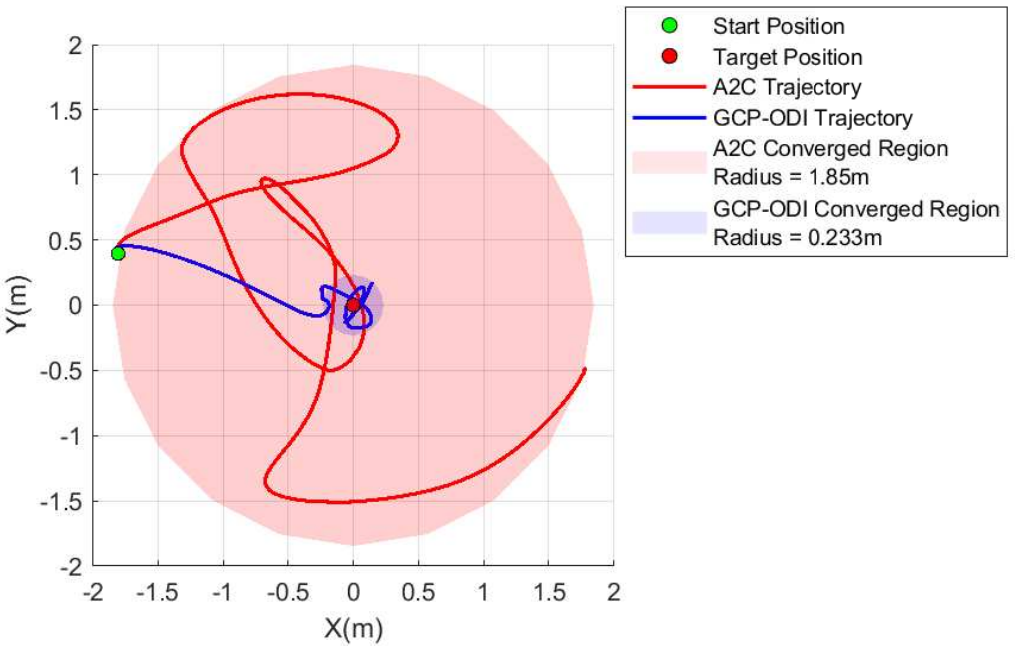


FIGURE 4.4: Trajectories of the robot using A2C and GCP-ODI with simulated dynamics model.

GCP when given the true disturbance parameters $\bar{\boldsymbol{\mu}}$. In addition, we found that DOB-Net also achieves an excellent stability under the same task settings, but not as good as the iteratively trained GCP-ODI. This result is reasonable, because GCP explicitly takes the values of disturbance forces as input, instead of implicitly encoding the prediction of disturbances into the hidden state of RNN, as is the case in end-to-end learning. Such additional input information allows the policy to better specialize at different disturbance waveforms, thus potentially improves the control performance.

Figure 4.4 provides a more intuitive illustration of the algorithm effectiveness, by comparing the trajectories of the robot using the conventional A2C and the well trained GCP-ODI (after 4 iterations of training). We set a performance metric to be the range of the robot's distance to the target during last 100 timesteps of an episode, refer to as converged region. It is found that GCP-ODI can dramatically improve the control stability of the robot under the excessive time-correlated disturbances, the converged region can be reduced from $1.85m$ to $0.233m$.

Chapter 5

Transferring Disturbance Rejection Policy with Hybrid Policy Adaptation

The application of pure learning procedures on the desired model is limited when there are expensive training samples using physical systems or unpredictable variations in the system dynamics. This gives rise to an idea of training a control policy on an initial guess of the desired model, which is supposed to be static and easy to get, then transfer the trained policy to the actual one to bridge the model mismatch.

Thus, two learning algorithms are proposed, which are HPA and TML, for transferring disturbance rejection policy under dynamics model mismatch. HPA is developed on the basis of the joint learning scheme in Chapter 3, and regards the external disturbances as parts of the system dynamics, so that these two sources of model mismatch are adapted together during transfer. While TML is built upon the modular architecture of GCP-ODI in Chapter 4, with the ability of ODI to predict the disturbance waveforms in the target task, thus separating the external disturbances from the system dynamics and reducing the total model mismatch to be adapted. In this chapter, we first focus on HPA, and present two implementations of this algorithm.

Model-free RL algorithms have been shown to be capable of learning a wide variety of robotic skills, from dexterous manipulation (Andrychowicz et al., 2020) to mapless navigation (Tai et al., 2017). But such algorithms suffer from high sample complexity, often requiring a large amounts of training samples to achieve satisfactory performance. Model-based learning algorithms are generally considered to be more sample efficient. When combining with some prior knowledge, such as a dynamics model or a conventional controller, RL can improve its sample efficiency. To this end, a hybrid framework of model-based and model-free learning, HPA, is introduced to transfer a control policy trained in the source task to succeed in the target task, so that the sample efficiency of learning the target task is improved compared with learning from scratch without transfer.

HPA first optimizes a control policy in the source task, through training a deep neural network to imitate the behaviours of iLQR, which uses a mathematical model of the system dynamics. Two types of the mathematical model are applied, with or without the consideration of simulated disturbances. The purpose is to test the difference on policy adaptation in the target task when the simulated disturbances are included or excluded in the source task. Because we do not have an accurate model for the real-world wave forces, there can be discrepancies between the simulated waveforms and the real ones. Thus it is uncertain whether including the simulated disturbances in the source task will reduce the model mismatch to be adapted in the target task or not. After that, the policy adaptation in the target task can be implemented by either training a compensatory policy in parallel with the model-based policy, or using the model-based policy to initialize a model-free learner, which is further fine-tuned in the target task.

5.1 Problem Formulation

The key idea of transfer learning (Taylor and Stone, 2009) is that experience or knowledge obtained from learning to specialize in one or more task(s) can help improve learning effectiveness in one or more different but related task(s). The former is called source task (domain), the latter is called target task (domain). In general, task and MDP are used interchangeably.

In this research, the source and target tasks differ in system dynamics. The source task defines a **mathematical model** of SPIR developed from the fundamental principles of dynamics. The target task applies an **empirical model** of SPIR derived from real-world experimental data. Specifically, the empirical model consists of two parts, the first part is a deep neural network learned from real-time motion data of the robot system in still water, representing real robot dynamics; the other part is force data of real-world ocean waves collected in open water, representing external disturbances. But in the meantime, the state and action spaces, the reward and objective functions between the source and target tasks are all kept consistent. Furthermore, the information transferred between the tasks is control policy. For this thesis, the following definitions are applied:
source policy: the policy directly trained on the source task;
target policy: the policy directly trained on the target task;
unadapted policy: the policy trained on the source task then directly applied on the target task without transfer learning;
adapted policy: the policy trained on the source task then further trained on the target task using transfer learning.

There are many metrics to measure the benefits of transfer, such as jumpstart, asymptotic performance, total reward, transfer ratio, and time to threshold (Taylor and Stone, 2009). In this work, we use jumpstart and learning time to measure the success of transfer. That is, the goals of transfer learning are to improve the initial performance of an agent in a target task, and to reduce the learning time required by the agent to achieve optimal performance, compared with learning from scratch in the target task. Learning time is regarded as a surrogate for sample complexity, which refers to the amount of data required by a learning algorithm to converge. These two concepts are strongly correlated, because RL agents collect data merely through repeated interactions with an environment.

5.2 Model-Based Policy Optimization

Trajectory optimization defines a process of finding a trajectory of states and controls that optimizes a given objective function Tassa et al. (2014). iLQR Li and Todorov (2004); Todorov and Li (2005) is a typical trajectory optimization algorithm, that applies to nonlinear systems and requires only first-order derivatives of the dynamics. The model-based controller is obtained using iLQR (Li and Todorov, 2004; Todorov and Li, 2005) optimized on a mathematical model of SPIR, which is given as follows:

$$\begin{aligned} \boldsymbol{M}_{RB}\dot{\boldsymbol{\nu}} + \boldsymbol{C}_{RB}(\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{g}_{RB}(\boldsymbol{\eta}) &= \boldsymbol{u} + \boldsymbol{d}, \\ \dot{\boldsymbol{\eta}} &= \boldsymbol{J}\boldsymbol{\nu}, \end{aligned} \tag{5.1}$$

where the hydrodynamics effects, such as added mass and inertia $\boldsymbol{M}_A$ and $\boldsymbol{C}_A$, and damping effects $\boldsymbol{D}_{RB}$, are excluded; the model uncertainties $\boldsymbol{\xi}$ are assume to be zero; and the disturbance term $\boldsymbol{d}$ can be either included or excluded, depending on the purpose of model-based control. The cost function l is defined in Equation 3.5, we recall it here as:

$$l\left(\boldsymbol{x}_t, \boldsymbol{u}_t\right) = \boldsymbol{x}_t^T \boldsymbol{Q} \boldsymbol{x}_t + \boldsymbol{u}_t^T \boldsymbol{R} \boldsymbol{u}_t, \tag{5.2}$$

where $\boldsymbol{Q}$ is the quadratic state cost matrix, $\boldsymbol{R}$ is the quadratic control cost matrix. The cost function l is given as a negative value of the reward function Equation 3.9 used in RL, this setting is reasonable because both trajectory optimization and RL optimize the same goal on the pose regulation task. We then optimize the sequence of control actions $\boldsymbol{U}$ over a whole trajectory with length T through Equation 3.6, using the given mathematical model Equation 5.1 to predict future states. When optimizing the control sequence, we also consider the control constraints with element-wise inequality, as given in Equation 3.7. The box constraint accurately describes nearly any set of standard mechanical actuators, and allows us to use a specialized active-set algorithm which is more efficient and easier to implement (Tassa et al., 2014). An algorithm called box quadratic programming (box-QP) (Tassa et al., 2014) is applied to accommodate box inequality constraints on the controls, without significantly sacrificing convergence quality or computational effort.

Trajectory optimizers are normally computationally expensive, since they need to solve an optimization problem every time they meet new initial states $\boldsymbol{x}_0$, which make them not suitable for real-time operation. However, constructing a neural network to produce similar results can lead to faster calculation of the control signals, each output control action only consumes the time of one forward propagation of the neural network. Thus, we use neural networks to represent a control policy that imitates the behaviours of the model-based controller iLQR. As shown in Figure 5.1, the example trajectories are first generated using iLQR, based on the given dynamics function (Equation 5.1) and the cost function (Equation 3.5). The trajectories are then collected into a dataset $\mathcal{D}_{LQR}$, and then an imitation policy π is trained to match these expert trajectories in $\mathcal{D}_{LQR}$. Notice that, if we use the mathematical model without simulated disturbances, classical RL policy constructed using feedforward networks $\pi\left(\boldsymbol{u}_t|\boldsymbol{x}_t;\theta_\pi\right)$ is sufficient; if we include the simulated disturbances into the dynamics model, then we need to build a recurrent policy (DOB-Net) $\pi\left(\boldsymbol{u}_t|\boldsymbol{x}_t,\boldsymbol{u}_{t-1},\boldsymbol{h}_{t-1};\theta_\pi\right)$ for the optimal control, where $\boldsymbol{h}_{t-1}$ is the recurrent hidden state from the last timestep. The policy is trained using the behavioral cloning objective (Nagabandi et al., 2018b):

$$\min_{\theta_\pi} \sum_{(\boldsymbol{x}_t,\boldsymbol{u}_t)\in\mathcal{D}_{LQR}} \left\|\boldsymbol{u}_t - \pi\left(\boldsymbol{x}_t;\theta_\pi\right)\right\|^2, \tag{5.3}$$

or

$$\min_{\theta_\pi} \sum_{\boldsymbol{\tau}_i\in\mathcal{D}_{LQR}} \sum_{t=0}^{T-1} \left\|\boldsymbol{u}_t - \pi\left(\boldsymbol{x}_t,\boldsymbol{u}_{t-1},\boldsymbol{h}_{t-1};\theta_\pi\right)\right\|^2, \tag{5.4}$$

which we optimize using SGD.

5.3 Hybrid Policy Adaptation

The model-based policy can successfully perform the task under the mathematical model after training, while may not be able to stabilize the robot under the empirical model due to the mismatch in the system dynamics. Thus, transfer learning between the two tasks is necessary, a hybrid model-based and model-free learning algorithm, HPA, is proposed. According to whether the source task includes the simulated disturbances into the mathematical model or not, two different scenarios for HPA are presented. When a feedforward policy is obtained through model-based policy optimization using the mathematical model without the simulated disturbances, this policy will have different dimension of input space with respect to the following model-free learner in the target task, since the model-free RL needs to use history data to deal with the excessive disturbances. Thus, it is not feasible to use the model-based policy as an initialization of the model-free policy. Instead, the

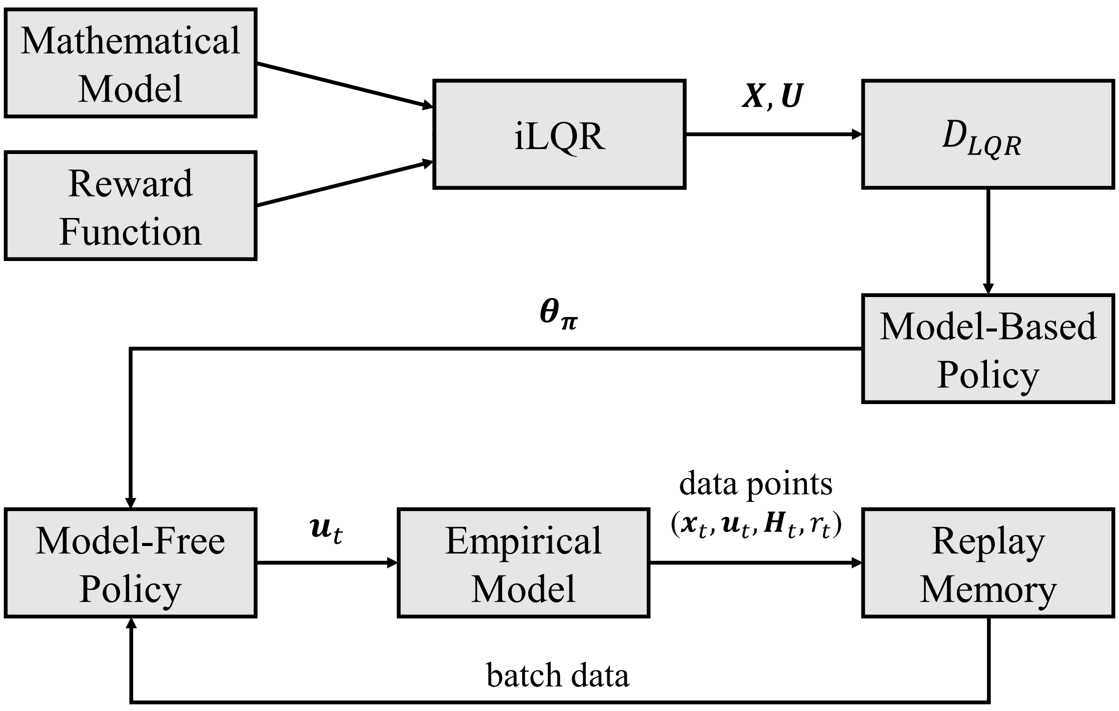


FIGURE 5.1: Working flow of model-based initialization of model-free reinforcement learning (HPA-init).

model-free policy can run with the model-based policy in parallel as a compensatory signal, with only the parameters of model-free policy trainable, and those of the model-based policy remain fixed. While considering the simulated disturbances during model-based learning leads to a recurrent policy, and it is able to initialize the model-free learner for further fine-tuning in the target task.

5.3.1 Learning Model-Free Compensatory Policy

We first consider the situation that the mathematical model does not consider the simulated disturbances, then we only need a feedforward policy that outputs a control vector based on the current system state. Having the model-based policy trained on iLQR trajectories, we then build a parallel architecture for HPA policy (Figure 5.2), consisting of the model-based policy with fixed network parameters and a trainable model-free RL policy, which we use DOB-Net to represent here. The latter one serves as a compensation term for the output of the model-based policy, in order to produce optimal control actions

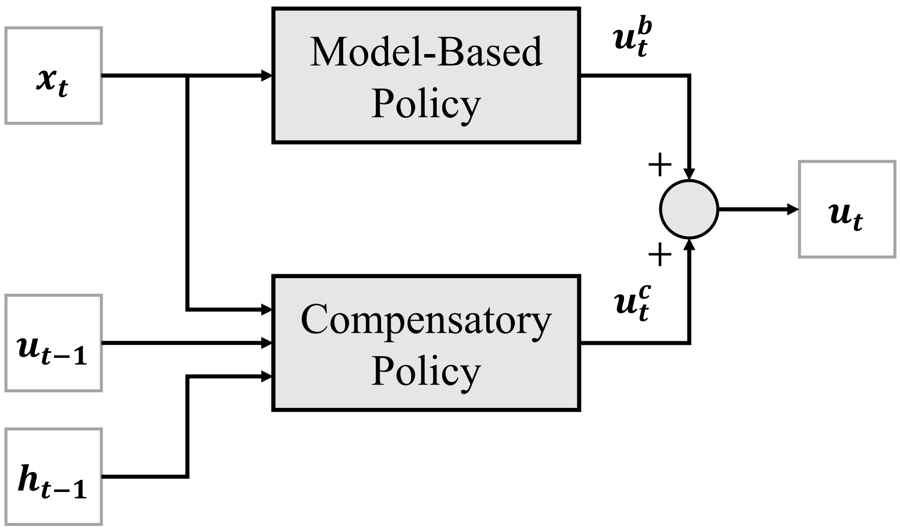


Figure 5.2: Compensation of model-based policy with model-free policy (HPA-comp).

to stabilize the robot under excessive disturbances in the target task. This algorithm is referred to as HPA-comp.

The training process is similar to Algorithm 2, but only the compensatory policy is trainable. Randomly initialize the network parameters of the compensatory policy encourages exploratory actions of the agent, thus leading to worse performance compared to the model-based policy at the beginning of policy adaptation. Due to the given well trained model-based policy, the model-free RL can start with zero output. But in order to avoid ineffective training, we initialize neural network weights $\boldsymbol{W}$ with random numbers close to zero ($\boldsymbol{W} \sim \mathcal{N}(0, 0.1)$) and initialize bias $\boldsymbol{b}$ to zero. Then, HPA policy uses the combined control actions $\boldsymbol{u}_t = \boldsymbol{u}_t^b + \boldsymbol{u}_t^c$ in the target task for data points sampling, and these data points $(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{H}_t, r_t)$ are collected in a replay memory R, then several tuples are grabbed each time from the replay memory to train the compensatory policy.

5.3.2 Model-Based Initialization of Model-Free Reinforcement Learning

When the mathematical model takes the simulated disturbances into consideration, the model-based policy is required to have the same recurrent formulation as the model-free policy, both of them are formulated as DOB-Net. As shown in Figure 5.1, we can combine the advantages of model-based and model-free learning through using a model-based policy as an initialization for a model-free policy. This algorithm is referred to as HPA-init. This combination can be simply implemented by training a policy on the trajectories generated by a model-based controller to imitate its behaviours, then using the well trained imitation policy (i.e., model-based policy) to initialize the network parameters of a model-free policy, which is then further fine-tuned on the target task using much less samples. Specifically, DOB-Net to formulate the policies, though our framework could also be combined with other model-free RL algorithms as long as they consider history to deal with excessive disturbances. DOB-Net outperforms other learning-based controllers on the problem of excessive disturbance rejection control, as evidenced in Chapter 3. Although this approach of combining model-based and model-free learning algorithms is extremely simple, the effectiveness of this approach has been demonstrated between two different dynamics models on a simulated pose regulation task.

5.4 Construction of Empirical Model

Due to the difficulty to create the excessive time-correlated disturbances in the water tank, we then collect real-time motion data of SPIR and train a deep neural network to represent the dynamics model of the real robot, which is used to define the target task, instead of directly deploying the transfer learning algorithm on the real robot.

5.4.1 Data Collection

We first build a visual localization system for the underwater robot SPIR in the water tank. As shown in Figure 5.3, a set of QR code markers is printed on a $1.5 \times 1.5m$ waterproof vinyl banner, which is clamped on an aluminum frame and put on the bottom of the water tank. The aluminum frame is used to keep the marker array in place, otherwise the water waves will drag it back and forth, then affect the localization. A downward-looking camera is mounted to the robot for marker detection and tracking. The camera is well calibrated underwater. We implemented a visual tracking algorithm on the camera to estimate the pose of the marker array from the camera image. In this work, the origin of the coordinate system associated with the markers is set to be the center of the marker

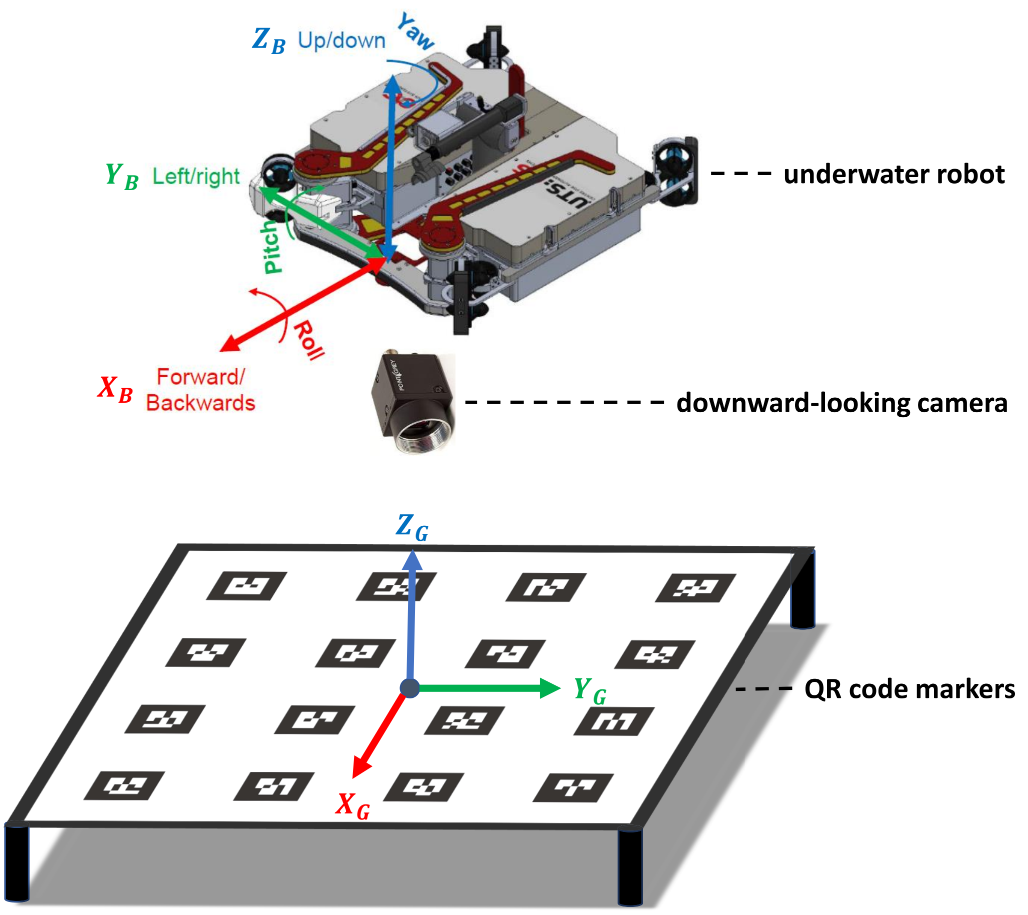


FIGURE 5.3: Diagram of the visual localization system.

array, with its Z-axis vertically upward, this coordinate system is regarded as global frame $\Sigma_G : O_G - X_G Y_G Z_G$. The camera frame is defined as $\Sigma_C : O_C - X_C Y_C Z_C$. The pose of the object can be estimated even though there are markers that are not visible on the camera, actually the pose can be estimated as long as at least one marker is captured in the camera. The visual tracking algorithm is able to calculate and record the marker pose at $30Hz$. Figure 5.4 shows the software interface for a test case out of water, where we use a 5 marker array. We can see that the algorithm is able to detect the relative pose of the markers with respect to the camera, even if only partial markers can be detected. Figure 5.5 provides a demonstration of the field test when the SPIR localizes itself over the markers in the water tank.

The Inertial Measurement Unit (IMU) data is also recorded at $200Hz$, consisting of linear acceleration and angular velocity. The IMU frame is defined as $\Sigma_I : O_I - X_I Y_I Z_I$. The

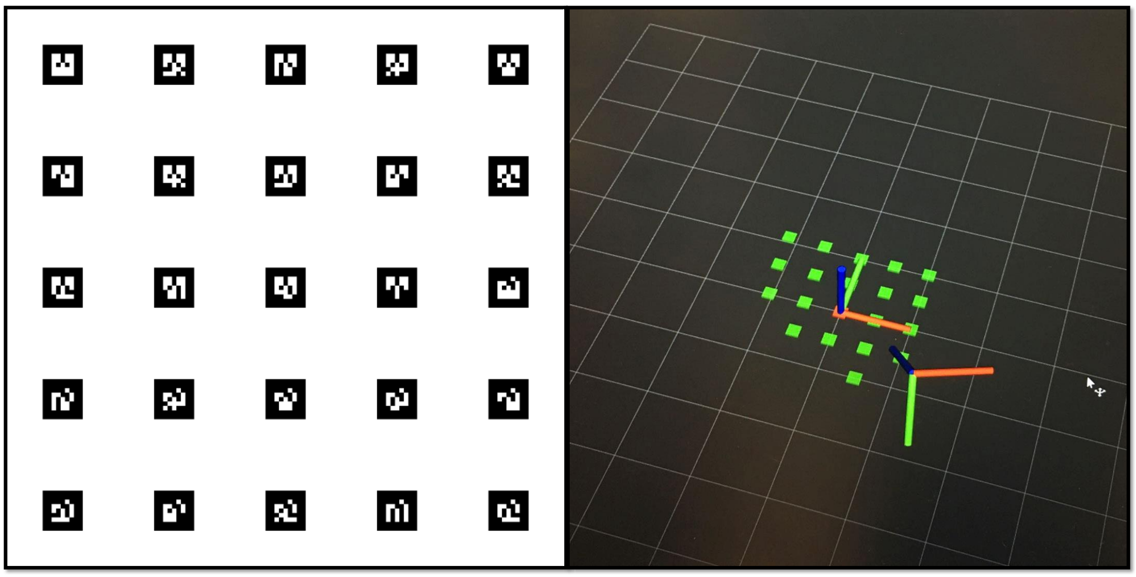


FIGURE 5.4: Software interface of the visual localization system.

underwater robot SPIR is controlled by 12 thrusters, which are T200 Thruster from Blue Robotics, the detailed layout is shown in Figure 5.6. The control data of each thruster is directly read from the onboard computer in the form of PWM signal $\boldsymbol{u}_{pwm}$ at $20Hz$, such signals are then converted into thrust forces $\boldsymbol{u}_{th}$ according to the thruster dynamics described in Figure 5.7, when operating at $24V$. After that, there is a linear relationship between the thrust forces $\boldsymbol{u}_{th}$ and the forces/torques exerted on the robot's center of mass $\boldsymbol{u}$:

$$\boldsymbol{u} = \boldsymbol{B}\boldsymbol{u}_{th}, \tag{5.5}$$

where $\boldsymbol{B} \in \mathbb{R}^{6 \times n_{th}}$ is a constant matrix known as the Thruster Control Matrix (TCM), and $n_{th} = 12$ for the SPIR. The TCM is constructed by the transformation from individual thruster frame to robot body frame $\Sigma_B : O_B - X_B Y_B Z_B$, whose origin is at the robot's center of mass.

During data collection, the robot moves in a fully autonomous mode and executes random control actions at each timestep. Since we are only interested in the horizontal motion and control (X, Y, yaw) of the robot, the roll, pitch and depth of SPIR are regulated from the feedback signals of IMU and depth sensor. In order to ensure safe operation, an additional control term is employed when necessary to keep the robot away from the tank wall. We continuously record the motion and control data for over 4 hours.

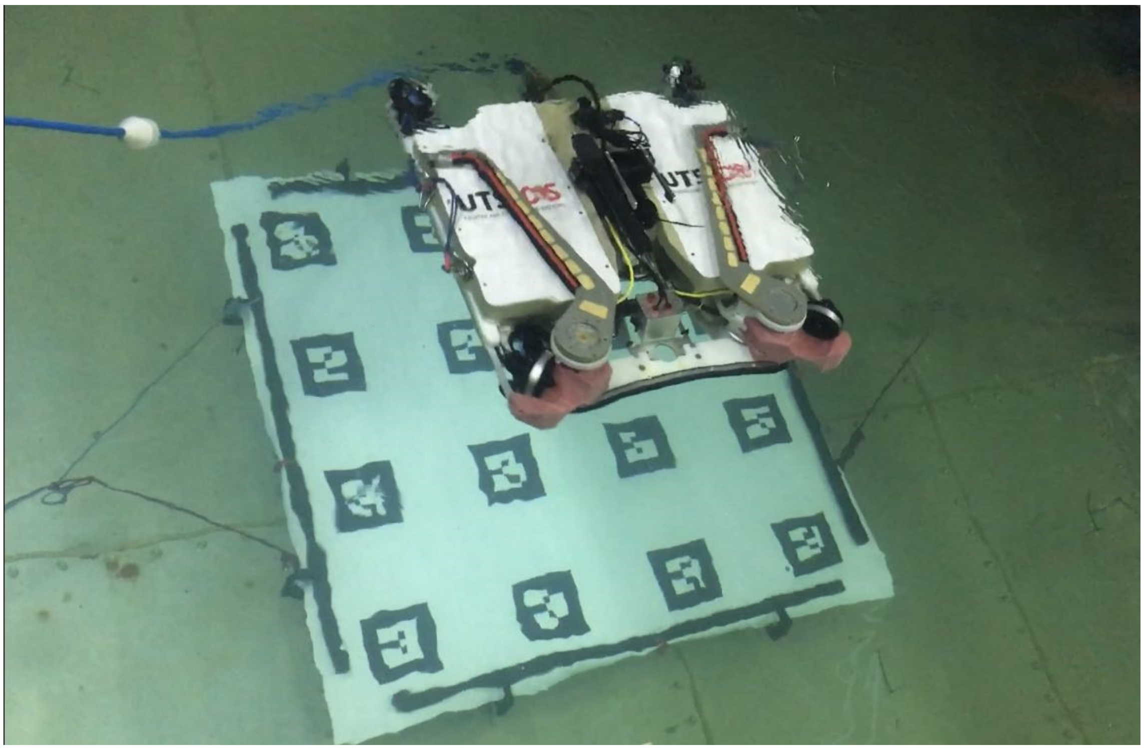


Figure 5.5: Field test of the SPIR over the QR code markers.

5.4.2 Kalman Filtering

The online data collection provides us with the raw sensor data expressed in their own frame. We then employ Extended Kalman Filter (EKF) to produce a more accurate estimation of the full state $\boldsymbol{x}$ of SPIR from these raw sensor data.

Kalman filtering (Zarchan and Musoff, 2013) is a method that uses a dynamics model of a system, multiple sequential measurements from sensors, as well as known control inputs to that system to estimate varying states of the system, which is more accurate than the state estimate obtained from only measurements. The Kalman filter is a recursive estimator. It can run in real time, the state estimate at the current timestep can be computed through only the state estimate from the last timestep and the measurement at the current timestep.

The basic Kalman filter is based on linear dynamical systems. More complex systems, however, can be nonlinear. Thus EKF has been proposed, it assumes the true state at

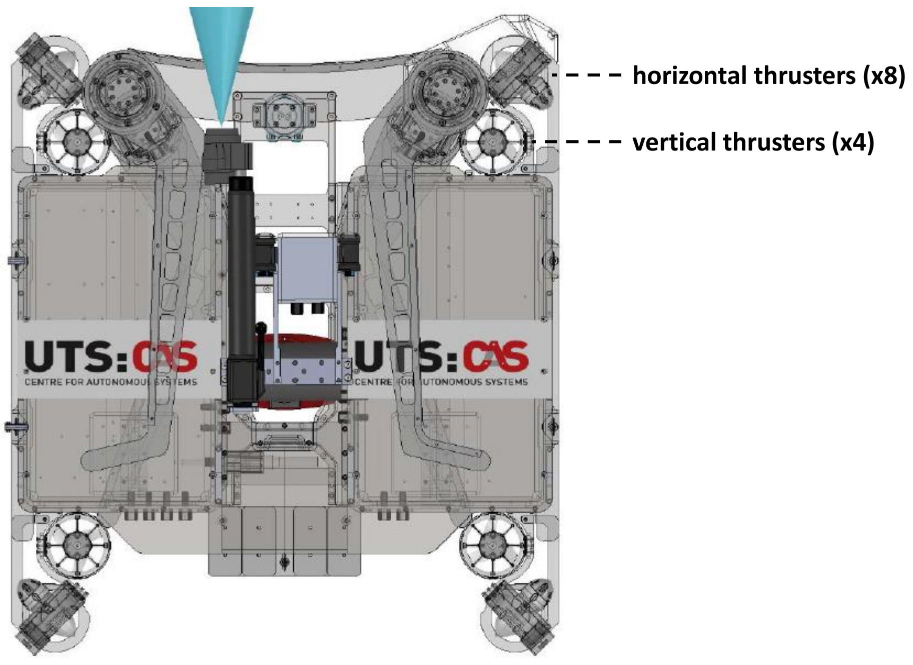

horizontal thrusters (x8)
vertical thrusters (x4)
UTS:CAS
CENTRE FOR AUTONOMOUS SYSTEMS
UTS:CAS
CENTRE FOR AUTONOMOUS SYSTEMS

FIGURE 5.6: Thruster layout of the SPIR.

time k is evolved from the state at $k-1$ according to:

$$\boldsymbol{x}_k = f\left(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k\right) + \boldsymbol{w}_k, \tag{5.6}$$

where the function f is the state transition model which is applied to compute the predicted state $\boldsymbol{x}_k$ from the previous state estimate $\boldsymbol{x}_{k-1}$ and the control vector $\boldsymbol{u}_k$, $\boldsymbol{w}_k$ is the process noise which is assumed to be drawn from a zero mean multivariate normal distribution $\mathcal{N}$ with covariance $\boldsymbol{Q}_k$: $\boldsymbol{w}_k \sim \mathcal{N}\left(\mathbf{0}, \boldsymbol{Q}_k\right)$. At time k an observation (or measurement) $\boldsymbol{z}_k$ of the true state $\boldsymbol{x}_k$ is made according to:

$$\boldsymbol{z}_k = h\left(\boldsymbol{x}_k\right) + \boldsymbol{v}_k, \tag{5.7}$$

where the function h is the observation model which maps the true state space into the observed space and $\boldsymbol{v}_k$ is the observation noise which is assumed to be zero mean Gaussian white noise with covariance $\boldsymbol{R}_k$: $\boldsymbol{v}_k \sim \mathcal{N}\left(\mathbf{0}, \boldsymbol{R}_k\right)$.

The system state $\boldsymbol{x}$ consists of the pose and velocity of the robot expressed in the global frame defined by the markers, $\boldsymbol{x} = \left[\boldsymbol{p}_B^{G^T}\ \boldsymbol{e}_B^{G^T}\ \boldsymbol{v}_B^{G^T}\ \boldsymbol{\omega}_B^{G^T}\right]^T \in \mathbb{R}^{12}$. The control vector

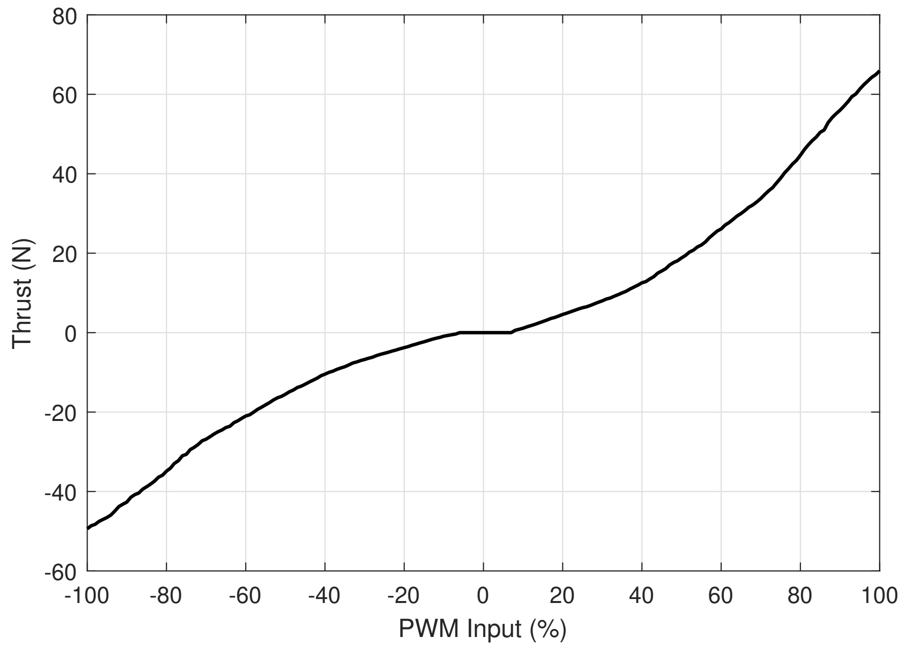


FIGURE 5.7: Thruster dynamics (mapping between PWM and thrust force) of Blue Robotics T200 Thruster.

$\boldsymbol{u}$ is chosen to be the linear acceleration and angular velocity in the IMU frame, $\boldsymbol{u} = \left[\dot{\boldsymbol{v}}_I^T \ \boldsymbol{\omega}_I^T\right]^T \in \mathbb{R}^6$. The observation $\boldsymbol{z}$ is chosen to be the pose of markers in the camera frame, $\boldsymbol{z} = \left[\boldsymbol{p}_G^{C^T} \ \boldsymbol{e}_G^{C^T}\right]^T \in \mathbb{R}^6$.

The state transition model f in Equation 5.6 is given as:

$$\begin{aligned}
\boldsymbol{p}^G_{B_k} &= \boldsymbol{p}^G_{B_{k-1}} + \boldsymbol{R}^G_B \boldsymbol{v}_{B_{k-1}} \Delta t, \\
\boldsymbol{e}^G_{B_k} &= \boldsymbol{e}^G_{B_{k-1}} + \boldsymbol{J}^{-1}_{ko}\left(\boldsymbol{e}^G_{B_{k-1}}\right) \boldsymbol{\omega}_{B_{k-1}} \Delta t, \\
\boldsymbol{v}_{B_k} &= \boldsymbol{v}_{B_{k-1}} + \left(\boldsymbol{R}^B_I \dot{\boldsymbol{v}}_{I_k} - \boldsymbol{R}^{G^T}_B \boldsymbol{g}^G\right) \Delta t, \\
\boldsymbol{\omega}_{B_k} &= \boldsymbol{R}^B_I \boldsymbol{\omega}_{I_k},
\end{aligned} \tag{5.8}$$

where $\boldsymbol{R}^G_B$ is the rotation matrix expressing the transformation from the global frame to the robot body frame, $\boldsymbol{J}_{ko}$ is the Jacobian to transform derivative of the Euler angles to robot body-fixed angular velocity, $\boldsymbol{R}^B_I$ is a constant rotation matrix from the IMU frame to the robot body frame, $\boldsymbol{g}^G$ is the acceleration of gravity expressed in the global frame.

The observation model h in Equation 5.7 is given as:

$$\begin{aligned} \boldsymbol{p}_G^C &= \boldsymbol{R}_C^{B^T}\left(-\boldsymbol{R}_B^{G^T}\boldsymbol{p}_B^G\right) - \boldsymbol{R}_C^{B^T}\boldsymbol{p}_C^B, \\ \boldsymbol{R}_G^C &= \boldsymbol{R}_C^{B^T}\boldsymbol{R}_B^{G^T}, \end{aligned} \tag{5.9}$$

where $\boldsymbol{p}_C^B$ and $\boldsymbol{R}_C^B$ are the constant translation and rotation transformation from the camera frame to the robot body frame.

During the application of Kalman filtering, typically the prediction and update phases alternate. That is, the prediction phase continuously advances the system state until the next observation occurs, and then the update phase incorporates the observation. However, this is not necessary. In this work, the control vector $\boldsymbol{u}$ coming from the IMU has a much higher frequency compare with the observed pose $\boldsymbol{z}$ calculated by the visual localization algorithm. Thus, the update may be skipped and multiple prediction steps performed.

5.4.3 Dynamics Model Learning

As discussed in Section 3.2, we know that directly learning a transition function that outputs the next state $\boldsymbol{x}_{t+1}$ might be difficult, since two consecutive states $\boldsymbol{x}_t$ and $\boldsymbol{x}_{t+1}$ can be too similar, and state difference cannot well indicate the underlying dynamics. Thus, we instead learn a dynamics model that predicts state change over one timestep Δt, the predicted next state is given as $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + f_{real}\left(\boldsymbol{x}_t, \boldsymbol{u}_t; \theta_f\right)$.

We divided the recorded motion data into training dataset $\mathcal{D}_{train}$ and validation dataset $\mathcal{D}_{val}$, where the data is further sliced into inputs $(\boldsymbol{x}_t, \boldsymbol{u}_t)$ and labels $\boldsymbol{x}_{t+1} - \boldsymbol{x}_t$. We then conduct feature normalization on both inputs and labels by subtracting the mean of the data and dividing by the standard deviation of the data, to ensure the loss function weights the different parts equally. Zero mean Gaussian noise is also added to the training data to increase model robustness. After data preprocessing, we train the dynamics model $f_{real}\left(\boldsymbol{x}_t, \boldsymbol{u}_t; \theta_f\right)$ offline using supervised learning by minimizing the error:

$$\mathcal{E}(\theta_f) = \frac{1}{|\mathcal{D}_{train}|} \sum_{(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{x}_{t+1}) \in \mathcal{D}_{train}} \left\|(\boldsymbol{x}_{t+1} - \boldsymbol{x}_t) - f_{real}\left(\boldsymbol{x}_t, \boldsymbol{u}_t; \theta_f\right)\right\|^2, \tag{5.10}$$

using SGD. While training on the training dataset $\mathcal{D}_{train}$, we also calculate the mean squared error in Equation 5.10 on the validation dataset $\mathcal{D}_{val}$ to optimize hyperparameters.

5.5 Simulation

The simulation setup is the same as Chapter 4. Only the horizontal motion and control (X, Y, yaw) of the robot are considered. The performance of the designed algorithms is tested on a pose regulation task. The simulated disturbances used in the source task are constructed as a superposition of multiple sinusoidal waves with different amplitudes, frequencies and phases, as shown in Table 3.1 and Figure 4.2.

5.5.1 Results of Model-Based Policy Optimization

For the task of pose regulation, we first evaluated iLQR, model-based policy optimization and model-free RL in the source task, including the effects of disturbances (Figure 5.8 (a)). Note that for these tasks, the robot dynamics and disturbances are explicitly constructed using mathematical formulations, and there is no difference between the internal model

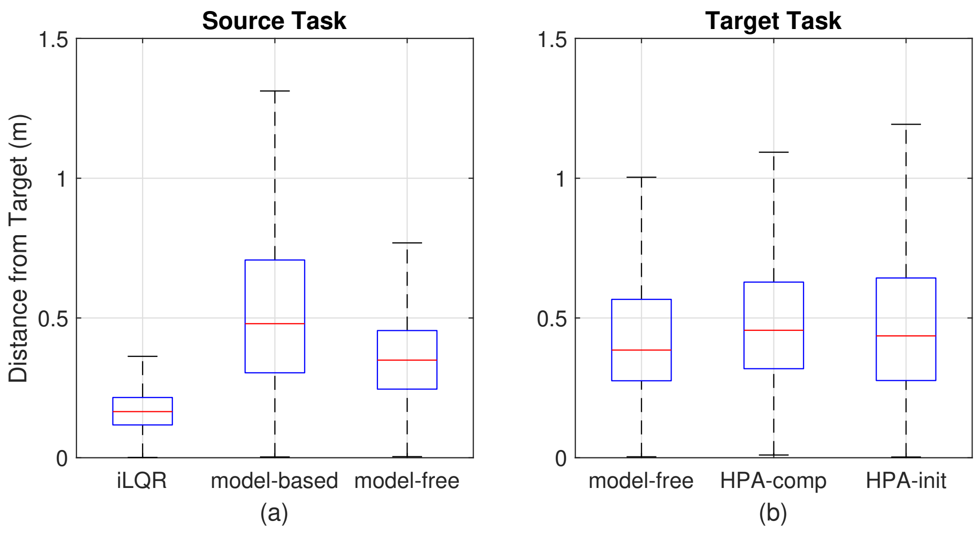


Figure 5.8: Distributions of the distance between the robot and the target during last 100 timesteps: (a) comparison among iLQR, model-based policy and model-free policy, all run in the source task with the simulated disturbances; (b) comparison among purely model-free policy, HPA-comp policy and HPA-init policy in the target task.

used by the model-based controller and the external test environment. The iLQR solutions act as an upper bound of control performance for the learned policies, since they are optimized given the full knowledge of the disturbances in advance. These results show that, even with the use of such a naive supervised-learning-based policy optimization, the learned model-based policy is powerful enough to well stabilize the robot at test time, its performance is comparable to the model-free RL that learns from scratch. Although there is still a gap between the trained policies (both model-based one and model-free one) and the optimal trajectories provided by iLQR, the near-optimal performance for the model-based policy can be regarded as satisfactory, and is sufficient to be a start point for the policy adaptation in the target task.

5.5.2 Results of Hybrid Policy Adaptation

We now compare the purely model-free RL algorithm with HPA algorithm. When the learning algorithms are deployed on the empirical model without disturbances, there is no need to include history data into the policy input, then HPA is implemented only by using the model-based policy as an initialization for the model-free adaptation. In this way, HPA is apparently better than the purely model-free learner. As shown in Figure 5.9 (a), HPA learner starts at a higher initial cumulative reward (-300 vs. -1400), and is nearly 4 times faster (1000 episodes vs. 4000 episodes) to converge, compared with the model-free learner. This illustrates that HPA does avoid risky exploratory actions in the beginning, ensuring safer sampling and motion.

When we take the disturbances into consideration, the history data and thus the recurrent formulation are required for the optimal control performance. We found in the Figure 5.8 (b) that, both the compensation and initialization approaches of HPA (HPA-comp and HPA-init) can achieve similar performance with the model-free RL. However, during training, HPA algorithms still outperform the purely model-free RL. As shown in Figure 5.9 (b), both HPA-comp and HPA-init algorithms start with a higher cumulative reward (-1100/-1700 vs. -2000) and achieve a higher convergence speed (2000 episodes/3000 episodes vs. 5000 episodes), compared with learning from scratch.

Also, it can be seen that using compensatory term helps, but is not that effective. While the initialization approach dramatically improves the training performance. The reason for this phenomenon might be that, in these two approaches, the model-based policy optimization was conducted on different dynamics model, either including the disturbances

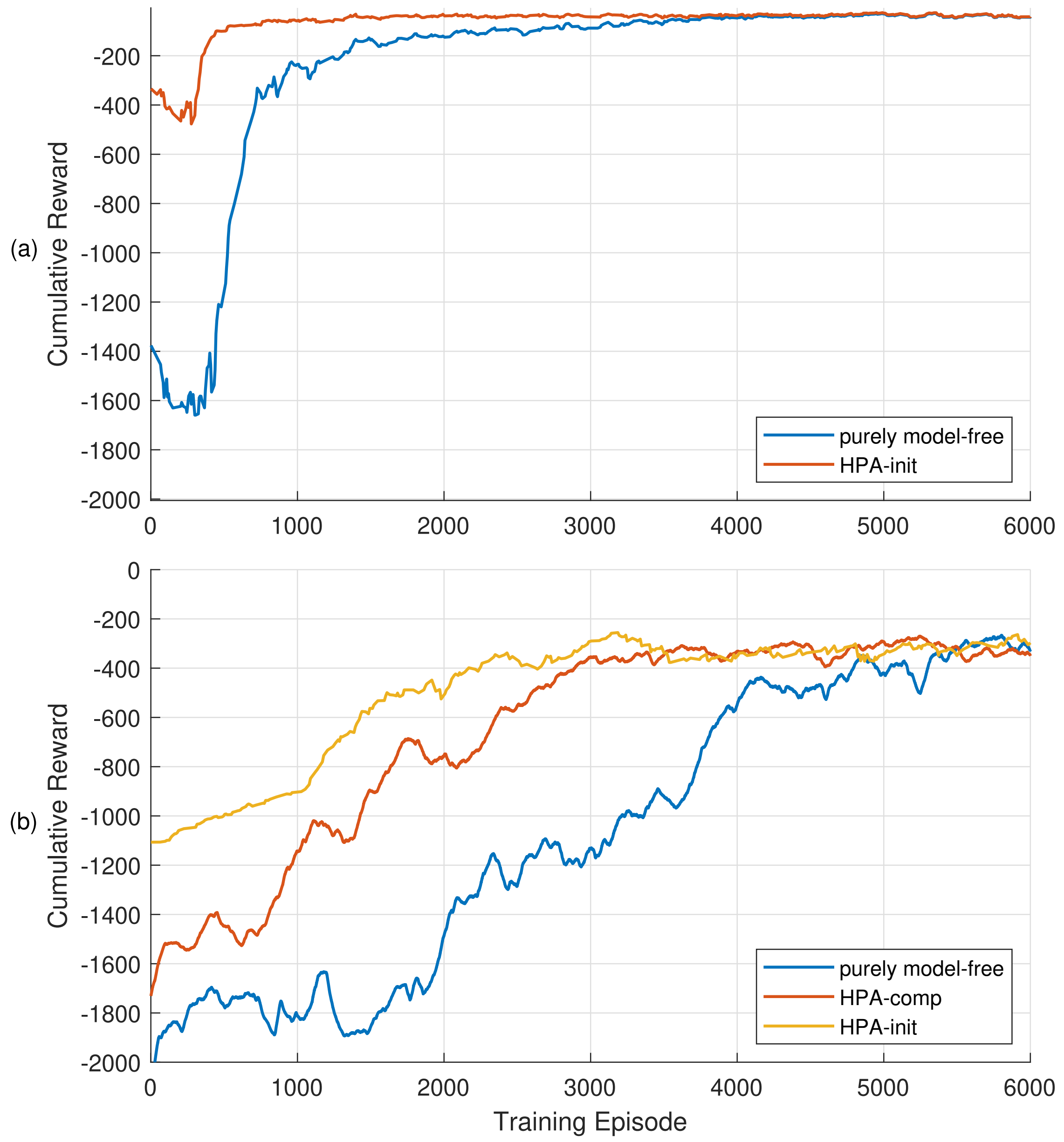


FIGURE 5.9: Training rewards of the purely model-free RL and HPA: (a) training in the target task without disturbances, both model-based policy and model-free policy are constructed using feedforward network, the model-based policy are only used as the initialization for the model-free adaptation; (b) training in the target task with disturbances, two different algorithms for HPA are compared, either using the simulated disturbances in the source task or not.

or not. Figure 5.10 shows the state-action distribution on X-axis during the task execution. We can see that the model-based policy only has relatively small control outputs when being trained on the mathematical model without disturbances. While HPA policy that is trained on the empirical model including the real disturbances, tends to produce the control signals that are mainly distributed close to the control constraints $\overline{\boldsymbol{u}}$ and $\underline{\boldsymbol{u}}$, and are distributed uniformly over the whole action space in an interval near the target position (zero). This kind of state-action distribution is caused by the strong impact of the excessive disturbances. Thus, the model-based policy is quite different with the hybrid one, then cannot provide much help for the adaptation process on the empirical model during transfer learning.

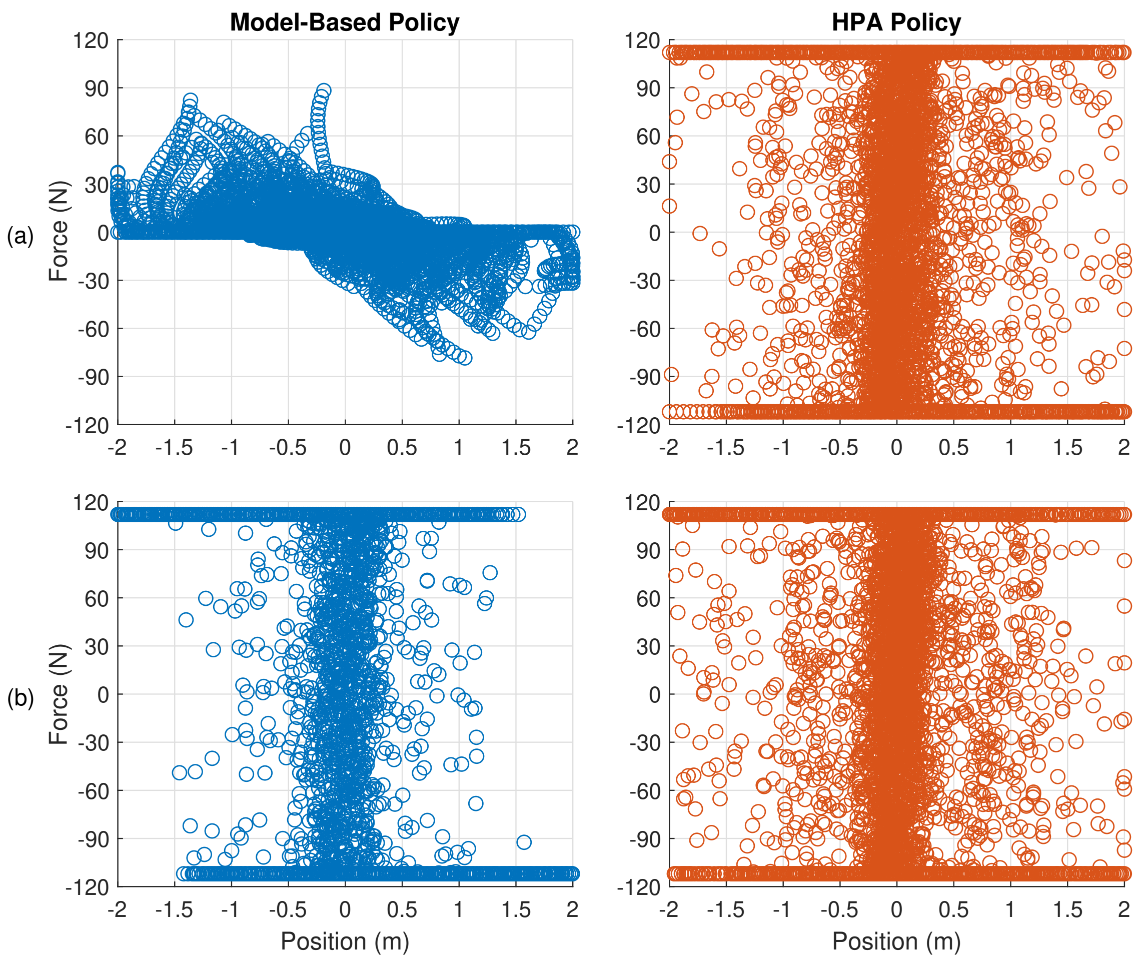


FIGURE 5.10: State-action distributions in X-axis of the model-based policy and HPA policy: (a) learning model-free compensatory policy; (b) model-based initialization of model-free RL.

For another transfer algorithm using model-based initialization of model-free RL, this situation is mitigated. Because the model-based policy in this case is trained on the mathematical model given the simulated disurbances, thus the pre-trained policy already possesses the capability to handle excessive disturbances using history data. The only difference needs to be adapted during transfer is the mismatch of robot dynamics and disturbances between the mathematical model and the empirical model, which is obviously an easier process compared to the previous one. From Figure 5.10 (b), we can see that there is only moderate difference between the two distributions.

Chapter 6

Transferring Disturbance Rejection Policy with Transition Mismatch Compensation

In Chapter 5, the two sources of model mismatch in external disturbances and system dynamics are combined together and treated equally by the transfer learning algorithm. While in this chapter, the external disturbances are separated from the system dynamics and processed independently, through using ODI to predict a set of disturbance waveforms that best fit the real ones in the target task. The residual mismatch in the external disturbances, which can be sufficiently small (as shown in Section 6.3), is then merged into the original mismatch of the system dynamics to be adapted together by the transfer learning algorithm. Such framework is expected to reduce the amount of total model mismatch that the transfer leaner needs to adapt, thus improves the learning efficiency during transfer.

This chapter discusses three transfer learning algorithms, CCL, TML-control and TML-feature, based on the modular architecture of GCP-ODI proposed in Chapter 4, all of them follow a basic idea of training an additional policy to compensate the dynamics model mismatch between the source and target tasks.

6.1 Compensatory Control Learning

Compensatory Control Learning (CCL) algorithm (Figure 6.1) learns an additional control policy, which generates a compensatory control action $\boldsymbol{u}_t^c$ directly added to the control input computed by the base policy $\hat{\boldsymbol{u}}_t$, the combined control action $\boldsymbol{u}_t$ is then applied to the empirical model in the target task. The training process uses the task reward function Equation 3.9, which establishes similar optimization goals for both the base policy and the compensatory policy. In order to ensure that ODI remains effective, the action fed into ODI should be the output of the base policy $\hat{\boldsymbol{u}}_t$, that is what ODI has seen during training. However, through transfer learning using Equation 3.9, CCL algorithm could not optimize the compensatory policy to a satisfactory performance. This is because, although ODI is fed with the action of the base policy, the whole trajectory data still comes from the combined control policy and the empirical model in the target task. In contrast, ODI is trained using the trajectory data generated by the base policy under the mathematical model in the source task. Thus, the differently distributed trajectories fed to ODI lead to worse performance of the whole system.

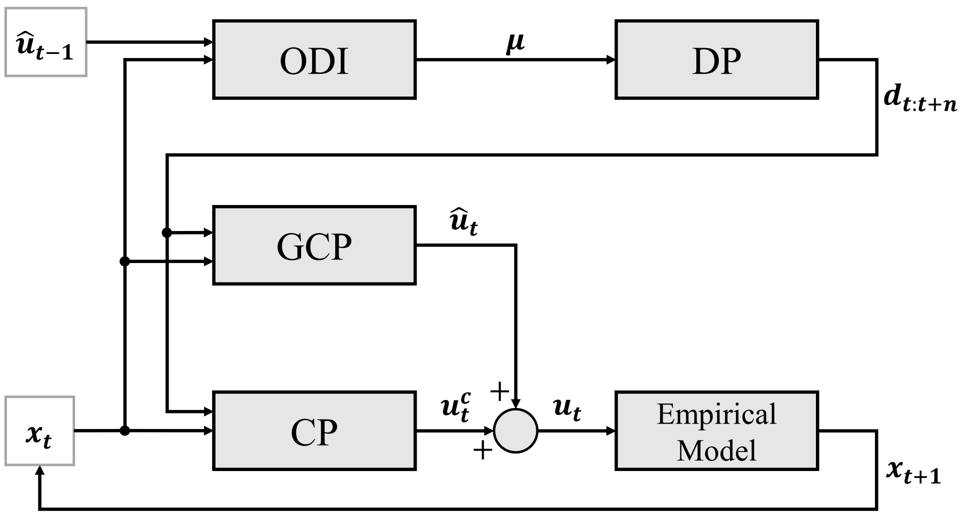


FIGURE 6.1: Diagram of compensatory control learning (CCL).

6.2 Transition Mismatch Learning

To address the issue of trajectory mismatch, Transition Mismatch Learning (TML) algorithm (Figure 6.2) learns a compensatory policy from the difference between transitions of the mathematical model and the empirical model, $\boldsymbol{e}_t = \boldsymbol{x}_t - \hat{\boldsymbol{x}}_t$. In this case, the compensatory policy uses a different optimization goal where the reward is given by

$$\begin{aligned} r_m\left(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{x}_{t+1}\right) &= r\left(\boldsymbol{x}_t, \boldsymbol{u}_t\right) - \left\|\boldsymbol{e}_{t+1}\right\|_2 \\ &= -\boldsymbol{x}_t^T \boldsymbol{Q} \boldsymbol{x}_t - \boldsymbol{u}_t^T \boldsymbol{R} \boldsymbol{u}_t - \left\|\boldsymbol{x}_{t+1} - \hat{\boldsymbol{x}}_{t+1}\right\|_2 \end{aligned}, \tag{6.1}$$

which does not divert the base policy from reaching its objective. The purpose of such optimization setting is to eliminate the transition mismatch $\boldsymbol{e}$ by forcing the empirical model to behave like the mathematical model as if there is no model mismatch. When the transition mismatch approaches zero $e_t \to 0$, the state trajectories of the empirical model will get similar with those of the mathematical model, making ODI work correctly by using the trajectory data matched with the training process. In the meantime, the outcome of the combined control policy will approach the outcome of the optimal policy with respect to the mathematical model. Also, under this setting, the well trained ODI is able to predict a set of waveforms that well describe the real disturbances from the

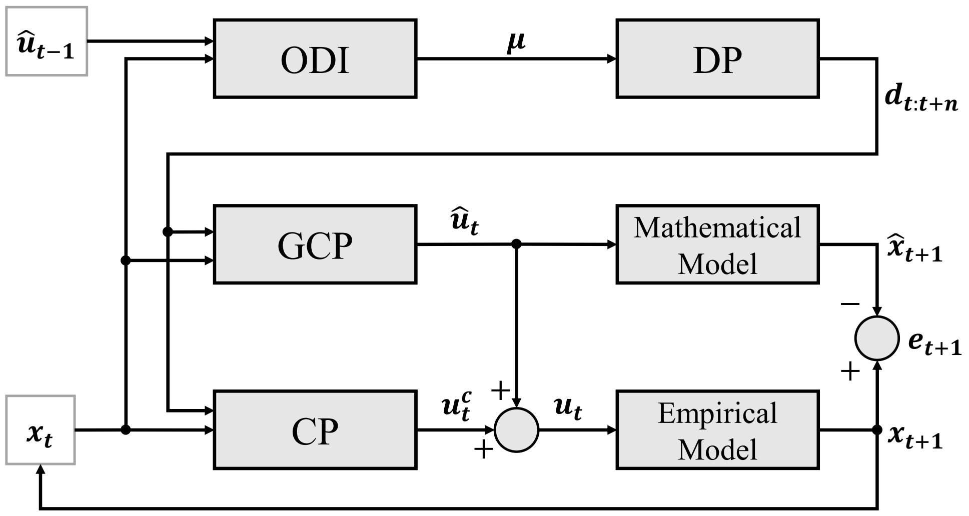


FIGURE 6.2: Diagram of transition mismatch learning with control combination (TML-control).

distributions of simulated parameters, thus the total model mismatch required to be compensated is minimized. However, these optimization effects are attainable only when the full power of the compensatory control is released. We found that, in order to have good performance against the excessive disturbances, the control actions of the base policy $\hat{\boldsymbol{u}}_t$ often reach or exceed the control constraints, leaving a little space for the compensation to directly take effect. Thus, the compensatory term needs to learn a very complicated function in order to optimize overall system performance under the constraints of robot control capabilities. That is why the transfer learning is difficult to achieve its theoretical optimization limit. In contrast, the compensatory policy learned in Chapter 5 is able to achieve similar performance with respect to learning from scratch on the empirical model directly, since the base policy in that case only has control outputs that much smaller than the control constraints, as shown in Figure 5.10 (a). Apparently, we need a more efficient way to apply the control compensation.

We then propose TML with feature combination (Figure 6.3) to further improve TML framework. Similar with TML-control, the optimization of TML-feature still applies the reward function Equation 6.1 for maximizing task performance as well as minimizing transition mismatch at the same time. The difference is that, rather than naively adding together the control actions, TML-feature integrates features before the last layer of the

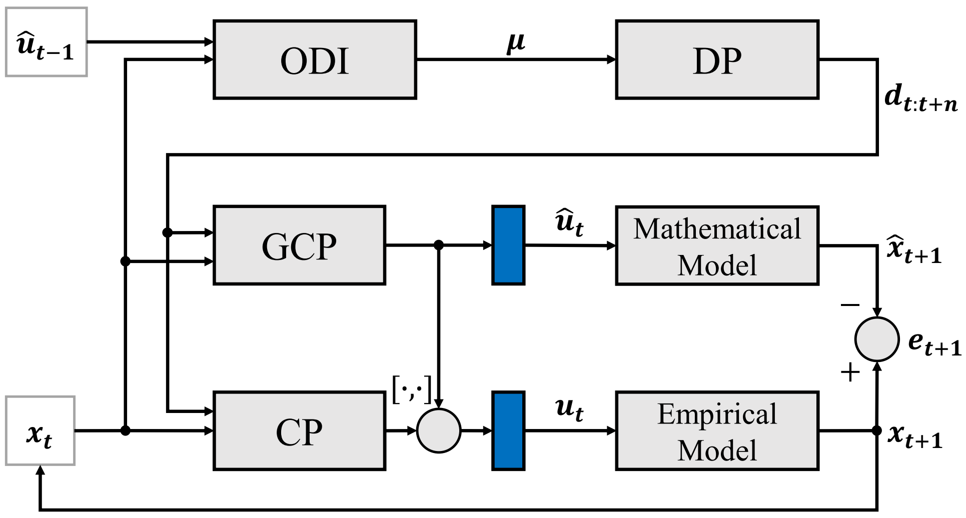


Figure 6.3: Diagram of transition mismatch learning with feature combination (TML-feature).

two policy networks, then feeds the integrated features into fully connected layers to produce expected control. Combining middle layer features might be an effective way to deal with control constraints, such approach can provide more flexibility and improve network capacity, since the middle layer features contain more comprehensive information compared with the final layer outputs with physical significance. Through the middle layer features integration, the restrict limitations in the control space might be eased in a “feature space”, then reducing the learning difficulty of the compensatory policy.

6.3 Simulation

The simulation setup is the same as Chapter 4. Only the horizontal motion and control (X, Y, yaw) of the robot are considered. The performance of the designed algorithms is tested on a pose regulation task of the underwater robot. The disturbances used in the simulation are constructed as a superposition of multiple sinusoidal waves with different amplitudes, frequencies and phases. In this work, we use a composition of five sinusoidal waves, as shown in Table 6.1. The transfer learning algorithms are tested on an empirical model including real wave forces. The empirical model is constructed using supervised learning based on the collected motion and control data of an underwater robot SPIR in a water tank, and the real wave data is gathered in open water using a force/torque sensor and an unactuated robot.

In the remaining part of this section, we first evaluate the effects of different model variations on the transfer learning, then test and compare the control performance among different transfer learning algorithms on the empirical model.

Table 6.1: Distributions of disturbance parameters used in the source task.

Component Wave	**1**	**2**	**3**	**4**	**5**
Amplitude w.r.t. $\lvert\overline{\boldsymbol{u}}\rvert$	$0 \sim 50\%$	$50 \sim 100\%$	$50 \sim 100\%$	$50 \sim 100\%$	$0 \sim 50\%$
Period (s)	$1 \sim 2$	$2 \sim 4$	$2 \sim 4$	$2 \sim 4$	$4 \sim 8$
Phase (rad)	$-\pi \sim \pi$	$-\pi \sim \pi$	$-\pi \sim \pi$	$-\pi \sim \pi$	$-\pi \sim \pi$

6.3.1 Evaluation on Dynamics Model Uncertainties

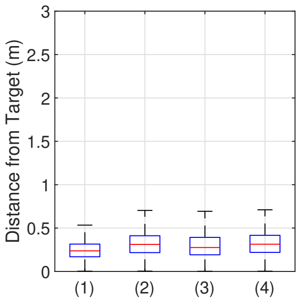
We already validate that GCP-ODI performs well on a mathematical model, as described in Equation 5.1. But there are still many uncertainties in the actual system dynamics. Before deploying the transfer learning on the empirical model, we first evaluate the influence of different sources of model uncertainties.

In this part of evaluation, both the source and target tasks are defined as mathematical models, which are called original model and model variation, respectively. Possible uncertainties in the dynamics model for the underwater robot may include:

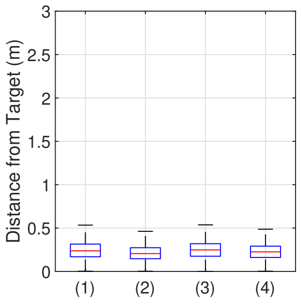
- Mass
- Geometry
- Hydrodynamics
- Velocity Constraints
- Control Constraints
- Control Offset
- Control Latency

TABLE 6.2: Variations in the mathematical model.

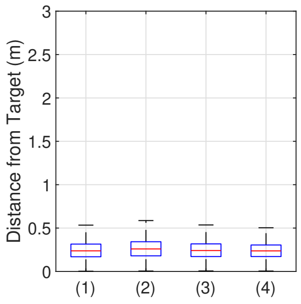
Parameters	**Original Model**	**Model Variation**
Mass	$60kg$	$50kg$
	$60kg$	$70kg$
Geometry	Cuboid Geometry: $0.68 \times 0.75 \times 0.19m^3$	CAD Geometry
Hydrodynamics	None	Added Mass and Inertia Damping Effects
Velocity Constraints	$\pm\left[1.0m/s\ 1.0m/s\ \frac{\pi}{2}rad/s\right]$	$\pm\left[0.7m/s\ 0.7m/s\ \frac{\pi}{3}rad/s\right]$
	$\pm\left[0.7m/s\ 0.7m/s\ \frac{\pi}{3}rad/s\right]$	$\pm\left[1.0m/s\ 1.0m/s\ \frac{\pi}{2}rad/s\right]$
Control Constraints	$\pm[112N\ 112N\ 82Nm]$	$\pm[86N\ 86N\ 62Nm]$
	$\pm[86N\ 86N\ 62Nm]$	$\pm[112N\ 112N\ 82Nm]$
Control Offset	None	30% of Control Constraints $\lvert\overline{\boldsymbol{u}}\rvert$
Control Latency	None	$100ms$



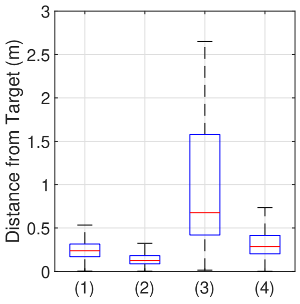
(a) Smaller Mass



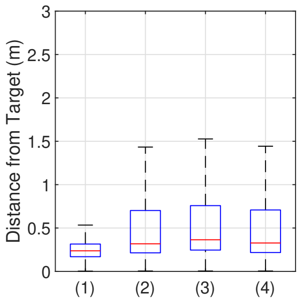
(b) Larger Mass



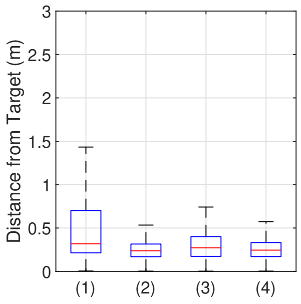
(c) Geometry



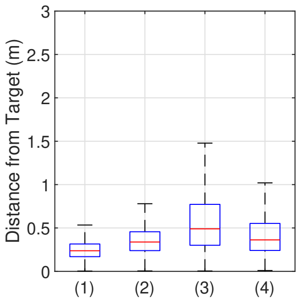
(d) Hydrodynamics



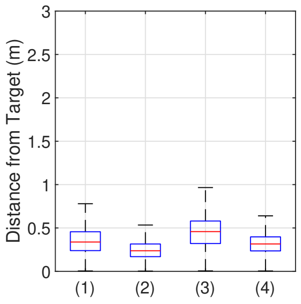
(e) Smaller Velocity Constraints



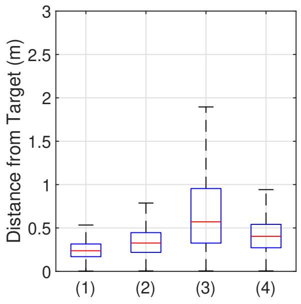
(f) Larger Velocity Constraints



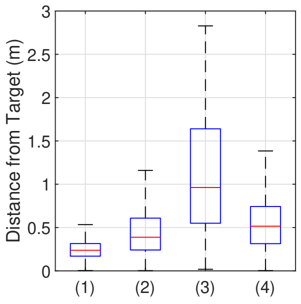
(g) Smaller Control Constraints



(h) Larger Control Constraints



(i) Control Offset



(j) Control Latency

FIGURE 6.4: Distribution of the distance between the robot and the target during last 100 timesteps for different model variations: (1) GCP-ODI trained and tested on the original model (i.e., source policy); (2) GCP-ODI trained and tested on the model variation (i.e., target policy); (3) GCP-ODI trained on the original model and tested on the model variation (i.e., unadapted policy); (4) transfer learning using TML-feature (i.e., adapted policy).

Table 6.2 summarizes different model parameters and corresponding variations in the mathematical model. Figure 6.4 gives the detailed evaluation results of each type of model variation. We can see that variations of mass and geometry have little influence on the control performance when using the unadapted policy compared with using the target policy on the model variation, then there is no need to spend much effort on estimating precise mass and geometry of the real system when designing a source task (a mathematical model) for transfer. For the variations of velocity constraints, we found that the target policy, the unadapted policy, and the adapted policy on the model variation have similar performance. That is to say, the control performance depends mostly on the dynamics model itself, rather than the applied algorithms. Thus, it is not necessary to apply the transfer learning under this kind of model uncertainties.

The theory of hydrodynamics is rather complex and it is difficult to develop a reliable model for most of the hydrodynamic effects. Thus we do not make any assumption of the hydrodynamics in the original model. This variation in the dynamics model brings a great impact on the control stability when using the unadapted policy. Another important source of uncertainties is control latency, which widely exists in most of mechanical and electronic systems. But just like the hydrodynamics, the control latency is also hard to be well simulated, so we assume no latency in the original model. The nature of latency might complicate the motion data used in ODI, leading to adverse effects on the control performance for the unadapted policy. The application of the transfer learning is necessary when these two model uncertainties exist, which is normally the case for the underwater robots.

The control constraints of the real system may have some scaling or offset compared to the mathematically modeled ones. This phenomenon can be caused by the uncertainties in thrusters' dynamics. Also, the robot may be subjected to some unexpected external forces, like the pulling force of the power cable for most of the ROV. The variation of the control constraints and the occurrence of the control offset have some influence for the unadapted policy, in which case the transfer learning is also required. But this influence is only moderate since most of the control signals are at the bound values $\overline{\boldsymbol{u}}$ and $\underline{\boldsymbol{u}}$ (see Figure 6.7), minor variations will not cause a great impact.

6.3.2 Results of Transfer Learning on Empirical Model

We have already constructed an empirical model using neural network to represent robot dynamics in still water and real wave forces collected in open water, then we focus on the transfer learning of a control policy trained on the mathematical model to be successfully deployed in the empirical model. Note that we provide the results of GCP-ODI trained directly on the empirical model, these results are used as optimal performance in the target task, and can be considered as an upper bound for the transfer learning algorithms. This policy is available since the empirical model including the real disturbances can be deployed in simulation, but these results are difficult to obtain on a real robot due to sample complexity and damaging exploratory policy.

Figure 6.5 shows the test results of different algorithms, we can see that GCP-ODI has poor stability when trained on the mathematical model then directly deployed on the empirical model. In contrast, GCP-ODI demonstrates much better performance when

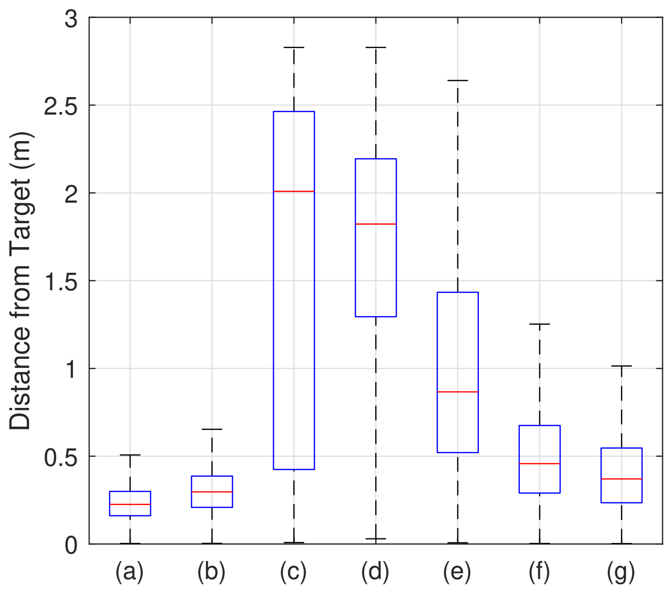


FIGURE 6.5: Distribution of the distance between the robot and the target during last 100 timesteps for the transfer learning: (a) GCP-ODI trained and tested on the mathematical model; (b) GCP-ODI trained and tested on the empirical model; (c) GCP-ODI trained on the mathematical model and tested on the empirical model; (d) transfer learning using CCL; (e) transfer learning using TML-control; (f) transfer learning using TML-feature; (g) transfer learning using HPA-init.

directly trained on the empirical model. That is the reason why we need the transfer learning between these two dynamics models.

The training process of the transfer learning can be visualized from Figure 6.6. We evaluate two objectives, which are cumulative task reward $\mathcal{R} = \sum_{t=1}^{T} r(\boldsymbol{x}_t, \boldsymbol{u}_t)$ and cumulative transition mismatch $\mathcal{E} = \sum_{t=1}^{T} \|\boldsymbol{e}_t\|_2$ in each training episode. It is obvious that the transfer learning algorithms converge faster than GCP using the true disturbance parameters

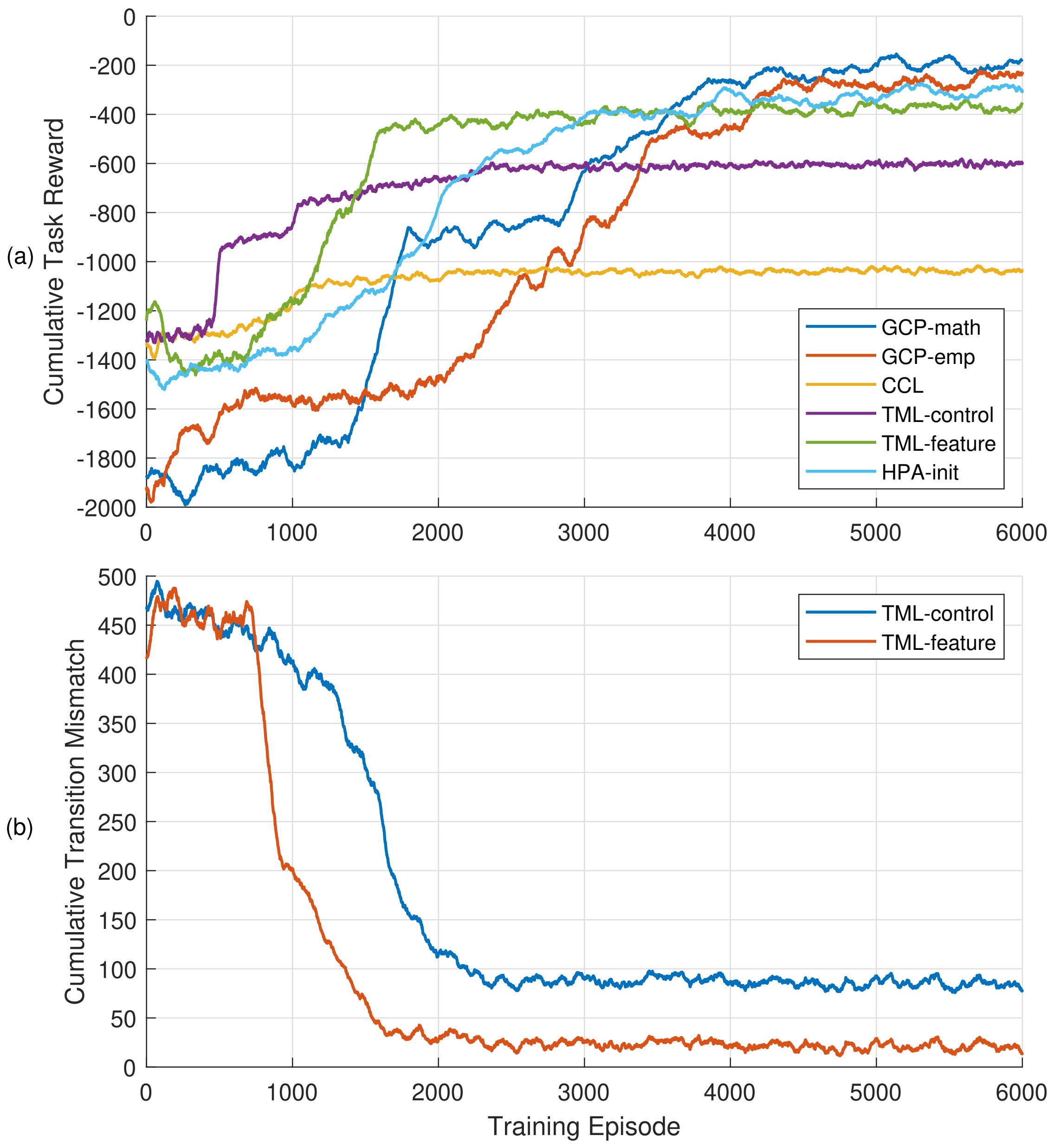


FIGURE 6.6: Training process of the transfer learning among different algorithms: (a) cumulative task reward; (b) cumulative transition mismatch.

$\bar{\boldsymbol{\mu}}$ on either the mathematical model or the empirical model (GCP-math and GCP-emp), not to mention GCP is also followed by the iteratively training of ODI. Thus, the three algorithms of transfer learning all improve the sample efficiency, compared with training GCP-ODI from scratch without transfer. In addition, it is reasonable that the training reward of TML is worse than GCP-emp after convergence. This is because GCP-emp directly uses the true disturbance parameters under the empirical model, thus provides optimal performance under ideal conditions. But such performance is almost impossible to achieve in real situations, since the errors in predicting disturbance parameters for ODI is difficult to be completely eliminated.

Among the three transfer learning algorithms, CCL cannot successfully stabilize the robot, since ODI uses inconsistent trajectory data during transfer and training. TML-control achieves much better performance due to the introduction of a parallel mathematical model, so that the transition mismatch $\boldsymbol{e}$ can be minimized. But this result still cannot be considered as a satisfactory solution. We then look at TML-feature, which gives a more effective combination approach of the compensatory control signals $\boldsymbol{u}^c$ and the original

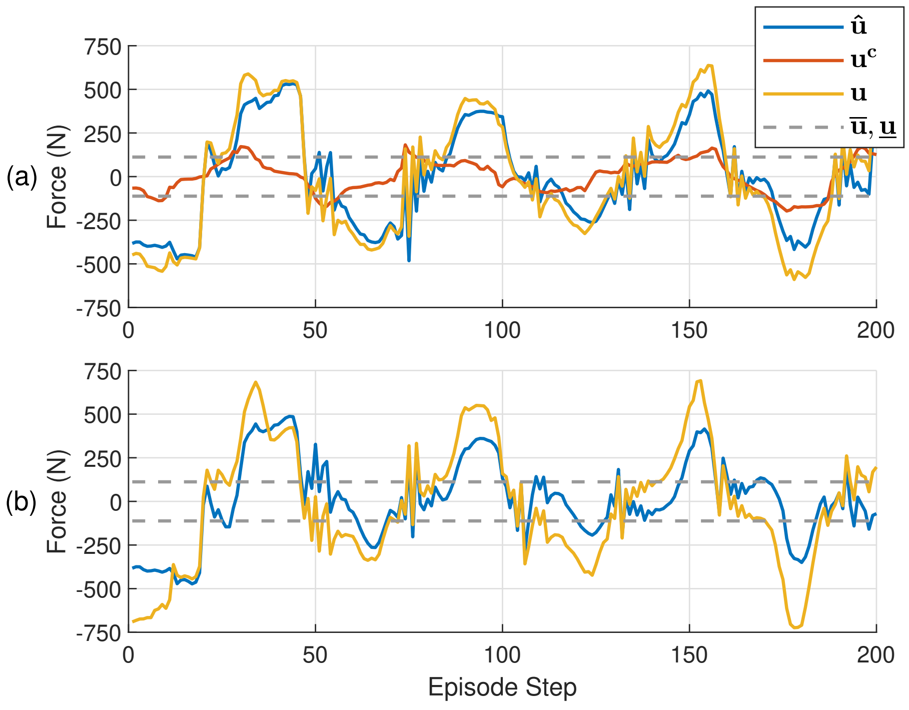


FIGURE 6.7: Comparison among different control signals (X-axis) for the robot during the transfer learning on the empirical model: (a) transfer learning using TML-control; (b) transfer learning using TML-feature.

ones $\hat{\boldsymbol{u}}$, by combining middle layer features of the policy networks. Both Figure 6.6 (a) and Figure 6.5 prove that TML-feature achieves better control performance than TML-control. In addition, through further evaluating the different control components during transfer (see Figure 6.7), we found that in TML-control algorithm, the compensation $\boldsymbol{u}^c$ is relatively small compared to GCP output $\hat{\boldsymbol{u}}$, then adding $\boldsymbol{u}^c$ to $\hat{\boldsymbol{u}}$ will not make much difference. Also, most of the added control actions have been truncated by the control constraints $\overline{\boldsymbol{u}}$ and $\underline{\boldsymbol{u}}$, leading to limited effects of the compensatory control actions. While for TML-feature algorithm, combining features distinguishes the resultant control actions $\boldsymbol{u}$ from the output of GCP $\hat{\boldsymbol{u}}$, there is even obvious phase advance in $\boldsymbol{u}$ that compensates the control latency. Figure 6.8 shows the trajectories of the robot when using TML-feature algorithm and the direct deployment of GCP-ODI on the empirical model after trained on the mathematical model. There is a significant improvement in the control stability of the robot after using the transfer learning, the converged region can be reduced to only $0.4m$.

However, even though the transfer learning using TML-feature is able to well stabilize the robot, there is still noticeable gap from GCP-ODI directly trained on the empirical model (see Figure 6.5). This phenomenon can be explained through Figure 6.6 (b), where

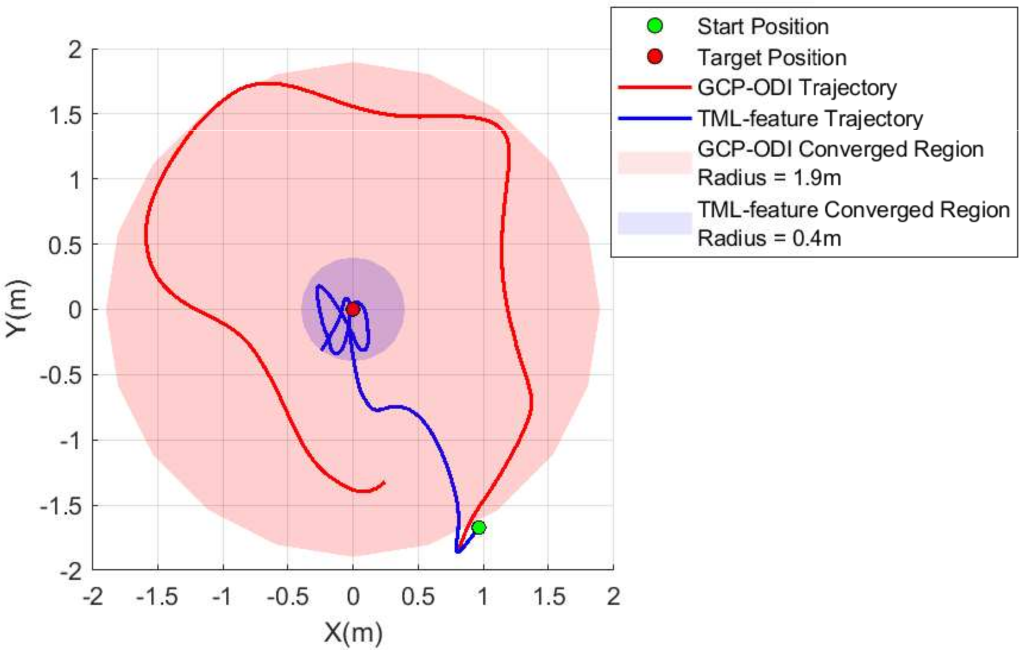


Figure 6.8: Trajectories of the underwater robot using GCP-ODI and TML-feature under the empirical model.

TABLE 6.3: Predicted disturbance parameters by ODI.

Component Wave	**1**	**2**	**3**	**4**	**5**
Amplitude w.r.t. $\lvert\overline{\boldsymbol{u}}\rvert$	34.06%	66.63%	84.59%	53.93%	37.93%
Period (s)	5.98	3.01	2.73	2.31	1.88
Phase (rad)	-0.22π	0.40π	-0.65π	-0.94π	-0.74π

the cumulative mismatch of TML-feature cannot be fully eliminated. This result might be caused by the existence of control constraints, which limits the algorithm's ability to actively reject disturbances and compensate model mismatch in the meantime.

In addition, during deployment on the empirical model, ODI is able to predict a set of disturbance parameters $\hat{\boldsymbol{\mu}}$ that best fit the real disturbances, which are not known to the learning algorithm and do not follow the distributions of simulated disturbance parameters. We give a waveform calculated from the disturbance parameters predicted at a timestep in the middle of a trajectory, as shown in Table 6.3, and compare it with the real disturbances exerted on the empirical model during test. We can see from Figure 6.9 that the predicted disturbances are quite similar to the real ones, then there is only a small mismatch between these two disturbance waveforms. Thus, the total model mismatch that the transfer learning is required to compensate is minimized, reducing the burden for the compensation.

Thanks to the ability of ODI to predict the unknown disturbances on the empirical model,

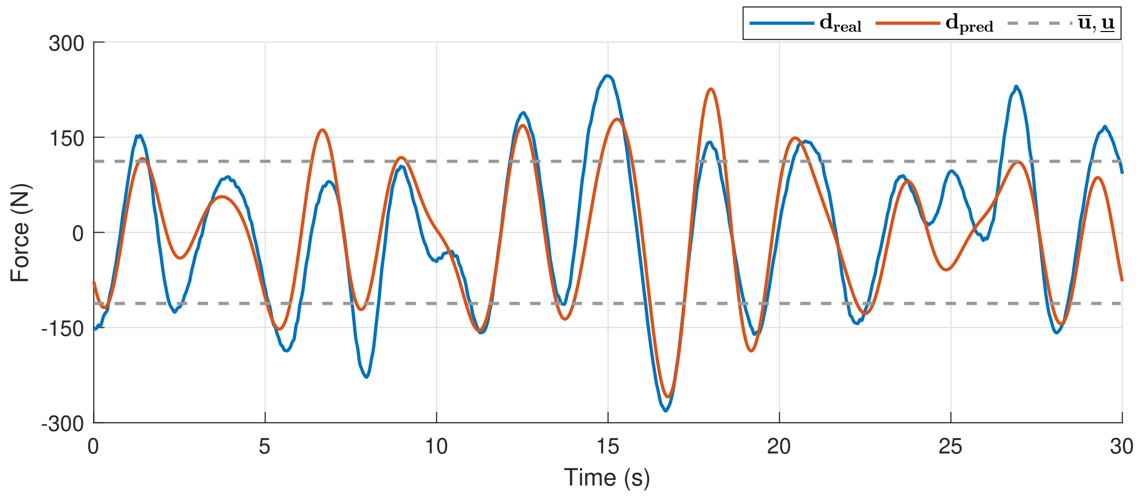


FIGURE 6.9: Comparison between the real disturbances and the predicted disturbances by ODI in Y direction, during transfer learning on the empirical model.

the total model mismatch is minimized and the transfer learning is easier to train, compared with HPA algorithm presented in Chapter 5. As illustrated in Figure 6.6, the convergence speed of TML-feature algorithm is clearly higher than that of HPA-init algorithm. The final performance of TML-feature, however, is not as good as the HPA policy (see Figure 6.5), this is reasonable since the transition mismatch $\boldsymbol{e}$ is difficult to be minimized to zero under current algorithm design. Further improvements could be investigated on a more optimized approach to combine compensatory signals.

Chapter 7

Conclusion

Underwater robots in shallow water environments usually suffer from turbulent flows and strong waves. Such environmental disturbances may frequently exceed the control constraints of robotic system, thus severely destabilize the robot during task operation. Learning-based controllers use neural networks to construct the disturbance estimation and attenuation system, simultaneously alleviating model dependency and achieving high computational efficiency.

Besides, due to the damaging exploratory actions on physical systems and the unpredictable variations in system dynamics during operation, policies often cannot be directly trained on desired dynamics model. Then, optimizing a policy based on an initial guess of the desired model, and transferring this policy to the desired model become promising approaches and may achieve improved sample efficiency.

In this thesis, reinforcement learning algorithms are applied to address the control problem of underwater robots under unobservable excessive time-correlated disturbances, and transfer learning algorithms are implemented for control policy adaptation under dynamics model mismatch.

7.1 Summary of Contributions

7.1.1 Learning Optimal Control under Excessive Time-Correlated Disturbances

We have developed learning algorithms for control under excessive time-correlated disturbances. The first idea is to jointly construct and optimize the disturbance observer and the motion controller in an end-to-end mode, with the unobservable disturbance waveforms implicitly encoded in the motion history of the robot and learned within the policy optimization. Two algorithms are presented and compared, one is developed based on history window formulation, called History Window Reinforcement Learning (HWRL). However, due to the difficulty in determining optimal history length through the history window formulation, RNN is then utilized to automatically learn how much past experience should be explored to achieve optimal performance. Then we have another algorithm called Disturbance Observer Network (DOB-Net).

The joint learning algorithms might achieve an optimized information transmission between observer and controller, compared with traditional hand-designed features. But in most cases, training such a large network leads to high sample complexity and difficulty of convergence to optimum. Then we seek a modular design and learning framework for the disturbance rejection control, called GCP-ODI. The idea is to precompute many of the possible disturbance waveforms the robot might encounter during operation, and learn a Generalized Control Policy (GCP) for each set of waveforms in a parameterized space. Then, an Online Disturbance Identification Model (ODI) is used to predict which waveforms best fit the observed motion history. During online execution, the disturbance parameters are first predicted by the learned ODI given the motion history of the robot, and the corresponding control policy is then selected for those disturbance waveforms to achieve stability control. We believe that decoupling a large network trained by reinforcement learning into two moderate-sized networks, might reduce the learning time and improve the sample efficiency.

The results are demonstrated on a pose regulation task that the learned policies are able to successfully stabilize the underwater robot in a relatively small region around the target under such excessive time-correlated disturbances. Also, DOB-Net outperforms HWRL on the stabilization performance, and the modular learning framework, GCP-ODI, show improved sample efficiency and control performance compared with the joint learning algorithms. But the policies trained on simulated disturbance waveforms cannot directly adopt to real-world wave data with good control stability.

7.1.2 Transferring Disturbance Rejection Policy under Dynamics Model Mismatch

Besides learning disturbance rejection control, we also apply transfer learning techniques to adapt a learned control policy to the mismatch in dynamics model. In this research, we formulate a transfer learning problem where the source and target tasks differ in system dynamics, the former defines a mathematical model of SPIR developed from the fundamental principles of dynamics, the latter applies an empirical model of SPIR derived from real-world experimental data. Hybrid Policy Adaptation (HPA) is introduced, that combines high sample efficiency of model-based learning with high task-specific performance of model-free learning. HPA employs a model-based controller (i.e., iLQR) to generate trajectories on the mathematical model, then optimizes a control policy using supervised learning to imitate the behaviours of iLQR. The transfer learning can be implemented by either training an additional model-free policy to compensate the output of model-based policy (i.e., HPA-comp), or using the model-based policy as an initialization of a model-free policy (i.e., HPA-init), which is further fine-tuned under the empirical model.

HPA treats the external disturbances as parts of the system dynamics and adapts their mismatch together during transfer. To further improve the transfer performance, we then have developed a transfer learning algorithm based on the modular architecture of GCP-ODI, called Transition Mismatch Learning (TML), where the disturbances are separated from the system dynamics and processed independently. When the learned control policy is deployed on the empirical model, a compensatory policy is trained as additional control, through maximizing task performance as well as minimizing mismatch of transitions predicted by the mathematical model and the empirical model in the meantime. In this way, ODI is able to predict the waveforms that best fit the real-world wave data based on the given distributions of the simulated disturbance parameters, then the total model mismatch required to be compensated can be minimized, and thus the learning difficulty during transfer is reduced. In addition, middle layer features instead of final network outputs of compensatory policy and generalized policy have been combined to mitigate the impact of control constraints (i.e. TML-feature).

The transfer learning algorithms are evaluated using the mathematical model and the empirical model both in simulation. As a result, both algorithms achieve satisfactory performance on the empirical model after transfer, in the meantime significantly enhance the sample efficiency with respect to learning from scratch without transfer. For HPA, the initialization approach (HPA-init) achieves better results due to the inclusion of the simulated disturbances in the mathematical model. Furthermore, transfer learning using TML outperforms that using HPA in terms of the sample efficiency.

7.2 Discussion and Future Work

7.2.1 Dimensionality of Identification Model

When using GCP-ODI, we have demonstrated that ODI has good performance with simulated disturbances composed of $3 \sim 5$ sinusoidal waves. If there are more component sinusoidal waves in simulated disturbances, then more realistic disturbance waveforms ODI can predict. But as the number of component waves increases (over 5), identifying high-dimensional disturbance parameters becomes challenging. Because it typically requires millions of training samples to cover a large output space of ODI, both trajectory data gathering by GCP and training of ODI have exponentially increased difficulty. More rigorous analysis is needed to evaluate the sample efficiency of GCP-ODI regarding high-dimensional disturbance parameters.

7.2.2 Theoretical Proof of Convergence

For both the disturbance rejection control using GCP-ODI and the transfer learning using TML, we can see that these policies can achieve similar performance with the corresponding baselines, which are GCP given the true disturbance parameters and GCP-ODI directly trained on the target task, respectively. However, the conditions of convergence and the theoretical limits of the proposed algorithms have not been established in this research, thus definitely require further investigation.

7.2.3 Representation of Wave Disturbances

In the formulation of modular learning algorithm GCP-ODI, GCP takes a fixed length of time-domain disturbance values as additional input, constructed from the predicted disturbance parameters of ODI. Such formulation outperforms using frequency-domain signals (i.e. predicted disturbance parameters), but it provides only partial information of the disturbance waveforms, thus cannot ensure the control solutions are optimal. A more efficient representation of the disturbance information is required.

In addition, real-world wave forces may be jointly determined by fluid conditions, robot morphology, as well as varying robot states and controls, and may vary with not only time but also space. Thus, using a superposition of multiple sinusoidal waves, which are only functions of time, to simulate the disturbances may not be sufficient. A comprehensive and multivariable function is required for a better description of the wave forces. We can seek a machine learning approach to explicitly build a wave model for interested water areas.

7.2.4 Real Robot Deployment

This thesis only covers the work using an empirical model in simulation, while the ultimate goal is to deploy the transfer learning algorithm on real-world robotic systems. This can be implemented through two ways, one is to deploy the underwater robot in a water tank, then a good wave environment needs to be set up. Such environment can be created by using professional devices like wave generators, but how the generated waves are related to the device location and output needs further investigation. Another way is to directly deploy the robot in open water, where an accurate localization system would be necessary. In the open water, self-positioning scenario is obviously more reasonable with respect to external positioning, since it is nearly impossible to deploy external markers or cameras in the open water. For the self-positioning, current solutions mainly focus on Simultaneous Localization And Mapping (SLAM), which can be implemented based on vision, IMU or sonar. This will introduce another research field, and the biggest concerns will be the localization accuracy on an unstable platform and the blurred images underwater.

7.2.5 Online Model Learning

The learning efficiency of the transfer learning directly depends on the mismatch between the mathematical model and the empirical model or the real robot. Sometimes, we may not be given a good prior model of the robot, then there is no way to pretrain a policy that helps in the transfer. In that case, we may investigate model-based reinforcement learning that online learns a model then optimizes a policy based on the model. But the dynamics model of an underwater robot under wave forces is ever-changing, we need a model that nearly constant at least in a certain period of time. Potential solutions include history window model $\boldsymbol{x}_{t+1} = f\left(\boldsymbol{x}_{t-T_H}, \boldsymbol{u}_{t-T_H}, \cdots, \boldsymbol{x}_t, \boldsymbol{u}_t\right)$, recurrent neural network model $\boldsymbol{x}_{t+1} = f\left(\boldsymbol{x}_0, \boldsymbol{u}_0, \cdots, \boldsymbol{x}_t, \boldsymbol{u}_t\right)$ or multi-step model $\left(\boldsymbol{x}_{t+1}, \cdots, \boldsymbol{x}_{t+N}\right) = f\left(\boldsymbol{x}_t, \boldsymbol{u}_t, \cdots, \boldsymbol{u}_{t+N-1}\right)$. Then, MPC or LQR can be employed to generate optimal control signals based on the learned model. Such model-based learning approaches still outperforms purely model-free reinforcement learning, and is able to overcome the constraints of real-time sample collection in the real world.

Bibliography

Gianluca Antonelli. *Underwater robots*, volume 123. Springer, 2018.

Gwyn Griffiths. *Technology and applications of autonomous underwater vehicles*, volume 2. CRC Press, 2002.

RG Dean and RA Dalrymple. Water wave mechanics for scientists and engineers. *Advanced Series on Ocean Engineering, World Scientific*, 2, 1991.

Daniel C Fernández and Geoffrey A Hollinger. Model predictive control for underwater robots in ocean waves. *IEEE Robotics and Automation letters*, 2(1):88–95, 2016.

Jonathan Woolfrey, Dikai Liu, and Marc Carmichael. Kinematic control of an autonomous underwater vehicle-manipulator system (auvms) using autoregressive prediction of vehicle motion and model predictive control. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 4591–4596. IEEE, 2016.

Tianming Wang, Wenjie Lu, and Dikai Liu. A case study: Modeling of a passive flexible link on a floating platform for intervention tasks. In *2018 13th World Congress on Intelligent Control and Automation (WCICA)*, pages 187–193. IEEE, 2018.

Liang-Liang Xie and Lei Guo. How much uncertainty can be dealt with by feedback? *IEEE Transactions on Automatic Control*, 45(12):2203–2217, 2000.

Zhiqiang Gao. On the centrality of disturbance rejection in automatic control. *ISA transactions*, 53(4):850–857, 2014.

Shihua Li, Jun Yang, Wen-Hua Chen, and Xisong Chen. *Disturbance observer-based control: methods and applications*. CRC press, 2014.

Steven Waslander and Carlos Wang. Wind disturbance estimation and rejection for quadrotor position control. In *AIAA Infotech@ Aerospace Conference and AIAA Unmanned... Unlimited Conference*, page 1983, 2009.

Sigurd Skogestad and Ian Postlethwaite. *Multivariable feedback control: analysis and design*, volume 2. Wiley New York, 2007.

Karl J Åström and Björn Wittenmark. *Adaptive control*. Courier Corporation, 2013.

Wenjie Lu and Dikai Liu. Active task design in adaptive control of redundant robotic systems. In *Australasian Conference on Robotics and Automation*. ARAA, 2017.

Wenjie Lu and Dikai Liu. A frequency-limited adaptive controller for underwater vehicle-manipulator systems under large wave disturbances. In *2018 13th World Congress on Intelligent Control and Automation (WCICA)*, pages 246–251. IEEE, 2018.

Dimitri P Bertsekas, Dimitri P Bertsekas, Dimitri P Bertsekas, and Dimitri P Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena scientific Belmont, MA, 1995.

Christopher Edwards and Sarah Spurgeon. *Sliding mode control: theory and applications*. Crc Press, 1998.

John C Doyle, Keith Glover, Pramod P Khargonekar, and Bruce A Francis. State-space solutions to standard h/sub 2/and h/sub infinity/control problems. *IEEE Transactions on Automatic control*, 34(8):831–847, 1989.

Hamed Ghafarirad, Seyed Mehdi Rezaei, Mohammad Zareinejad, and Ahmed AD Sarhan. Disturbance rejection-based robust control for micropositioning of piezoelectric actuators. *Comptes Rendus Mécanique*, 342(1):32–45, 2014.

Haiyan Gao and Yuanli Cai. Nonlinear disturbance observer-based model predictive control for a generic hypersonic vehicle. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, 230(1):3–12, 2016.

Jun Yang, Shihua Li, Xisong Chen, and Qi Li. Disturbance rejection of ball mill grinding circuits using dob and mpc. *Powder Technology*, 198(2):219–228, 2010.

Wen-Hua Chen, Jun Yang, Lei Guo, and Shihua Li. Disturbance-observer-based control and related methods—an overview. *IEEE Transactions on Industrial Electronics*, 63(2): 1083–1095, 2016.

Kiyoshi Ohishi, Masato Nakao, Kouhei Ohnishi, and Kunio Miyachi. Microprocessor-controlled dc motor for load-insensitive position servo system. *IEEE Transactions on Industrial Electronics*, (1):44–49, 1987.

Wen-Hua Chen, Donald J Ballance, Peter J Gawthrop, and John O'Reilly. A nonlinear disturbance observer for robotic manipulators. *IEEE Transactions on industrial Electronics*, 47(4):932–938, 2000.

Takaji Umeno, Tomoaki Kaneko, and Yoichi Hori. Robust servosystem design with two degrees of freedom and its application to novel motion control of robot manipulators. *IEEE Transactions on Industrial Electronics*, 40(5):473–485, 1993.

Takaji Umeno and Yoichi Hori. Robust speed control of dc servomotors using modern two degrees-of-freedom controller design. *IEEE Transactions on Industrial Electronics*, 38 (5):363–368, 1991.

C Johnson. Optimal control of the linear regulator with constant disturbances. *IEEE Transactions on Automatic Control*, 13(4):416–421, 1968.

Cn Johnson. Accomodation of external disturbances in linear regulator and servomechanism problems. *IEEE Transactions on Automatic Control*, 16(6):635–644, 1971.

JQ Han. The extended state observer of a class of uncertain systems. *Control and decision*, 10(1):85–88, 1995.

Zhiqiang Gao, Yi Huang, and Jingqing Han. An alternative paradigm for control system design. In *Decision and Control, 2001. Proceedings of the 40th IEEE Conference on*, volume 5, pages 4578–4585. IEEE, 2001.

Haibin Sun and Lei Guo. Neural network-based dobc for a class of nonlinear systems with unmatched disturbances. *IEEE transactions on neural networks and learning systems*, 28(2):482–489, 2016.

Carlos E Garcia, David M Prett, and Manfred Morari. Model predictive control: theory and practice—a survey. *Automatica*, 25(3):335–348, 1989.

Eduardo F Camacho and Carlos Bordons Alba. *Model predictive control*. Springer Science & Business Media, 2013.

Urban Maeder and Manfred Morari. Offset-free reference tracking with model predictive control. *Automatica*, 46(9):1469–1476, 2010.

Jun Yang, Shihua Li, Xisong Chen, and Qi Li. Disturbance rejection of dead-time processes using disturbance observer and model predictive control. *Chemical engineering research and design*, 89(2):125–135, 2011.

Cunjia Liu, Wen-Hua Chen, and John Andrews. Tracking control of small-scale helicopters using explicit nonlinear mpc augmented with disturbance observers. *Control Engineering Practice*, 20(3):258–268, 2012.

Jun Yang, Zhenhua Zhao, Shihua Li, and Wei Xing Zheng. Nonlinear disturbance observer enhanced predictive control for airbreathing hypersonic vehicles. In *Control Conference (CCC), 2014 33rd Chinese*, pages 3668–3673. IEEE, 2014.

Christian Dirscherl, CM Hackl, and Korbinian Schechner. Explicit model predictive control with disturbance observer for grid-connected voltage source power converters. In *Industrial Technology (ICIT), 2015 IEEE International Conference on*, pages 999–1006. IEEE, 2015.

Samarth Brahmbhatt and James Hays. Deepnav: Learning to navigate large cities. In *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, pages 3087–3096, 2017.

Peter Karkus, David Hsu, and Wee Sun Lee. Particle filter networks: End-to-end probabilistic localization from visual observations. *arXiv preprint arXiv:1805.08975*, 2018.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Karl J Åström. *Introduction to stochastic control theory*. Courier Corporation, 2012.

Lei Guo and Hong Wang. *Stochastic distribution control system design: a convex optimization approach.* Springer Science & Business Media, 2010.

Kemin Zhou and John Comstock Doyle. *Essentials of robust control*, volume 104. Prentice hall Upper Saddle River, NJ, 1998.

Arjan J Van Der Schaft. L/sub 2/-gain analysis of nonlinear systems and nonlinear state-feedback h/sub infinity/control. *IEEE transactions on automatic control*, 37(6):770–784, 1992.

Chua-Liang Lin and Tsai-Yuan Lin. An h/sub/spl infin//design approach for neural net-based control schemes. *IEEE Transactions on Automatic Control*, 46(10):1599–1605, 2001.

Jingqing Han. From pid to active disturbance rejection control. *IEEE transactions on Industrial Electronics*, 56(3):900–906, 2009.

Jie Huang. *Nonlinear output regulation: theory and applications.* SIAM, 2004.

Carlos E Garcia and Manfred Morari. Internal model control. a unifying review and some new results. *Industrial & Engineering Chemistry Process Design and Development*, 21 (2):308–323, 1982.

Enrico S Canuto and Fabio Musso. Embedded model control: application to web winding. part i: modelling. In *Emerging Technologies and Factory Automation, 2006. ETFA'06. IEEE Conference on*, pages 477–484. IEEE, 2006.

Alberto Isidori and Christopher I Byrnes. Output regulation of nonlinear systems. *IEEE transactions on Automatic Control*, 35(2):131–140, 1990.

Riccardo Marino and Patrizio Tomei. *Nonlinear control design: geometric, adaptive and robust.* Prentice Hall International (UK) Ltd., 1996.

Christopher I Byrnes, Francesco Delli Priscoli, and Alberto Isidori. *Output regulation of uncertain nonlinear systems.* Springer Science & Business Media, 2012.

Jie Huang and Zhiyong Chen. A general framework for tackling the output regulation problem. *IEEE Transactions on Automatic Control*, 49(12):2203–2218, 2004.

B Xian, N Jalili, DM Dawson, and Y Fang. Adaptive tracking control of linear uncertain mechanical systems subjected to unknown sinusoidal disturbances. *TRANSACTIONS-AMERICAN SOCIETY OF MECHANICAL ENGINEERS JOURNAL OF DYNAMIC SYSTEMS MEASUREMENT AND CONTROL*, 125(1):129–133, 2003.

Zairong Xi and Zhengtao Ding. Global adaptive output regulation of a class of nonlinear systems with nonlinear exosystems. *Automatica*, 43(1):143–149, 2007.

VO Nikiforov. Nonlinear servocompensation of unknown external disturbances. *Automatica*, 37(10):1647–1653, 2001.

Marc Bodson, Jonathan S Jensen, and Scott C Douglas. Active noise control for periodic disturbances. *IEEE Transactions on control systems technology*, 9(1):200–205, 2001.

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, 2015.

Junhyuk Oh, Valliappa Chockalingam, Satinder Singh, and Honglak Lee. Control of memory, active perception, and action in minecraft. *arXiv preprint arXiv:1605.09128*, 2016.

Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. In *International Conference on Machine Learning*, pages 2829–2838, 2016a.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015a.

Shixiang Gu, Timothy Lillicrap, Zoubin Ghahramani, Richard E Turner, and Sergey Levine. Q-prop: Sample-efficient policy gradient with an off-policy critic. *arXiv preprint arXiv:1611.02247*, 2016b.

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.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015b.

Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

Guy Shani, Joelle Pineau, and Robert Kaplow. A survey of point-based pomdp solvers. *Autonomous Agents and Multi-Agent Systems*, 27(1):1–51, 2013.

Daan Wierstra, Alexander Förster, Jan Peters, and Jürgen Schmidhuber. Recurrent policy gradients. *Logic Journal of the IGPL*, 18(5):620–634, 2010.

Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. In *2015 AAAI Fall Symposium Series*, 2015.

Nicolas Heess, Jonathan J Hunt, Timothy P Lillicrap, and David Silver. Memory-based control with recurrent neural networks. *arXiv preprint arXiv:1512.04455*, 2015.

Daan Wierstra, Alexander Foerster, Jan Peters, and Juergen Schmidhuber. Solving deep memory pomdps with recurrent policy gradients. In *International Conference on Artificial Neural Networks*, pages 697–706. Springer, 2007.

Ivan Sorokin, Alexey Seleznev, Mikhail Pavlov, Aleksandr Fedorov, and Anastasiia Ignateva. Deep attention recurrent q-network. *arXiv preprint arXiv:1512.01693*, 2015.

Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. Social lstm: Human trajectory prediction in crowded spaces. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 961–971, 2016.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org, 2017.

Anusha Nagabandi, Ignasi Clavera, Simin Liu, Ronald S Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. *arXiv preprint arXiv:1803.11347*, 2018a.

Kate Rakelly, Aurick Zhou, Deirdre Quillen, Chelsea Finn, and Sergey Levine. Efficient off-policy meta-reinforcement learning via probabilistic context variables. *arXiv preprint arXiv:1903.08254*, 2019.

Adam Daniel Laud. Theory and application of reward shaping in reinforcement learning. Technical report, 2004.

Marek Grzes and Daniel Kudenko. Plan-based reward shaping for reinforcement learning. In *2008 4th International IEEE Conference Intelligent Systems*, volume 2, pages 10–22. IEEE, 2008.

Denis Yarats, Amy Zhang, Ilya Kostrikov, Brandon Amos, Joelle Pineau, and Rob Fergus. Improving sample efficiency in model-free reinforcement learning from images. *arXiv preprint arXiv:1910.01741*, 2019.

Simon S Du, Sham M Kakade, Ruosong Wang, and Lin F Yang. Is a good representation sufficient for sample efficient reinforcement learning? *arXiv preprint arXiv:1910.03016*, 2019.

Jie Tang and Pieter Abbeel. On a connection between importance sampling and the likelihood ratio policy gradient. In *Advances in Neural Information Processing Systems*, pages 1000–1008, 2010.

Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*, 2016.

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

Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685, 2009.

Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3357–3364. IEEE, 2017.

Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016a.

Andrei A Rusu, Matej Vecerik, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. Sim-to-real robot learning from pixels with progressive nets. *arXiv preprint arXiv:1610.04286*, 2016b.

Eric Tzeng, Coline Devin, Judy Hoffman, Chelsea Finn, Pieter Abbeel, Sergey Levine, Kate Saenko, and Trevor Darrell. Adapting deep visuomotor representations with weak pairwise constraints. In *Algorithmic Foundations of Robotics XII*, pages 688–703. Springer, 2020.

Konstantinos Bousmalis, Alex Irpan, Paul Wohlhart, Yunfei Bai, Matthew Kelcey, Mrinal Kalakrishnan, Laura Downs, Julian Ibarz, Peter Pastor, Kurt Konolige, et al. Using simulation and domain adaptation to improve efficiency of deep robotic grasping. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4243–4250. IEEE, 2018.

Ljung Lennart. System identification: theory for the user. *PTR Prentice Hall, Upper Saddle River, NJ*, pages 1–14, 1999.

Fouad Giri and Er-Wei Bai. *Block-oriented nonlinear system identification*, volume 1. Springer, 2010.

Marc Peter Deisenroth, Gerhard Neumann, Jan Peters, et al. A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2(1–2):1–142, 2013.

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 (ICML-11)*, pages 465–472, 2011.

Leonid Kuvayev and Richard S Sutton. Model-based reinforcement learning with an approximate, learned model. In *in Proceedings of the Ninth Yale Workshop on Adaptive and Learning Systems*. Citeseer, 1996.

Jeffrey Forbes and David Andre. Representations for learning control policies. In *Proceedings of the ICML-2002 Workshop on Development of Representations*, pages 7–14, 2002.

Todd Hester and Peter Stone. Intrinsically motivated model learning for developing curious robots. *Artificial Intelligence*, 247:170–186, 2017.

Nicholas K Jong and Peter Stone. Model-based function approximation in reinforcement learning. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, page 95. ACM, 2007.

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

Charles Brauer. Using eureqa in a stock day-trading application. *Cypress Point Technologies, LLC*, 2012.

Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *science*, 324(5923):81–85, 2009.

Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L Lewis, and Satinder Singh. Action-conditional video prediction using deep networks in atari games. In *Advances in neural information processing systems*, pages 2863–2871, 2015.

Manuel Watter, Jost Springenberg, Joschka Boedecker, and Martin Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in neural information processing systems*, pages 2746–2754, 2015.

Niklas Wahlström, Thomas B Schön, and Marc Peter Deisenroth. From pixels to torques: Policy learning with deep dynamical models. *arXiv preprint arXiv:1502.02251*, 2015a.

Niklas Wahlström, Thomas B Schön, and Marc Peter Deisenroth. Learning deep dynamical models from image pixels. *IFAC-PapersOnLine*, 48(28):1059–1064, 2015b.

Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Deep spatial autoencoders for visuomotor learning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 512–519. IEEE, 2016.

Bradly C Stadie, Sergey Levine, and Pieter Abbeel. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*, 2015.

Pieter Abbeel and Andrew Y Ng. Exploration and apprenticeship learning in reinforcement learning. In *Proceedings of the 22nd international conference on Machine learning*, pages 1–8, 2005.

Michel Gevers et al. System identification without lennart ljung: what would have been different? *Forever Ljung in System Identification, Studentlitteratur AB, Norrtalje*, 2, 2006.

Josh C Bongard and Hod Lipson. Nonlinear system identification using coevolution of models and tests. *IEEE Transactions on Evolutionary Computation*, 9(4):361–384, 2005.

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 *Robotics and Automation (ICRA), 2018 IEEE International Conference on*, pages 7579–7586. IEEE, 2018b.

Visak CV Kumar, Sehoon Ha, and Katsu Yamane. Improving model-based balance controllers using reinforcement learning and adaptive sampling. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7541–7547. IEEE, 2018.

Ivan Koryakovskiy, Manuel Kudruss, Heike Vallery, Robert Babuška, and Wouter Caarls. Model-plant mismatch compensation using reinforcement learning. *IEEE Robotics and Automation Letters*, 3(3):2471–2477, 2018.

Vladimir Feinberg, Alvin Wan, Ion Stoica, Michael I Jordan, Joseph E Gonzalez, and Sergey Levine. Model-based value estimation for efficient model-free reinforcement learning. *arXiv preprint arXiv:1803.00101*, 2018.

Justin A Boyan. Least-squares temporal difference learning. In *International Conference on Machine Learning*, pages 49–56. Citeseer, 1999.

Vitchyr Pong, Shixiang Gu, Murtaza Dalal, and Sergey Levine. Temporal difference models: Model-free deep rl for model-based control. *arXiv preprint arXiv:1802.09081*, 2018.

Fereshteh Sadeghi and Sergey Levine. Cad2rl: Real single-image flight without a single real image. *arXiv preprint arXiv:1611.04201*, 2016.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017.

Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018.

OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8973–8979. IEEE, 2019.

Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. *arXiv preprint arXiv:1804.10332*, 2018.

Kendall Lowrey, Svetoslav Kolev, Jeremy Dao, Aravind Rajeswaran, and Emanuel Todorov. Reinforcement learning for non-prehensile manipulation: Transfer from simulation to physical system. In *2018 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, pages 35–42. IEEE, 2018.

Rika Antonova, Silvia Cruciani, Christian Smith, and Danica Kragic. Reinforcement learning for pivoting task. *arXiv preprint arXiv:1703.00472*, 2017.

Aravind Rajeswaran, Sarvjeet Ghotra, Balaraman Ravindran, and Sergey Levine. Epopt: Learning robust neural network policies using model ensembles. *arXiv preprint arXiv:1610.01283*, 2016.

Silvia Chiappa, Sébastien Racaniere, Daan Wierstra, and Shakir Mohamed. Recurrent environment simulators. *arXiv preprint arXiv:1704.02254*, 2017.

Yuval Tassa, Nicolas Mansard, and Emo Todorov. Control-limited differential dynamic programming. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 1168–1175. IEEE, 2014.

Steindór Sæmundsson, Katja Hofmann, and Marc Peter Deisenroth. Meta reinforcement learning with latent variable gaussian processes. *arXiv preprint arXiv:1803.07551*, 2018.

W-H Chen, Donald J Ballance, Peter J Gawthrop, Jenny J Gribble, and John O'Reilly. Nonlinear pid predictive controller. *IEE Proceedings-Control Theory and Applications*, 146(6):603–611, 1999.

Shihua Li, Jun Yang, Wen-Hua Chen, and Xisong Chen. *Disturbance observer-based control: methods and applications.* CRC press, 2016.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Nicholas Fischer, Devin Hughes, Patrick Walters, Eric M Schwartz, and Warren E Dixon. Nonlinear rise-based control of an autonomous underwater vehicle. *IEEE Transactions on Robotics*, 30(4):845–852, 2014.

Wenhao Yu, C Karen Liu, and Greg Turk. Preparing for the unknown: Learning a universal policy with online system identification. In *Proceedings of Robotics: Science and Systems*, Cambridge, Massachusetts, July 2017. doi: 10.15607/RSS.2017.XIII.048.

Lei Tai, Giuseppe Paolo, and Ming Liu. Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 31–36. IEEE, 2017.

Weiwei Li and Emanuel Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. In *ICINCO (1)*, pages 222–229, 2004.

Emanuel Todorov and Weiwei Li. A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *American Control Conference, 2005. Proceedings of the 2005*, pages 300–306. IEEE, 2005.

Paul Zarchan and Howard Musoff. *Fundamentals of Kalman filtering: a practical approach.* American Institute of Aeronautics and Astronautics, Inc., 2013.