Modular Transfer Learning with Transition Mismatch Compensation for Excessive Disturbance Rejection

Tianming Wang, *Student Member, IEEE,* Wenjie Lu, *Member, IEEE,* Huan Yu, *Student Member, IEEE,* and Dikai Liu, *Member, IEEE*

Abstract—Underwater robots in shallow waters usually suffer from strong wave forces, which may frequently exceed robot's control constraints. Learning-based controllers are suitable for disturbance rejection control, but the excessive disturbances heavily affect the state transition in Markov Decision Process (MDP) or Partially Observable Markov Decision Process (POMDP). Also, pure learning procedures on targeted system may encounter damaging exploratory actions or unpredictable system variations, and training exclusively on a prior model usually cannot address model mismatch from the targeted system. In this paper, we propose a transfer learning framework that adapts a control policy for excessive disturbance rejection of an underwater robot under dynamics model mismatch. A modular network of learning policies is applied, composed of a Generalized Control Policy (GCP) and an Online Disturbance Identification Model (ODI). GCP is first trained over a wide array of disturbance waveforms. ODI then learns to use past states and actions of the system to predict the disturbance waveforms which are provided as input to GCP (along with the system state). A transfer reinforcement learning algorithm using Transition Mismatch Compensation (TMC) is developed based on the modular architecture, that learns an additional compensatory policy through minimizing mismatch of transitions predicted by the two dynamics models of the source and target tasks. We demonstrated on a pose regulation task in simulation that TMC is able to successfully reject the disturbances and stabilize the robot under an empirical model of the robot system, meanwhile improve sample efficiency.

Index Terms—Underwater robot, varying environment, disturbance rejection, reinforcement learning, transfer learning.

I. Introduction

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 underwater tasks are more and more common, such as sea bottom survey, offshore structures monitoring, pipeline maintenance, biological samples collection and shipwreck search [1], [2]. The rising demand for robotic advancements

T. Wang and D. Liu are with the Centre for Autonomous Systems, Faculty of Engineering and Information Technology, University of Technology Sydney, Ultimo, NSW 2007, Australia (e-mail: tianming.wang@student.uts.edu.au; dikai.liu@uts.edu.au).

W. Lu is with the School of Mechanical Engineering and Automation, Harbin Institute of Technology (Shenzhen), Shenzhen, 518055 Guangdong, P.R. China (e-mail: luwenjie@hit.edu.cn).

H. Yu is with the School of Automation, Beijing Institute of Technology, 100081 Beijing, P.R. China (e-mail: yuhuan.bit@gmail.com).
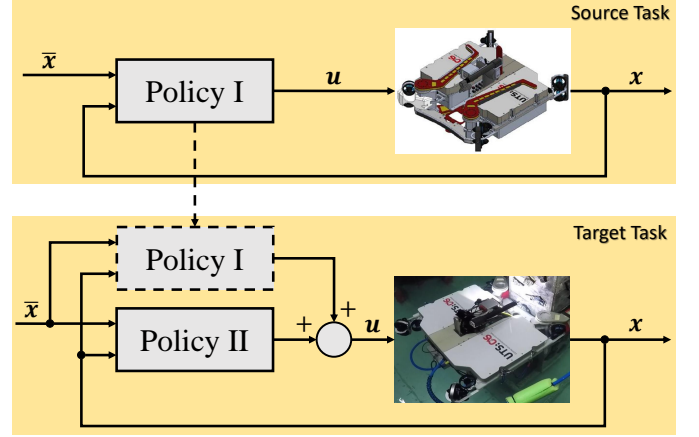
Fig. 1. Diagram of transfer learning in reinforcement learning between two different dynamics models of underwater robots, Policy II produces a compensatory control action and runs in parallel with Policy I in the target task.

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 [3]. Owing to this decay, as well as the considerable size and thrust capabilities of underwater robotic systems, the strength and changes of ocean waves are often neglected in robot motion planning and control in deep water applications [4]. In field applications with low operational depths and turbulent wave climates, like bridge pile inspection [5] and sea-ice algae characterization in Antarctica [6], this assumption can quickly break down, since shallow water environments usually accommodate only small-size robots that have limited thrust capabilities, 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 this paper). As a result, increased wave forces inevitably hinder the stability and precision of robot motion control [7]–[9].

Disturbance Observer Based Control (DOBC) [10]–[12] has been widely investigated for decades. The main objective is to estimate the unknown disturbances from measurable variables, then take a control action based on the disturbance estimation to compensate for the influence of the disturbances. In this

method, external disturbances and model uncertainties are generally lumped together, 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, thus making it difficult for predicting future disturbances. A further problem is that naive addition of the disturbance estimation onto the control signal leads to inputs over constraints.

To this end, Model Predictive Control (MPC) [4], [13], [14] is often applied due to its constraint handling capability through optimizing system behaviours over a future time horizon [15]. However, the performance of MPC relies on the accuracy of the given system dynamics model, and the requirement for online optimization at each timestep leads to low computational efficiency.

In contrast, Reinforcement Learning (RL) [16] has drawn a lot of attention in learning optimal controllers for systems that are difficult to model accurately. It is a trial-and-error approach that allows to find an optimal sequence of commands without any prior assumption about the world, and can naturally adapt to uncertainties and noises (particularly noises that are independent and identically distributed) in the real system. Meanwhile, 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. But 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 unknown disturbances, 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). However, even if the disturbances are considered as the unobservable parts of the state space, the underlying transition function may still not be able to define a MDP, since it can be difficult to predict the next disturbances based on the current state (including disturbances) and action only. Thus, the first research questions in this paper are how to find a better state space to represent the disturbances, so that the problem of excessive disturbance rejection can be formulated into a POMDP, then how to learn a control policy in such POMDP.

In addition, RL is able to successfully solve problems when being trained directly on the targeted system. Sometimes, training samples are expensive to obtain on the targeted system. 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. Other times, the targeted system may change over time in an unpredictable way during operation. These variations in the dynamics model may include variable friction coefficients, actuator failures, or varying load to be manipulated. In such cases, it is difficult to implement pure learning procedures on the targeted system.

A promising approach is to make an initial guess of the targeted system, such as a simulated counterpart of the real robot or a static model of the system before the 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 targeted system dynamics, initially learned policies usually cannot be directly applied on the targeted system. Transfer learning offers a pathway to bridge the mismatch between different dynamics models. Thus, the second research question in this paper is how to transfer a control policy for excessive disturbance rejection under dynamics model mismatch.

This paper introduces a transfer learning framework that enables successful deployment of a control policy for excessive disturbance rejection of an underwater robot under dynamics model mismatch, as shown in Fig. 1. The contributions of this work are threefold. Firstly, we seek a modular design of learning policies for excessive disturbance rejection, consisting of an observer network and a controller network. The observer is used to predict disturbance waveforms which are then provided as input to the controller together with the current system state to produce control action. The modular architecture benefits policy transfer between different dynamics in sample efficiency, since only the controller network needs to be adapted, but the observer network can remain fixed. Secondly, transfer learning is developed based on the modular architecture. To ensure the observer working correctly, an additional compensatory policy is learned through minimizing the mismatch of transitions predicted by the two dynamics models of the source and target tasks, respectively. Such design enables the observer to predict the external disturbances in the target task. Then the model mismatch exists almost only in the internal dynamics, leading to reduced transfer difficulty. Thirdly, the compensatory policy is added in terms of middle layer features instead of final network outputs in the target task, in order to offer more flexibility in compensation under the control constraints.

The transfer learning algorithms are evaluated on a pose regulation task under unobservable wave forces in simulation. In the evaluation, the source task defines a first-principle model of an underwater robot developed from the fundamental principles of dynamics, the target task applies an empirical model of an underwater robot derived from real-world experimental data. As a result, a control policy trained on the first-principle model is still able to successfully reject the disturbances and stabilize the robot on the empirical model through a small amount of adaptation.

In this paper, Section II covers a review of related work in the current literature. Section III introduces our problem formulation. Section IV provides the detailed description of the modular network design for excessive disturbance rejection control, followed by the details of the transfer learning algorithms in Section V. Then, Section VI presents experimental evaluation procedures and result analysis. Limitations and some potential future improvements are discussed in the last section (Section VII).

II. RELATED WORK

A. Reinforcement Learning in Partially Observable Markov Decision Processes

Deep RL algorithms based on Q-learning [17]–[19], policy gradients [20], [21], and actor-critic methods [22]–[24] 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. Then POMDP better describes the dynamics of real-world environments through explicitly recognizing that the agent only observes partial glimpses of the underlying system state. Existing solutions to 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 [25], [26].

Using Recurrent Neural Network (RNN) to represent policies is a more popular approach to handle partial observability [18], [23], [27]–[29]. The idea being that 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. 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 [28] 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) [30], 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. In addition, RNN is not limited to the learning algorithms based on value function, there are also successful applications to policy gradients [27] and actor-critic methods [23], [29].

The problem of excessive disturbance rejection has some differences from the generic POMDP. Normally, POMDP can be transformed back to MDP when there is full observability of the environment. However, this is not the case when the disturbances are considered as the unobservable parts of the state space, since it it difficult to formulate a transition function to predict next disturbances from current state (including disturbances) and action only. Both history window approach [31] and recurrent policy [32] attempt to resolve this issue through characterizing the disturbed system transition as a multi-step MDP, and assuming the unobservable disturbance waveforms are encoded in robot motion history. The difference lies in the way to use the history data, the history window approach directly takes most recent state-action pairs as additional input to the policy, while the recurrent policy employs RNN to explore past experience in order to learn an optimal embedding of history data.

In contrast to these "end-to-end" control policies, that take as input a sequence of motion history and directly output the optimal control, this paper explicitly decouples the process into disturbance identification and motion control. We believe that using the decoupled moderate-sized networks instead of a large network trained by RL in an end-to-end mode [33], might mitigate the learning difficulty and thus improve the sample efficiency, and such modular network design benefits policy transfer between different dynamics as well.

B. Transfer Learning in Reinforcement Learning

Although RL algorithms can learn complex skills in high-dimensional state and action spaces, it is almost impossible to directly deploy RL agent on real-world systems, due to the high sample complexity. In order to accelerate the learning process of RL, the knowledge previously obtained from related tasks can be used [34], [35]. Transferring policy from one task to another [36], especially from learning in physical simulators to adapting on real robots, has aroused great interest. The most simple idea is to use identical network architecture for both simulation and real environment [37]. 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 [38], [39]. There are also other methods including domain adaptation that learns aligned visual representations between synthetic and real-world images [40], [41].

Much of the previous work has focused on system identification [42], [43], 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 [44]. 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. Various model-based RL methods have been proposed [45]–[50], and applied successfully on both simulated and real-world robots, such as inverted pendulums [45], manipulators [51], and legged robots [52].

In fact, system identification is usually interleaved with policy optimization [53]–[56]. 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 an optimal policy.

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 system dynamics have been used to design controllers that are robust to model uncertainties. Peng et al. [57] 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. [58] 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 [59]. Thus, system identification and domain randomization have also been combined [60]–[62]. 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. [63] and Chebotar et al. [59] 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.

In summary, although neural-network-based dynamics model can make reasonable predictions over a future time horizon without physical interaction [64], the success of model-based RL still heavily depends on the quality and quantity of the real-world experimental data. 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.

III. PROBLEM FORMULATION

A. Disturbed Robot Dynamics

Consider a nonlinear time-invariant system to represent the dynamics of an underwater robot in the form of

$$\boldsymbol{M}\dot{\boldsymbol{\nu}} + \boldsymbol{C}\left(\boldsymbol{\nu}\right)\boldsymbol{\nu} + \boldsymbol{D}_{RB}\left(\boldsymbol{\nu}\right)\boldsymbol{\nu} + \boldsymbol{g}_{RB}\left(\boldsymbol{\eta}\right) = \boldsymbol{u} + \boldsymbol{d} + \boldsymbol{\xi},$$
$$\dot{\boldsymbol{\eta}} = \boldsymbol{J}\boldsymbol{\nu}, \qquad (1)$$

where $\boldsymbol{\eta}, \boldsymbol{\nu} \in \mathbb{R}^6$ are the robot's pose and velocity, $\boldsymbol{x} = \left[\boldsymbol{\eta}^T \ \boldsymbol{\nu}^T\right]^T \in \mathcal{X} \in \mathbb{R}^{12}$ is the system state, $\mathcal{X}$ represents the state space, $\boldsymbol{J} \in \mathbb{R}^{6 \times 6}$ 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 the wave forces, and $\boldsymbol{\xi} \in \mathbb{R}^6$ represents the model uncertainties.

In this work, the wave forces $\boldsymbol{d}$ are considered as external disturbances to the system, all the other components in the system dynamics model (1) are termed as internal dynamics. The external disturbances are assumed to be time-correlated signals following specific waveforms, and may frequently exceed the control constraints of the robot. We consider the control constraints of the form $\underline{\boldsymbol{u}} \leq \boldsymbol{u} \leq \overline{\boldsymbol{u}}$ with element-wise inequality, and $\underline{\boldsymbol{u}}, \overline{\boldsymbol{u}} \in \mathbb{R}^6$ represent the respective lower and upper bounds. The model uncertainties $\boldsymbol{\xi}$ include parametric and structural uncertainties and exist in both internal dynamics and external disturbances. The sources of model uncertainties may include inaccurate estimation of hydrodynamics coefficients, control latency from unmodeled thruster dynamics, and so on. In this work, we consider discretization of the continuous-time system in (1) modeled by

$$\boldsymbol{x}_{t+1} = f\left(\boldsymbol{x}_t, \boldsymbol{u}_t\right), \qquad (2)$$

which describes the evolution from time t to $t + 1$ of the state $\boldsymbol{x}$, given the action $\boldsymbol{u}$. Also notice that the parameters of the internal dynamics are assumed fixed but unknown to the learning algorithms.

B. Control Objective

In this control problem, the model of the dynamics f is unknown to the controller. 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 (2). The objective function denoted by J is the discounted sum of rewards r, 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:

$$J\left(\boldsymbol{x}_0\right) = \sum_{t=0}^{T-1} \gamma^t r\left(\boldsymbol{x}_t, \boldsymbol{u}_t\right), \qquad (3)$$

where the reward function is given as:

$$r\left(\boldsymbol{x}_t, \boldsymbol{u}_t\right) = -\boldsymbol{x}_t^T \boldsymbol{Q} \boldsymbol{x}_t - \boldsymbol{u}_t^T \boldsymbol{R} \boldsymbol{u}_t, \qquad (4)$$

with $\boldsymbol{Q} \in \mathbb{R}^{n_x \times n_x}$ and $\boldsymbol{R} \in \mathbb{R}^{n_u \times n_u}$ being weight matrices, and $\gamma \in [0, 1]$ is a discount factor that prioritizes near-term rewards. The trajectory is implicitly represented using only the controls $\boldsymbol{U}$. The state sequence $\boldsymbol{X}$ is recovered by interaction with the environment (2) from the initial state $\boldsymbol{x}_0$.

In this work, RL is used to optimize this control objective through a trial-and-error approach. 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 $\boldsymbol{u}_t$ chosen from some policy π under a dynamics model f, meanwhile collecting a scalar reward r_t according to a reward function r. 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 objective function J:

$$\pi^* = \underset{\pi}{\operatorname{argmax}} J\left(\boldsymbol{x}_0\right). \qquad (5)$$

C. Transfer Learning

The key idea of transfer learning [36] is that experience or knowledge obtained from learning to specialize in one task can help improve learning effectiveness in a different but related task. The former is called source task, the latter is called target task. 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 an underwater robot developed from the fundamental principles of dynamics, this model corresponds to the dynamics function in (1) excluding the model uncertainties term ξ, and is referred to as "first-principle model". The target task applies a data-driven model of an underwater robot derived from real-world experimental data, this model contains various uncertainties in the real system, and is referred to as "empirical model". 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 internal dynamics; the other part is force data of real-world ocean waves collected in open water, representing external disturbances. These two models differ in both internal dynamics and 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 the control policy.

In this paper, the policy directly trained on the source task is defined as "source policy"; the policy directly trained on the target task is defined as "target policy"; the policy trained on the source task then directly applied on the target task without transfer learning is defined as "unadapted policy"; the policy trained on the source task then further adapted on the target task using transfer learning is defined as "adapted policy".

There are many metrics to measure the benefits of transfer. In this work, we use jumpstart and learning time. 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.

IV. MODULAR NETWORK DESIGN

Previous work [32] has demonstrated that RNN can directly learn to control a dynamical system with unobservable disturbances in an end-to-end mode, where the past motion history is mapped to the control action. While, inspired by [33], this work applies modular learning procedures, that explicitly decouple the process into disturbance identification and motion control.

The learning algorithm for excessive disturbance rejection control proposed in this paper is composed of two main modules, namely a Generalized Control Policy (GCP) and an Online Disturbance Identification Model (ODI), as
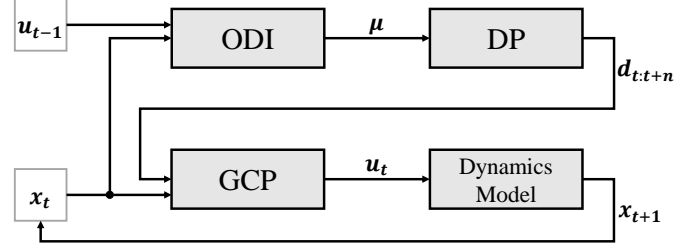

Fig. 2. Diagram of GCP-ODI. The Online Disturbance Identification Model (ODI) employs RNN to identify the disturbance parameters $\mu = \{A_1, \omega_1, \phi_1, \cdots, A_k, \omega_k, \phi_k\}$ from the past states and actions, and the Disturbance Prediction Model (DP) formulates a sequence of future 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 $\boldsymbol{u}_t$.

shown in Fig. 2. Firstly, we build a RL framework to train GCP, $\boldsymbol{u}_t = \pi(\boldsymbol{x}_t, \boldsymbol{d}_{t:t+n})$, under a system dynamics model, $\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t; \boldsymbol{\mu})$, parameterized using the disturbance parameters $\boldsymbol{\mu}$. Unlike classical RL policies where a mapping between states and controls is established, GCP explicitly 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, and is shown to improve the control performance in Section VI. 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.

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 hidden state vector $\boldsymbol{h}_{t-1}$ from the previous timestep 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 (Fig. 2).

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 the target task, the identification network 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.

A. 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

Algorithm 1 Learning Generalized Control Policy

1: Randomly initialize policy network π
2: Initialize rollout buffer R
3: Initialize episode step counter $t = 0$
4: Sample state $\boldsymbol{x}_t \sim p(\boldsymbol{x}_0)$
5: Sample disturbance parameters $\boldsymbol{\mu} \sim p(\boldsymbol{\mu})$
6: Initialize disturbance sequence $\boldsymbol{d}_{t:t+n}$ from $\boldsymbol{\mu}$
7: Insert $(\boldsymbol{x}_t, \boldsymbol{d}_{t:t+n})$ into R
8: **while** $step \leq MaxStep$ **do**
9:     **while** R is not full **do**
10:         Calculate disturbances $\boldsymbol{d}_t$ from $\boldsymbol{\mu}$
11:         Perform action $\boldsymbol{u}_t = \pi(\boldsymbol{x}_t, \boldsymbol{d}_{t:t+n})$
12:         Receive next state $\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t; \boldsymbol{\mu})$
13:         Receive reward $r_t = r(\boldsymbol{x}_t, \boldsymbol{u}_t)$
14:         Update $\boldsymbol{d}_{t+1:t+1+n}$
15:         Insert $(\boldsymbol{x}_{t+1}, \boldsymbol{d}_{t+1:t+1+n}, \boldsymbol{u}_t, r_t)$ into R
16:         Update $t \leftarrow t + 1$
17:         **if** episode is terminated **then**
18:             Reset $t = 0$
19:             Sample $\boldsymbol{x}_t \sim p(\boldsymbol{x}_0)$
20:             Sample $\boldsymbol{\mu} \sim p(\boldsymbol{\mu})$
21:         **end if**
22:     **end while**
23:     Update π using data in R
24: **end while**

Algorithm 2 Learning Online Disturbance Identification Model

1: Randomly initialize ODI netowrk ψ
2: Initialize training buffer B
3: Initialize iteration counter $iter = 1$
4: **while** ψ is not converged **do**
5:     **for** $i = 1 : K$ **do**
6:         Sample $\bar{\boldsymbol{\mu}} \sim p(\boldsymbol{\mu})$
7:         **for** $j = 1 : N$ **do**
8:             Sample $\boldsymbol{x}_0 \sim p(\boldsymbol{x}_0)$
9:             **for** $t = 0 : T - 1$ **do**
10:                 Calculate $\bar{\boldsymbol{d}}_t$ from $\bar{\boldsymbol{\mu}}$
11:                 **if** $iter == 1$ **then**
12:                     Calculate $\bar{\boldsymbol{d}}_{t:t+n}$ from $\bar{\boldsymbol{\mu}}$
13:                     Perform $\boldsymbol{u}_t = \pi(\boldsymbol{x}_t, \bar{\boldsymbol{d}}_{t:t+n})$
14:                 **else if** $iter > 1$ **then**
15:                     Predict $(\hat{\boldsymbol{\mu}}, \boldsymbol{h}_t) = \psi(\boldsymbol{x}_t, \boldsymbol{u}_{t-1}, \boldsymbol{h}_{t-1})$
16:                     Calculate $\hat{\boldsymbol{d}}_{t:t+n}$ from $\hat{\boldsymbol{\mu}}$
17:                     Perform $\boldsymbol{u}_t = \pi(\boldsymbol{x}_t, \hat{\boldsymbol{d}}_{t:t+n})$
18:                 **end if**
19:                 Receive $\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t; \bar{\boldsymbol{\mu}})$
20:             **end for**
21:             Form trajectory $\boldsymbol{\tau} = (\boldsymbol{u}_0, \boldsymbol{x}_1, \cdots, \boldsymbol{u}_{T-1}, \boldsymbol{x}_T)$
22:             Store $(\boldsymbol{\tau}, \bar{\boldsymbol{\mu}})$ in B
23:         **end for**
24:     **end for**
25:     Optimize ψ using data in B
26:     Update $iter \leftarrow iter + 1$
27: **end while**

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 disturbances usually leads to poor generalization.

In this work, we found that it is possible 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 $\boldsymbol{\mu}$, its performance can be comparable with the policies trained for a specific set of waveforms based on state input only. 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 VI). The reason behind this phenomenon is that the frequency domain signals only provide a description of the waveforms, but do not explicitly indicate phase information, so that the algorithm cannot figure out where it is along the waveforms 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 $\boldsymbol{d}_t = A_1 \sin(\omega_1 t + \phi_1) + \cdots + A_k \sin(\omega_k t + \phi_k)$. Compared with using the frequency domain signals, the time domain signals do not contain complete information of the disturbances. But they directly give the disturbance waveforms in the near future, which are more

closely related to control. Thus, they are still proved to achieve better performance in Section VI.

Advantage Actor Critic (A2C) [23] is used to train GCP. During each episode, the algorithm (Algorithm 1) samples disturbance parameters $\boldsymbol{\mu}$ 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 $f(\boldsymbol{x}_t, \boldsymbol{u}_t; \boldsymbol{\mu})$. Once the trajectory data are collected, the policy network π will be updated following A2C rules.

B. 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), $\boldsymbol{\mu} = \psi(\boldsymbol{x}_t, \boldsymbol{u}_{t-1}, \boldsymbol{h}_{t-1})$, 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 $(\boldsymbol{x}_t, \boldsymbol{u}_{t-1})$ and the hidden state $\boldsymbol{h}_{t-1}$ from the previous timestep. The optimization is conducted through minimizing the objective function using Stochastic Gradient

Descent (SGD):

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

(6)

where $\boldsymbol{\tau} = \{\boldsymbol{x}_0, \boldsymbol{u}_0, \cdots, \boldsymbol{x}_{T-1}, \boldsymbol{u}_{T-1}, \boldsymbol{x}_T\}$ 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 $\bar{\boldsymbol{\mu}}$, with the policy $\pi\left(\boldsymbol{x}_t, \bar{\boldsymbol{d}}_{t:t+n}\right)$ and the dynamics $f\left(\boldsymbol{x}_t, \boldsymbol{u}_t; \bar{\boldsymbol{\mu}}\right)$. The trajectory data is then stored in the training buffer B and used to update ODI network ψ using (6) (Algorithm 2).

After optimizing ODI ψ using (6), 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 VI). 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 $f\left(\boldsymbol{x}_t, \boldsymbol{u}_t; \bar{\boldsymbol{\mu}}\right)$.

In order to solve this issue, one possible idea is to iteratively train ODI by using mismatched disturbance parameters $(\hat{\boldsymbol{\mu}} \neq \bar{\boldsymbol{\mu}})$ for the control policy $\pi(\boldsymbol{x}_t, \hat{\boldsymbol{d}}_{t:t+n})$ and the system dynamics $f\left(\boldsymbol{x}_t, \boldsymbol{u}_t; \bar{\boldsymbol{\mu}}\right)$. In each iteration, more training samples are generated following the previous procedures, but the disturbances used by GCP now come from the predicted disturbance 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 (6). After a small number of iterations, the combined system, GCP-ODI, achieves close performance with GCP that is fed with the true disturbance parameters $\bar{\boldsymbol{\mu}}$.

V. Transfer Learning Algorithm

Previous work on transfer learning in RL [57], [59], [65] generally consider model discrepancies as a whole. Then in the problem of excessive disturbance rejection, the model mismatch in both external disturbances and internal dynamics will be combined together and treated equally by the existing transfer learning algorithms, without taking full advantage of the characteristics of the underwater disturbances. While in this work, the external disturbances are separated from the internal 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 is shown to be sufficiently small in Section VI, is then merged into the mismatch of
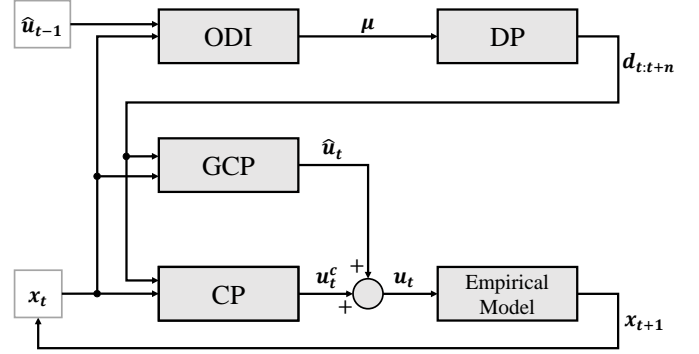

Fig. 3. Diagram of transfer reinforcement learning using Control Action Compensation (CAC).

the internal dynamics and adapted together by the transfer learning. Such framework is expected to reduce the amount of total model mismatch that the transfer process needs to adapt, thus improves the learning efficiency during transfer.

This section discusses three transfer RL algorithms, Control Action Compensation (CAC), Transition Mismatch Compensation with Control Combination (TMC-control) and with Feature Combination (TMC-feature), based on the modular architecture of GCP-ODI proposed in Section IV. All of them follow a basic idea of training an additional policy to compensate the dynamics model mismatch between the source and target tasks, as shown in Fig. 1.

A. Control Action Compensation

We first propose a transfer RL algorithm using Control Action Compensation (CAC), as shown in Fig. 3. This algorithm learns an additional control policy, which generates a compensatory control action $\boldsymbol{u}_t^c$ directly added to the control input computed by the source policy (i.e. GCP) $\hat{\boldsymbol{u}}_t$, 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 (4), which establishes similar optimization goals for both the source policy and the compensatory policy. In order to ensure that ODI remains effective, the action fed into ODI should be the output of the source policy $\hat{\boldsymbol{u}}_t$, that is what ODI has seen during training. However, through transfer learning using (4), CAC algorithm could not optimize the compensatory policy to a satisfactory performance. This is because, although ODI is fed with the action of the source 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 source policy under the first-principle model in the source task. Thus, the differently distributed trajectories fed to ODI lead to worse performance of the whole system.

B. Transition Mismatch Compensation

To address the issue of trajectory mismatch, we develop another transfer RL algorithm based on Transition Mismatch Compensation (TMC), as shown in Fig. 4. This algorithm learns a compensatory policy from the difference between
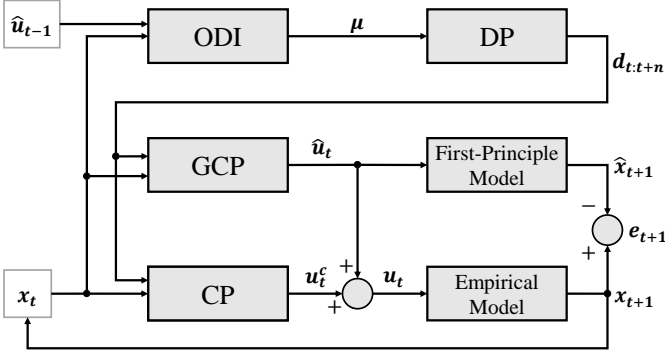
Fig. 4. Diagram of transfer reinforcement learning using Transition Mismatch Compensation with Control Combination (TMC-control).
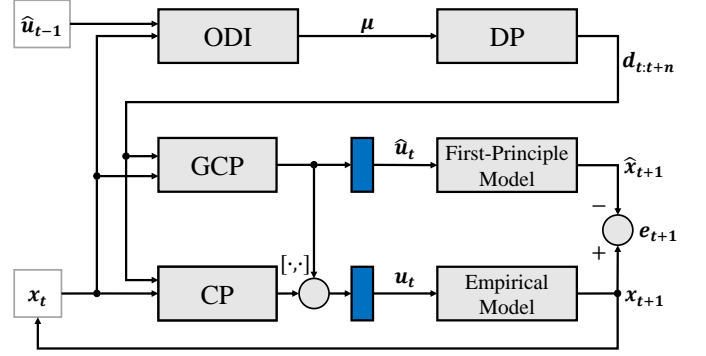

Fig. 5. Diagram of transfer reinforcement learning using Transition Mismatch Compensation with Feature Combination (TMC-feature), $[\cdot, \cdot]$ represents feature combination, the blue blocks represent the final layers of two policy networks.

transitions of the first-principle model and the empirical model, $e_t = \boldsymbol{x}_t - \hat{\boldsymbol{x}}_t$. In this case, the compensatory policy uses a different optimization goal where the reward is given by

$$
\begin{aligned}
r_m\left(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{x}_{t+1}\right) &= r\left(\boldsymbol{x}_t, \boldsymbol{u}_t\right) - \|e_{t+1}\|_2 \\
&= -\boldsymbol{x}_t^T \boldsymbol{Q} \boldsymbol{x}_t - \boldsymbol{u}_t^T \boldsymbol{R} \boldsymbol{u}_t - \|\boldsymbol{x}_{t+1} - \hat{\boldsymbol{x}}_{t+1}\|_2,
\end{aligned}
\tag{7}
$$

which does not divert the source policy from reaching its objective. The purpose of such optimization setting is to eliminate the transition mismatch e by forcing the empirical model to behave like the first-principle 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 first-principle 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 first-principle 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 distributions of simulated parameters, thus the total model mismatch required to be compensated is reduced.

However, these optimization effects are achievable 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 source 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 we believe the transfer learning has not reached its performance limit, and we may need a more efficient way to apply the control compensation.

We then propose TMC with feature combination (see Fig. 5) to further improve TMC framework. Similar with TMC-control, the optimization of TMC-feature still applies the reward function (7) 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, TMC-feature combines features before the last layer of the two policy networks, then feeds the combined

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

VI. EXPERIMENTS

A. Construction of Empirical Model

We are interested in deploying an underwater robot in shallow water where there are excessive wave forces. But it can be difficult to implement an accurate localization system in open water. Thus, we build a simulator based on an empirical model. Specifically, we collect motion data of an underwater robot, named Submerged Pile Inspection Robot (SPIR) (as shown in Fig. 6), in a water tank and train a deep neural network to represent the internal dynamics of the real robot (introduced in Section VI-A). We also collect real-world wave
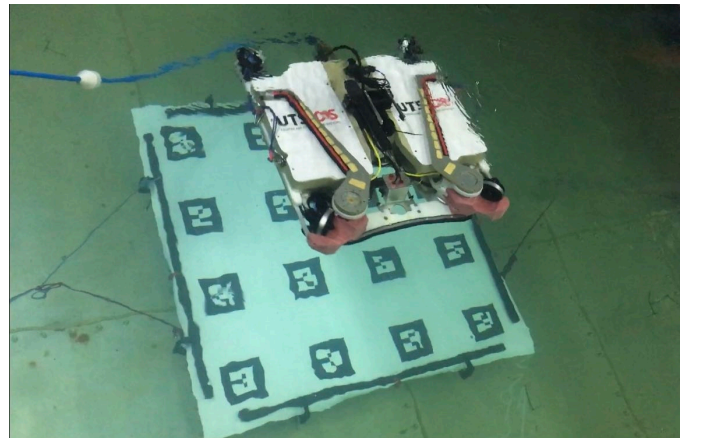

Fig. 6. Tank test of Submerged Pile Inspection Robot (SPIR) over a set of QR code markers in a water tank.

TABLE I
DISTRIBUTIONS OF SIMULATED DISTURBANCE PARAMETERS USED IN THE SOURCE TASK.

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

Fig. 7. Example of simulated disturbances in X, Y and yaw directions.

Fig. 8. Example of real-world disturbances in X, Y and yaw directions.

forces in open water to represent the external disturbances (introduced in Section VI-B). The simulator is then used to define the target task.

We first build a visual localization system through using a downward-looking camera mounted to the underwater robot SPIR and a set of QR code markers fixed on the bottom of the water tank. The localization algorithm is able to calculate the relative pose of the robot with respect to a global frame defined by the markers. Fig. 6 provides a demonstration of the tank test when SPIR localizes itself over the markers in the water tank. Inertial Measurement Unit (IMU) data is also recorded. SPIR is controlled by 12 thrusters, the control data of each thruster is directly read from the onboard computer in the form of Pulse-Width Modulation (PWM) signal. During data collection, the robot moves in a fully autonomous mode and executes random control actions at each timestep. 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. The online data collection provides us with the raw sensor data expressed in their own frame. We then employ Extended Kalman Filter (EKF) [66] to produce a more accurate estimation of the full state $\boldsymbol{x}$ of SPIR from these raw sensor data.

Normally, a transition function (i.e. dynamics model) would take the current state $\boldsymbol{x}_t$ and control $\boldsymbol{u}_t$ as input, and output the predicted next state $\boldsymbol{x}_{t+1}$. However, when there is a very short time interval Δt between two consecutive states, the predicted next state $\boldsymbol{x}_{t+1}$ will become too similar with the current state $\boldsymbol{x}_t$, and the state difference may not well indicate the underlying dynamics [56]. Thus, the transition function might be difficult to learn. This problem can be solved by learning a transition function that predicts the state change over one timestep Δt. Then the predicted next state is now given by: $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + f_{nn}(\boldsymbol{x}_t, \boldsymbol{u}_t; \theta_f)$.

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. After data preprocessing, we train the dynamics model $f_{nn}(\boldsymbol{x}_t, \boldsymbol{u}_t; \theta_f)$ offline using supervised learning by minimizing the error:

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

(8)

using SGD. While training on the training dataset $\mathcal{D}_{train}$, we also calculate the mean squared error in (8) on the validation dataset $\mathcal{D}_{val}$ to optimize hyperparameters.

B. Experimental Setup

The performance of the designed algorithms is tested through a pose regulation task in simulation, where the robot starts with random pose and velocity in each episode, the goal is to control the robot to navigate toward a target pose then stabilize itself under the external disturbances. 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 in the source task is 16.67 hours.

Normally, an underwater robot has 6 degree of freedom (DOF). In this experiment, only the horizontal motion and control (X, Y and yaw) of the robot are considered, then the robot has a 6-dimensional state space $\mathcal{X} \in \mathcal{R}^6$ and a 3-dimensional action space $\mathcal{U} \in \mathcal{R}^3$. In the source task, we build a first-principle model of SPIR that enables large scale training in simulation. The model is assumed to has the mass
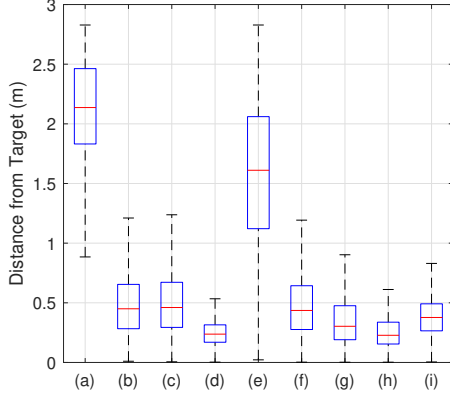
Fig. 9. Distribution of the distance from the robot to the target during last 100 timesteps under the first-principle model: (a) A2C; (b) GCP-true using as input the frequency domain signals $\{A_1, \omega_1, \phi_1, \cdots, A_k, \omega_k, \phi_k\}$; (c) GCP-true using as input the processed frequency 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.

Fig. 10. Trajectories of the robot using A2C and GCP-ODI under the first-principle model.

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. The control constraints are given as $|\overline{\boldsymbol{u}}| = |\underline{\boldsymbol{u}}| = [112N \ 112N \ 82Nm]^T$.

Besides, the source task uses simulated disturbances. They are exerted on all 3 directions (X, Y and yaw) of the robot in the global frame, and are constructed as a superposition of multiple sinusoidal waves with different amplitudes A, frequencies ω and phases ϕ. In this work, we use a composition of 5 sinusoidal waves, whose parameter distributions and waveforms are given in Table I and Fig. 7. Our goal is to enable the trained control policy to deal with unknown time-correlated 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 simulated disturbances are randomly sampled from the given distributions in each training episode.

The target task uses an empirical model of SPIR, composed of the neural network dynamics in Section VI-A and real-world disturbances. These disturbances come from force data of real-world ocean waves collected in open water, as shown in Fig. 8. The data collection process is implemented through connecting a force/torque sensor between SPIR 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. We notice that the real-world disturbances have widely varying amplitudes, which are not constrained within the ranges of the simulated disturbance parameters, leading to a more challenging control problem.

C. Modular Network

The performance of the modular learning architecture, GCP-ODI, is evaluated first for excessive disturbance rejection
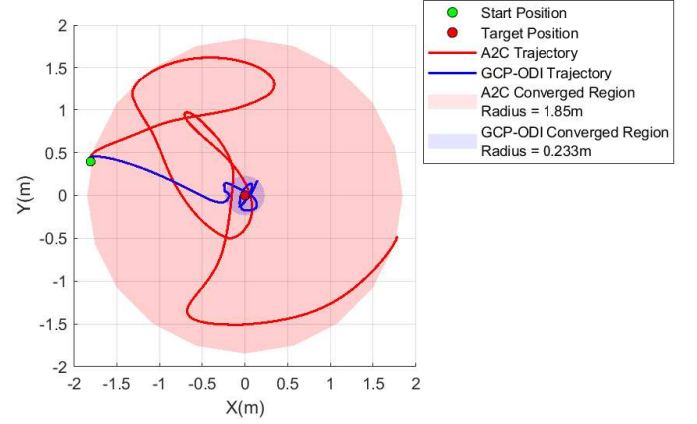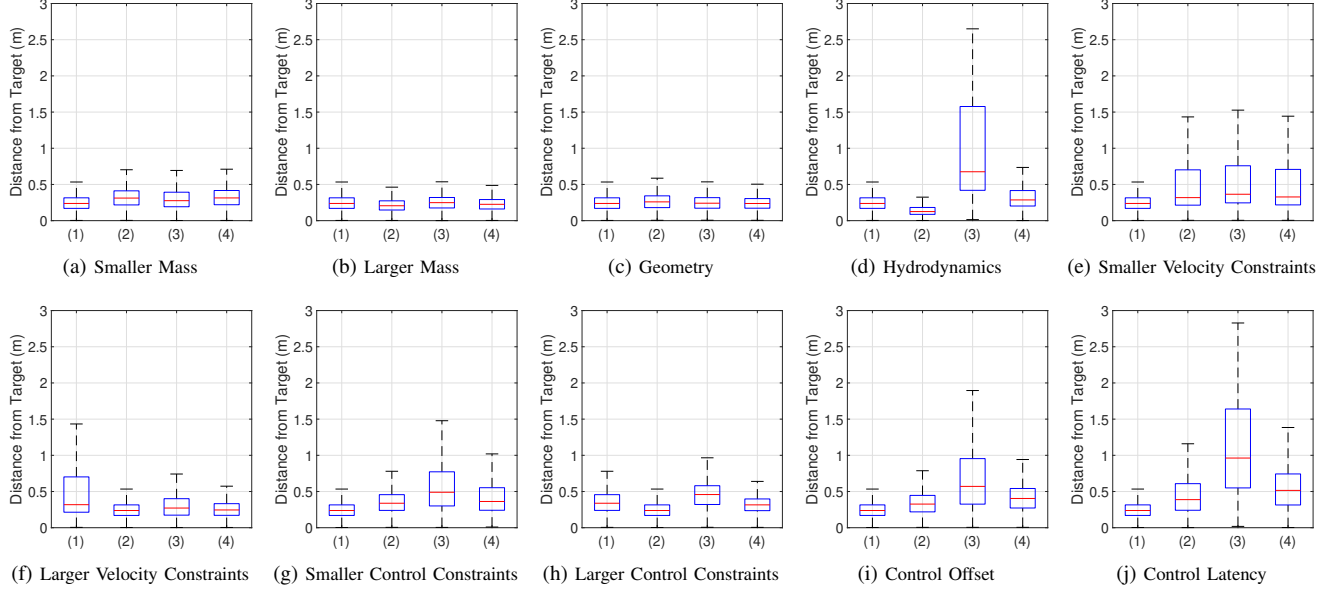
under the first-principle model. As shown in Fig. 9, it can be seen that training a RL policy with disturbances $\boldsymbol{d}_{t:t+n}$ or disturbance parameters $\boldsymbol{\mu}$ as additional input can achieve better performance, compared with conventional RL policy (A2C). In terms of the part of policy input representing disturbances, we found that using time domain signals outperforms using frequency domain signals, even though time is combined with the frequency domain parameters to give phase information $\boldsymbol{\mu} = \{A_1, \omega_1, \omega_1 t + \phi_1, \cdots, A_k, \omega_k, \omega_k t + \phi_k\}$. This result indicates that it might be difficult for RL to find an effective mapping between the frequency domain signals of the disturbances and the control output. In contrast, the time domain signals are more closely related to control. Thus, although the time domain variables only contain partial information of the disturbance waveforms (i.e. only n timesteps of future disturbances), adopting these variables still perform well.

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, even approaching the performance of GCP when given the true disturbance parameters $\bar{\boldsymbol{\mu}}$. In addition, we also evaluate an end-to-end learning framework for excessive disturbance rejection, called Disturbance Observer Network (DOB-Net) [32], as a comparison with the modular architecture of GCP-ODI. 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 (4th iteration). 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

TABLE II
VARIATIONS IN THE FIRST-PRINCIPLE MODEL.

Parameters	Original Model	Variated Model
Mass	$60kg$	$50kg$
	$60kg$	$70kg$
Geometry	Cuboid Geometry: $0.68 \times 0.75 \times 0.19m^3$	CAD Geometry
Hydrodynamics	No Added Mass or Damping	Added Mass and Damping
Velocity Constraints	$\pm \begin{bmatrix} 1.0m/s & 1.0m/s & \frac{\pi}{2}rad/s \end{bmatrix}$	$\pm \begin{bmatrix} 0.7m/s & 0.7m/s & \frac{\pi}{3}rad/s \end{bmatrix}$
	$\pm \begin{bmatrix} 0.7m/s & 0.7m/s & \frac{\pi}{3}rad/s \end{bmatrix}$	$\pm \begin{bmatrix} 1.0m/s & 1.0m/s & \frac{\pi}{2}rad/s \end{bmatrix}$
Control Constraints	$\pm \begin{bmatrix} 112N & 112N & 82Nm \end{bmatrix}$	$\pm \begin{bmatrix} 86N & 86N & 62Nm \end{bmatrix}$
	$\pm \begin{bmatrix} 86N & 86N & 62Nm \end{bmatrix}$	$\pm \begin{bmatrix} 112N & 112N & 82Nm \end{bmatrix}$
Control Offset	None	30% of Control Constraints $|\overline{u}|$
Control Latency	None	$100ms$

Fig. 11. Distribution of the distance from the robot to the target during last 100 timesteps for different variations in the first-principle model: (1) GCP-ODI trained and tested on the original model (i.e. source policy); (2) GCP-ODI trained and tested on the variated model (i.e. target policy); (3) GCP-ODI trained on the original model and tested on the variated model (i.e. unadapted policy); (4) transfer learning using TMC-feature (i.e. adapted policy).

potentially improves the control performance.

Fig. 10 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, referred to as "converged region". It is found that GCP-ODI can dramatically improve the control stability of the robot under the unknown excessive disturbances, the converged region can be reduced from $1.85m$ to $0.233m$.

D. Transfer Learning on Various Model Uncertainties

It has been validated in the previous section that GCP-ODI performs well on the first-principle model, 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 apply the first-principle models but with different model parameters, they are called original model and variated model, 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 II summarizes different model parameters and corresponding variations in the first-principle model. Fig. 11 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 variated model, 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 first-principle model) for transfer. For the variations of velocity constraints, we found that the target policy, the unadapted policy, and the adapted policy
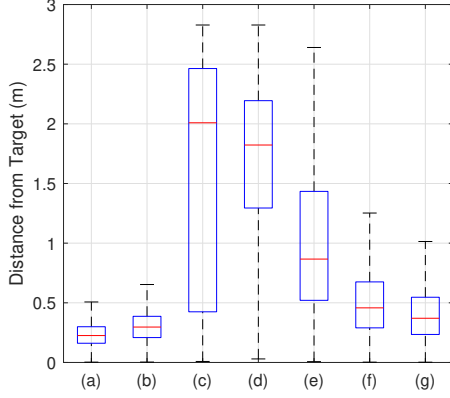
Fig. 12. Distribution of the distance from the robot to the target during last 100 timesteps for the transfer learning: (a) GCP-ODI trained and tested on the first-principle model; (b) GCP-ODI trained and tested on the empirical model; (c) GCP-ODI trained on the first-principle model and tested on the empirical model; (d) transfer learning using CAC; (e) transfer learning using TMC-control; (f) transfer learning using TMC-feature; (g) transfer learning based on DOB-Net.

Fig. 13. Training process of the transfer learning among different algorithms: (a) cumulative task reward; (b) cumulative transition mismatch.

on the variated model 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.

It is possible to model most of the hydrodynamic effects, but still impossible to quantify online, due to the difficulty in estimating coefficients. 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 tethered AUV. 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 $\overline{u}$ and $\underline{u}$ (see Fig. 14).

E. Transfer Learning on Empirical Model

An empirical model has been constructed using real-world experimental data, then we focus on transferring a control policy trained on the first-principle model to be successfully
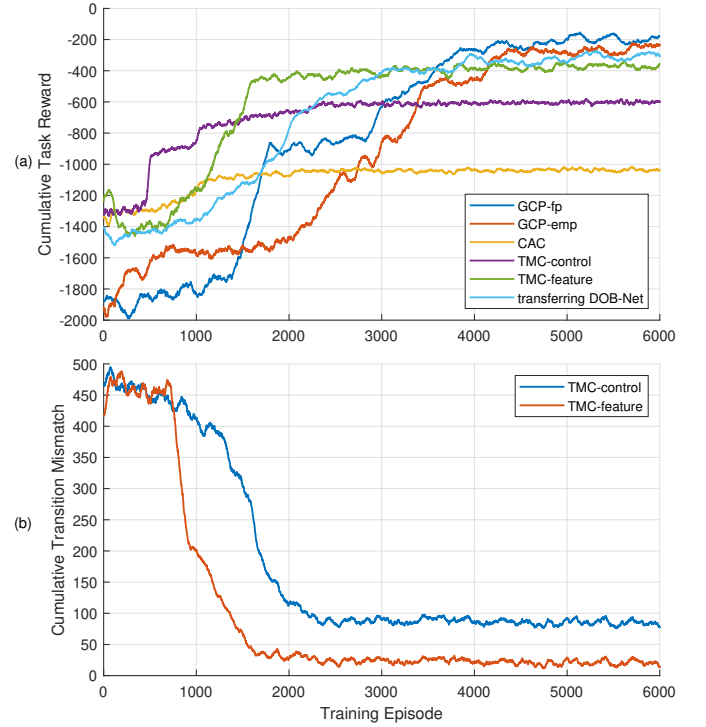
deployed on the empirical model. Note that we provide the results of GCP-ODI trained directly on the empirical model, these results are used as optimal performance in the target task, and can be considered as an informal performance upper bound for the transfer learning algorithms. This policy is available since the empirical model including the real-world disturbances can be deployed in the simulation, but these results are difficult to obtain on a real robot due to high sample complexity and damaging exploratory policy.

Fig. 12 shows the test results of different algorithms, we can see that GCP-ODI has poor stability when trained on the first-principle model then directly deployed on the empirical model. In contrast, GCP-ODI demonstrates much better performance when 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 Fig. 13. We evaluate two objectives, which are cumulative task reward $\mathcal{R} = \sum_{t=1}^{T} r\left(\boldsymbol{x}_t, \boldsymbol{u}_t\right)$ and cumulative transition mismatch $\mathcal{E} = \sum_{t=1}^{T} \|\boldsymbol{e}_t\|_2$ in each training episode. It can be seen that the transfer learning algorithms converge faster than GCP using the true disturbance parameters $\bar{\boldsymbol{\mu}}$ on either the first-principle model or the empirical model (GCP-fp and GCP-emp), not to mention GCP is also followed by the iteratively training of ODI. Thus, the three algorithms of transfer learning based on GCP-ODI all improve the sample efficiency, compared with training GCP-ODI from scratch without transfer.

Among the three transfer learning algorithms, CAC cannot successfully stabilize the robot, since ODI uses inconsistent trajectory data during transfer and training. TMC-control
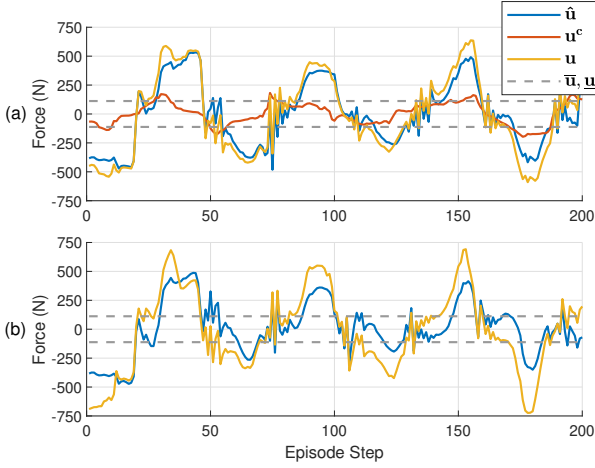
Fig. 14. Comparison among different control signals (X-axis) for the robot during the transfer learning on the empirical model: (a) transfer learning using TMC-control; (b) transfer learning using TMC-feature.

Fig. 15. Trajectories of the robot using GCP-ODI and TMC-feature under the empirical model.

achieves much better performance due to the introduction of a parallel first-principle model, so that the transition mismatch e can be minimized. But this result still cannot be considered as a satisfactory solution. We then look at TMC-feature, which gives a more effective combination approach of the compensatory control signals u^c and the original ones $\hat{u}$, by combining middle layer features of the policy networks. Both Fig. 13 (a) and Fig. 12 prove that TMC-feature achieves better control performance than TMC-control. In addition, through further evaluating the different control components during transfer (see Fig. 14), we found that in TMC-control algorithm, the compensation u^c is relatively small compared to GCP output $\hat{u}$, then adding u^c to $\hat{u}$ will not make much difference. Also, most of the added control actions have been truncated by the control constraints $\overline{u}$ and $\underline{u}$, leading to limited effects of the compensatory control actions. While for TMC-feature algorithm, combining features distinguishes the resultant control actions u from the output of GCP $\hat{u}$, there is even obvious phase advance in u that compensates the control latency. Fig. 15 shows the trajectories of the robot when using TMC-feature algorithm and the direct deployment of GCP-ODI on the empirical model after trained on the first-principle 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 TMC-feature is able to well stabilize the robot, there is still noticeable gap from GCP-ODI directly trained on the empirical model (see Fig. 12). This phenomenon can be explained through Fig. 13 (b), where the cumulative mismatch of TMC-feature cannot be fully eliminated. This result might be caused by the existence of control constraints, which limits the algorithm's ability to 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 $\hat{\mu}$ that best fit the real-world disturbances, which are not known to the learning algorithm and do not follow the distributions
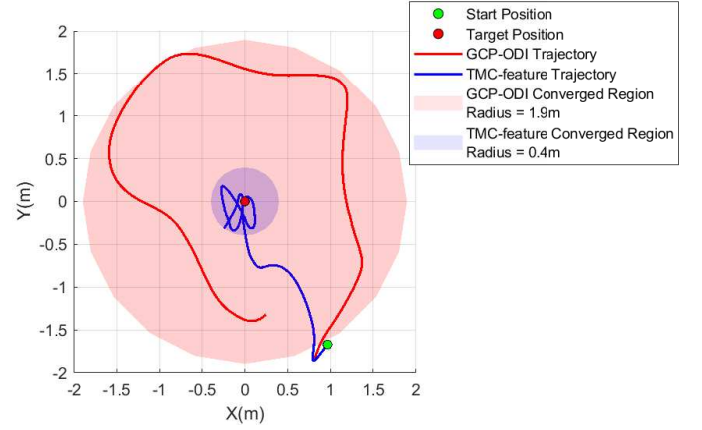
of simulated disturbance parameters. We give a waveform calculated from the disturbance parameters predicted at a timestep in the middle of a trajectory, as shown in Table III, and compare it with the real-world disturbances exerted on the empirical model during test. We can see from Fig. 16 that the predicted disturbances are quite similar to the real-world 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 reduced, then the burden for the compensation is reduced.

In this section, we also evaluate the transfer algorithm based on the end-to-end learning framework for excessive disturbance rejection, i.e. DOB-Net. Similar with TMC, this transfer process also trains an additional policy to compensate the dynamics model mismatch between the source and target tasks. This algorithm is used as a comparison with TMC. Thanks to the ability of ODI to predict the unknown disturbances on the empirical model, the total model mismatch is reduced and the transfer learning is easier to train, compared with transfer learning based on DOB-Net. As illustrated in Fig. 13, the convergence speed of TMC-feature is clearly higher than that of transferring DOB-Net. The final performance of TMC-feature, however, is not as good as the transferred DOB-Net (see Fig. 12), this is reasonable since the transition mismatch 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.

VII. Conclusion & Future Work

This paper proposes a learning framework to address the control problem for excessive disturbance rejection of an underwater robot under dynamics model mismatch. We first introduce a modular architecture of RL, composed of an observer network (ODI) and a controller network (GCP). ODI is used to predict a set of disturbance waveforms based on the observed past states and actions of the system, GCP then is able to produce expected control to actively reject disturbances from the current system state and the predicted disturbance waveforms. Then we develop a transfer RL algorithm, TMC, based on the modular architecture that learns an additional

TABLE III
PREDICTED DISTURBANCE PARAMETERS BY ODI UNDER THE EMPIRICAL MODEL.

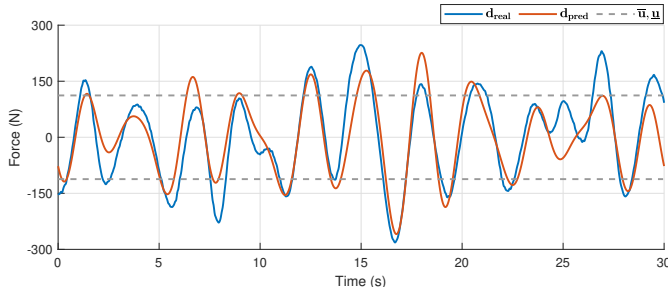| Component Wave	1	2	3	4	5
Amplitude w.r.t. $|\overline{\boldsymbol{u}}|$	34.06%	66.63%	84.59%	53.93%	37.93%
Period (s)	5.98	3.01	2.73	2.31	1.88
Phase (rad)	-0.22π	0.40π	-0.65π	-0.94π	-0.74π

Fig. 16. Comparison between the real-world disturbances and the predicted disturbances by ODI in Y direction, during transfer learning on the empirical model.

compensatory policy through minimizing mismatch of transitions predicted by the two dynamics models of the source and target tasks. And the compensatory policy is added in terms of middle layer features instead of final network outputs in the target task.

The transfer learning algorithms are evaluated on a pose regulation task in simulation, where the source task defines a first-principle 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. As a result, TMC algorithm achieves satisfactory performance on the empirical model after transfer, in the meantime enhance the sample efficiency with respect to learning from scratch without transfer. Furthermore, the modular architecture also outperforms the end-to-end network during transfer in terms of the sample efficiency, since the observer network ODI is able to predict the disturbances in the target task, then the total model mismatch is reduced.

Several aspects can be further explored in the future. It is known that 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.

For both the disturbance rejection control using GCP-ODI and the tranfer learning using TMC, 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.

In the formulation of modular network architecture, GCP takes a fixed length of time domain disturbances 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, the 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.

This research only covers the work using an empirical model in simulation, while the ultimate goal is to deploy the transfer learning on real-world robotic systems. A possible idea is to use professional devices, like wave generators, to generate realistic wave forces in the water tank. Another way is to directly deploy the robot in open water, where an accurate localization system would be necessary. Current solutions mainly focus on Simultaneous Localization And Mapping (SLAM), and the biggest concerns will be the localization accuracy on an unstable platform and the blurred images underwater.

The sample efficiency of the transfer learning directly depends on the mismatch between the first-principle 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.

REFERENCES

[1] G. Antonelli, *Underwater robots*. Springer, 2018, vol. 123.
[2] G. Griffiths, *Technology and applications of autonomous underwater vehicles*. CRC Press, 2002, vol. 2.
[3] R. Dean and R. Dalrymple, "Water wave mechanics for scientists and engineers," *Advanced Series on Ocean Engineering, World Scientific*, vol. 2, 1991.
[4] D. C. Fernández and G. A. Hollinger, "Model predictive control for underwater robots in ocean waves," *IEEE Robotics and Automation letters*, vol. 2, no. 1, pp. 88–95, 2016.
[5] J. Woolfrey, D. Liu, and M. 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*. IEEE, 2016, pp. 4591–4596.

[6] T. Wang, W. Lu, and D. 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)*. IEEE, 2018, pp. 187–193.

[7] L.-L. Xie and L. Guo, "How much uncertainty can be dealt with by feedback?" *IEEE Transactions on Automatic Control*, vol. 45, no. 12, pp. 2203–2217, 2000.

[8] Z. Gao, "On the centrality of disturbance rejection in automatic control," *ISA transactions*, vol. 53, no. 4, pp. 850–857, 2014.

[9] S. Li, J. Yang, W.-H. Chen, and X. Chen, *Disturbance observer-based control: methods and applications*. CRC press, 2014.

[10] W.-H. Chen, D. J. Ballance, P. J. Gawthrop, and J. O'Reilly, "A nonlinear disturbance observer for robotic manipulators," *IEEE Transactions on industrial Electronics*, vol. 47, no. 4, pp. 932–938, 2000.

[11] W.-H. Chen, J. Yang, L. Guo, and S. Li, "Disturbance-observer-based control and related methodsâĂŤan overview," *IEEE Transactions on Industrial Electronics*, vol. 63, no. 2, pp. 1083–1095, 2016.

[12] H. Sun and L. Guo, "Neural network-based dobc for a class of nonlinear systems with unmatched disturbances," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 2, pp. 482–489, 2016.

[13] C. E. Garcia, D. M. Prett, and M. Morari, "Model predictive control: theory and practiceâĂŤa survey," *Automatica*, vol. 25, no. 3, pp. 335–348, 1989.

[14] E. F. Camacho and C. B. Alba, *Model predictive control*. Springer Science & Business Media, 2013.

[15] H. Gao and Y. 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*, vol. 230, no. 1, pp. 3–12, 2016.

[16] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[17] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[18] J. Oh, V. Chockalingam, S. Singh, and H. Lee, "Control of memory, active perception, and action in minecraft," *arXiv preprint arXiv:1605.09128*, 2016.

[19] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep q-learning with model-based acceleration," in *International Conference on Machine Learning*, 2016, pp. 2829–2838.

[20] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International Conference on Machine Learning*, 2015, pp. 1889–1897.

[21] S. Gu, T. Lillicrap, Z. Ghahramani, R. E. Turner, and S. Levine, "Q-prop: Sample-efficient policy gradient with an off-policy critic," *arXiv preprint arXiv:1611.02247*, 2016.

[22] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[23] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.

[24] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[25] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial intelligence*, vol. 101, no. 1-2, pp. 99–134, 1998.

[26] G. Shani, J. Pineau, and R. Kaplow, "A survey of point-based pomdp solvers," *Autonomous Agents and Multi-Agent Systems*, vol. 27, no. 1, pp. 1–51, 2013.

[27] D. Wierstra, A. Förster, J. Peters, and J. Schmidhuber, "Recurrent policy gradients," *Logic Journal of the IGPL*, vol. 18, no. 5, pp. 620–634, 2010.

[28] M. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," in *2015 AAAI Fall Symposium Series*, 2015.

[29] N. Heess, J. J. Hunt, T. P. Lillicrap, and D. Silver, "Memory-based control with recurrent neural networks," *arXiv preprint arXiv:1512.04455*, 2015.

[30] I. Sorokin, A. Seleznev, M. Pavlov, A. Fedorov, and A. Ignateva, "Deep attention recurrent q-network," *arXiv preprint arXiv:1512.01693*, 2015.

[31] T. Wang, W. Lu, and D. Liu, "Excessive disturbance rejection control of autonomous underwater vehicle using reinforcement learning," in *Australasian Conference on Robotics and Automation*, 2018.

[32] T. Wang, W. Lu, Z. Yan, and D. Liu, "Dob-net: Actively rejecting unknown excessive time-varying disturbances," *arXiv preprint arXiv:1907.04514*, 2019.

[33] W. Yu, C. K. Liu, and G. Turk, "Preparing for the unknown: Learning a universal policy with online system identification," in *Proceedings of Robotics: Science and Systems*, Cambridge, Massachusetts, July 2017.

[34] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th annual international conference on machine learning*, 2009, pp. 41–48.

[35] R. Caruana, "Multitask learning," *Machine learning*, vol. 28, no. 1, pp. 41–75, 1997.

[36] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," *Journal of Machine Learning Research*, vol. 10, no. Jul, pp. 1633–1685, 2009.

[37] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 3357–3364.

[38] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive neural networks," *arXiv preprint arXiv:1606.04671*, 2016.

[39] A. A. Rusu, M. Vecerik, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell, "Sim-to-real robot learning from pixels with progressive nets," *arXiv preprint arXiv:1610.04286*, 2016.

[40] E. Tzeng, C. Devin, J. Hoffman, C. Finn, P. Abbeel, S. Levine, K. Saenko, and T. Darrell, "Adapting deep visuomotor representations with weak pairwise constraints," in *Algorithmic Foundations of Robotics XII*. Springer, 2020, pp. 688–703.

[41] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige, *et al.*, "Using simulation and domain adaptation to improve efficiency of deep robotic grasping," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 4243–4250.

[42] L. Lennart, "System identification: theory for the user," *PTR Prentice Hall, Upper Saddle River, NJ*, pp. 1–14, 1999.

[43] F. Giri and E.-W. Bai, *Block-oriented nonlinear system identification*. Springer, 2010, vol. 1.

[44] M. P. Deisenroth, G. Neumann, J. Peters, *et al.*, "A survey on policy search for robotics," *Foundations and Trends® in Robotics*, vol. 2, no. 1–2, pp. 1–142, 2013.

[45] M. Deisenroth and C. 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)*, 2011, pp. 465–472.

[46] L. Kuvayev and R. 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.

[47] J. Forbes and D. Andre, "Representations for learning control policies," in *Proceedings of the ICML-2002 Workshop on Development of Representations*, 2002, pp. 7–14.

[48] T. Hester and P. Stone, "Intrinsically motivated model learning for developing curious robots," *Artificial Intelligence*, vol. 247, pp. 170–186, 2017.

[49] N. K. Jong and P. Stone, "Model-based function approximation in reinforcement learning," in *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*. ACM, 2007, p. 95.

[50] R. S. Sutton, "Dyna, an integrated architecture for learning, planning, and reacting," *ACM SIGART Bulletin*, vol. 2, no. 4, pp. 160–163, 1991.

[51] C. Brauer, "Using eureqa in a stock day-trading application," *Cypress Point Technologies, LLC*, 2012.

[52] M. Schmidt and H. Lipson, "Distilling free-form natural laws from experimental data," *science*, vol. 324, no. 5923, pp. 81–85, 2009.

[53] P. Abbeel and A. Y. Ng, "Exploration and apprenticeship learning in reinforcement learning," in *Proceedings of the 22nd international conference on Machine learning*, 2005, pp. 1–8.

[54] M. Gevers *et al.*, "System identification without lennart ljung: what would have been different?" *Forever Ljung in System Identification, Studentlitteratur AB, Norrtalje*, vol. 2, 2006.

[55] J. C. Bongard and H. Lipson, "Nonlinear system identification using coevolution of models and tests," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 4, pp. 361–384, 2005.

[56] A. Nagabandi, G. Kahn, R. S. Fearing, and S. 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*. IEEE, 2018, pp. 7579–7586.

[57] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1–8.

[58] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, *et al.*, "Learning dexterous in-hand manipulation," *The International Journal of Robotics Research*, vol. 39, no. 1, pp. 3–20, 2020.

[59] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox, "Closing the sim-to-real loop: Adapting simulation randomization with real world experience," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8973–8979.

[60] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," *arXiv preprint arXiv:1804.10332*, 2018.

[61] K. Lowrey, S. Kolev, J. Dao, A. Rajeswaran, and E. 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)*. IEEE, 2018, pp. 35–42.

[62] R. Antonova, S. Cruciani, C. Smith, and D. Kragic, "Reinforcement learning for pivoting task," *arXiv preprint arXiv:1703.00472*, 2017.

[63] A. Rajeswaran, S. Ghotra, B. Ravindran, and S. Levine, "Epopt: Learning robust neural network policies using model ensembles," *arXiv preprint arXiv:1610.01283*, 2016.

[64] S. Chiappa, S. Racaniere, D. Wierstra, and S. Mohamed, "Recurrent environment simulators," *arXiv preprint arXiv:1704.02254*, 2017.

[65] I. Koryakovskiy, M. Kudruss, H. Vallery, R. Babuška, and W. Caarls, "Model-plant mismatch compensation using reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2471–2477, 2018.

[66] P. Zarchan and H. Musoff, *Fundamentals of Kalman filtering: a practical approach*. American Institute of Aeronautics and Astronautics, Inc., 2013.