Learning Context-Aware Neural ODE Dynamics for Adaptive Robotic Control

Shao-Yi Yu[1*], Jen-Wei Wang[1*], Maya Horii[1], Masayoshi Tomizuka[1] and Vikas Garg[2]

***Abstract*—Robotic systems deployed in uncertain and dynamically changing environments often face variations in contact conditions, aerodynamic effects, and external disturbances that challenge reliable control. To remain effective under model-based control, these systems require dynamics models that can adapt to such changes, especially when direct access to complete environmental information is limited. To enable adaptability and facilitate integration with model predictive control, we propose a context-aware dynamics model based on neural ordinary differential equations, which infers environmental factors from state–action histories using a two-phase training procedure. We validate the approach across diverse robotic platforms, including a quadrotor in simulation, as well as a Sphero BOLT robot and a Fanuc manipulator in real-world experiments. The results demonstrate that our method effectively adapts to temporally and spatially varying environmental changes across different tasks. Videos are available here, and the source code is available at our GitHub repository.**

***Index Terms*—Machine Learning for Robot Control, Reinforcement Learning, Adaptive Control, Learning-based MPC**

I. Introduction

ROBOTS are often deployed in uncertain and dynamically changing environments. For example, mobile robots, such as ground vehicles or drones, often need to adapt to varying terrain conditions and wind disturbances, while manipulators may experience changes in payload when transporting items in warehouses. These environmental variations make real-time adaptability essential. However, achieving such adaptability remains challenging for model-based controllers, as they rely on accurate dynamics models for planning over long horizons [1], [2]. Furthermore, many environmental variations cannot be fully detected using onboard sensors, making it important for the system to infer hidden environmental factors from limited data.

Previous efforts in in-context reinforcement learning (RL) have led to major advances in adapting to different environments based on past trajectories [3], [4], [5]. A line of research in adaptive model-free RL proposes specially designed adaptive modules, known as Rapid Motor Adaptation (RMA), to encode environmental information in RL policy [6], [7], [8]. However, model-free RL methods often struggle to explicitly incorporate desired trajectories or constraints, such as collision avoidance, into the policy, which in turn requires large amounts of exploratory data.

Several model-based RL approaches have been proposed [1], [9], [10]; however, they often model dynamics over a predefined discrete time domain, which overlooks the continuously-evolving dynamics of rigid-body robotic systems [11]. Since the dynamics of these systems are typically governed by ordinary differential equations (ODE), neural ordinary differential equations (NODE) [12], which learn (first-order) derivatives and compute system states using numerical integrators, are well-suited for modeling continuous-time dynamics. The approach of modeling derivatives has also shown success in time-series prediction tasks across various domains [13], [14], [15]. In robotics, learning dynamics with NODE has demonstrated robustness to noisy and irregular data in standard RL tasks [16]. However, most NODE-related work either focuses primarily on data efficiency or requires prior system knowledge for successful adaptation. [17], [18].

In this work, we propose a context-aware NODE-based dynamics model that learns full dynamics without prior physics knowledge. To improve training efficiency, we incorporate an RMA-style adaptive module that decouples the learning of environmental conditions from system dynamics, thereby reducing the input dimensions required for training the NODE. As a result, our approach enables efficient adaptation with low computational overhead while retaining the advantages of NODE, including smooth dynamics and reduced compounding errors. This makes the context-aware dynamics model well-suited for integration with model predictive control (MPC) to determine optimal actions. We validate the approach across multiple robotic platforms. Experimental results show that our model outperforms baseline methods on a quadrotor flying through varying wind fields in simulation. We also deploy the model on two real-world systems: a Sphero BOLT navigating across different ground textures and a Fanuc manipulator pushing an object with a changing or unseen center of mass (CoM) during operation. In both cases, our method achieves higher accuracy and robustness than its non-adaptive counterpart. The videos are available here.

II. Related Work

Reliance on Physics-Based Nominal Model. Adaptive dynamics modeling for robotic systems has been explored through a variety of approaches aimed at handling changing and partially observed environments. Classical adaptive

Manuscript received: February, 1, 2026; Revised April, 12, 2026; Accepted May, 11, 2026.

This paper was recommended for publication by Editor Jens Kober upon evaluation of the Associate Editor and Reviewers comments.

*Shao-Yi Yu and Jen-Wei Wang are co-first authors.

Emails: {syyu410, jwwang}@berkeley.edu

[1]Authors are with Department of Mechanical Engineering, University of California, Berkeley, CA, USA

[2]Author is with Aalto University and YaiYai Ltd, Finland

Digital Object Identifier (DOI): see top of this page.

control [19] and online estimation with known features [20] can compensate for disturbances with minimal computational overhead, but their performance is limited by the accuracy of the nominal model and the prior specification of features. Some recent works incorporate NODE variants, such as KNODE [21] and Hamiltonian NODE [18], to model residual dynamics not captured by the nominal model. While more expressive, these approaches still rely on the availability of a known physics model. In contrast, our approach removes this dependency by learning the full system dynamics directly from data, without requiring prior physics knowledge.

Test-Time Adaptation vs. Train-Time Adaptation. A major distinction in adaptive modeling lies in whether adaptation occurs during deployment or is amortized into training. Several methods perform test-time adaptation by updating model parameters online. For example, meta-learning approaches obtain expressive nonlinear features offline and adapt linear parameters online [22], while graph-based Koopman embeddings update linear transition matrices via least-squares [23]. These methods are computationally efficient at deployment but are restricted to updating only a linear layer, leading to poor adaptation under highly nonlinear regime shifts such as contact-rich interactions or rapidly changing terrain. Other approaches adapt model weights online via gradient descent [24], [2], [25], improving expressiveness but at the cost of higher computational overhead, potential instability from noisy gradients, and increased risk of overfitting to recent observations. Alternatively, train-time adaptation methods encode adaptability into the model during training to avoid parameter updates at deployment. These include architectures such as transformers that leverage in-context learning from state-action history [26], Neural Process context encoders that incorporate terrain-specific and robot-specific context [27], and multimodal latent mappers that fuse sensory inputs [28]. Additional training objectives, such as joint forward and backward dynamics prediction [9], further improve representation learning. However, these end-to-end approaches may struggle with learning complex mappings due to entangled system and environmental dynamics, as well as high-dimensional histories inputs.

Leveraging Privileged Environmental Information. Recent work demonstrates that incorporating privileged environmental information during training can significantly improve the learning of adaptive representations [29], [5], [30], [31]. By guiding the model to disentangle environment-dependent factors from system dynamics, these methods enable more effective adaptation at deployment. However, many existing approaches do not explicitly integrate this advantage with continuous-time modeling frameworks, limiting their compatibility with certain control strategies.

NODE-Based Dynamics Modeling. A growing line of research explores the use of NODE for robot modeling and control, including applications in continuous forward kinematics for soft robots [17] and dynamics learning for manipulation [32]. These works highlight the benefits of continuous-time representations and improved data efficiency. However, they primarily focus on stationary settings and do not address adaptation to changing environments. While some adaptive extensions using NODE variants exist [21], [18], they typically rely on residual modeling with known physics priors, as discussed earlier. In contrast, our method leverages NODE to learn full system dynamics in an adaptive framework without requiring prior models, while also benefiting from continuous-time structure that integrates naturally with MPC.

Model-Free vs. Model-Based Approaches. Model-free approaches, such as the RMA [6] and DATT [33] frameworks, have been shown to be effective in learning adaptive control policies directly from interaction data. However, they often suffer from poor sample efficiency and limited flexibility at test time. In particular, incorporating new objectives or constraints—such as collision avoidance—typically requires retraining the policy. In contrast, model-based approaches offer greater flexibility by explicitly modeling system dynamics, allowing new cost functions or constraints to be incorporated at test time without retraining the entire framework. This makes model-based methods more suitable for scenarios requiring adaptability not only to changing environments but also to evolving task specifications.

TABLE I
RELATED WORK ON ADAPTIVE STRATEGIES IN ROBOTIC CONTROL.

Paper	Method	NODE	Adaptation	Privileged Info.	Nominal-model-free	Real Test
[6]	Model-free	✗	✓	✓	✓	✓
[33]		✗	✓	✓	✗	✓
[2], [28]	Model-based	✗	✓	✗	✓	✓
[30], [29], [31]		✗	✓	✓	✗	✓
[5]		✗	✓	✓	✓	✓
[17]		✓	✗	NA	NA	✓
[21]		✓	✓	✗	✗	✓
[18]		✓	✓	✓	✗	✗
Ours	**Model-based**	✓	✓	✓	✓	✓

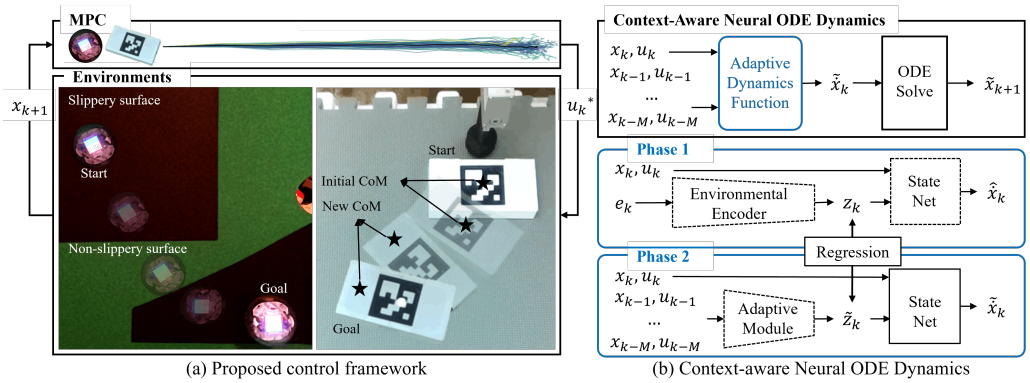
III. PROBLEM STATEMENT

We formulate robot goal reaching and path tracking as a discrete-time MPC problem, which optimizes a sequence of future control actions over a finite time horizon. We consider a dynamical system governed by

$$\begin{aligned} \dot{x}_k &= f(x_k, u_k, e_k), \\ x_{k+1} &= x_k + \int_{t_k}^{t_{k+1}} f(x_k, u_k, e_k)\, dt \end{aligned} \tag{1}$$

The state of the system at time step k is denoted by $x_k \in \mathbb{R}^n$, the control input by $u_k \in \mathbb{R}^m$, and environmental factors (such as ground textures, wind fields, and payloads), which influence the system dynamics, by $e_k \in \mathbb{R}^l$. The function f represents the system dynamics, modeling $\dot{x_k}$. A numerical integrator, such as the *Euler* method or a higher-order *Runge–Kutta* solver, can then be used to obtain x_{k+1}. Given a prediction horizon of length H, the task for this system becomes an optimization problem that can be solved using MPC at each time step k:

$$\begin{aligned} \min_{u_k,\ldots,u_{k+H-1}} \quad & \sum_{i=k}^{k+H-1} \ell(x_i, u_i) + \ell_f(x_{k+H}) \\ \text{s.t.} \quad & x_{i+1} = \text{ODESolve}(x_i, u_i, f, t_i, t_{i+1}), \\ & \qquad \forall i = k, \ldots, k+H-1, \\ & u_i \in \mathcal{U}, \quad \forall i = k, \ldots, k+H-1, \\ & x_0 \in \mathcal{X}_0 \end{aligned} \tag{2}$$



(a) Proposed control framework (b) Context-aware Neural ODE Dynamics

Fig. 1. (a) Proposed control framework for robotic control, where MPC is adopted to determine optimal control actions by predicting future trajectories with our proposed dynamics model. (b) Structure of the context-aware dynamics: State Net models the derivatives of states evolution, the environmental encoder processes privileged information, and the adaptive module reconstructs a latent environmental vector from historical state-action data by regressing on the corresponding latent vector from Phase 1. State prediction is obtained through numerical integration of the dynamics function. Models with trainable weights are indicated with dashed lines.

Here, the stage cost and the terminal cost are denoted by ℓ and ℓ_f, respectively. The input constraint set is denoted by $\mathcal{U}$, and the initial constraint set, designed to ensure that the robot starts from the designated state, is denoted by $\mathcal{X}_0$. After solving the problem (2), only the first control input $u_k^\star$ of the optimal sequence is applied to the system. In the next time step, the horizon is shifted forward, and the optimization is repeated with updated state information. MPC can handle a certain degree of uncertainty or disturbances by solving Equation (2) online during execution. However, it fails to perform well when the system deviates significantly from the reference dynamics model. Therefore, a dynamics model that can mitigate model mismatch is crucial for maintaining control performance.

IV. Context-Aware Neural ODE Dynamics

A. Two-Phase Framework

In this section, we discuss how to learn the dynamics mapping from the current state x_k and current action u_k to the next state x_{k+1} conditioned on historical data. The objective of learning the context-aware dynamics is to capture environments based solely on historical data, so that the dynamics function can be adjusted according to the inferred environment. While most model-based RL methods learn the mapping in an end-to-end manner, we decompose dynamics learning into two phases as shown in Fig. 1(b).

In Phase 1, we use privileged information e_k, which contains direct and complete environmental information, to facilitate learning the state evolution in response to control actions. However, since e_k may not be measurable during deployment, we apply an *environmental encoder* to first transform e_k into a latent vector z_k. We then feed z_k into *State Net*, which is part of the dynamics function f, to obtain $\hat{\dot{x}}_k$, and use a numerical integrator to compute the next state x_{k+1}. This process models trajectories as solutions to a continuous-time differential equation, inherently producing smooth and physically consistent outputs. After completing end-to-end training in Phase 1, Phase 2 addresses the original mapping problem by learning to infer z_k from historical data. Since Phase 1 is dedicated to modeling the temporal evolution of system states, we freeze the weights of the learned State Net in Phase 2. We then regress the historical data on their corresponding z_k and obtain $\tilde{z}_k$. The process can be expressed as:

$$z_k = g(e_k),\ \tilde{z}_k = h(\{(x_i, u_i)\}_{i=k-M}^{k-1}),\ L = L_2(z_k, \tilde{z}_k) \quad (3)$$

State-action history over a horizon of length M is denoted by $\{(x_i, u_i)\}_{i=k-M}^{k-1}$, and regression loss is denoted by L. The environmental encoder, denoted by g, encodes e_k into z_k. The adaptive module, denoted by h, reconstructs key information pertaining to the current environment by encoding the state-action history into $\tilde{z}_k$ and regressing it on z_k using an MSE loss. We treat the environment in latent space because it is easier to align the domain of $\{(x_i, u_i)\}_{i=k-M}^{k-1}$ and its corresponding e_k in another lower-dimensional space, as they carry distinct physical meanings and have significantly different dimensionality.

B. Training Details

To implement MPC, we use the Model Predictive Path Integral (MPPI) framework [34] to obtain optimal control actions. To improve long-horizon prediction accuracy, we apply curriculum learning to train NODE with gradually increasing prediction horizons, from 1-step to H-step alignment. This mitigates the gradient explosion and vanishing issues that commonly arise when training on long sequences or fine-tuning an existing model. In addition, for the adaptive module design, using a 1D convolutional neural network to handle the high dimensionality of historical data shows benefits in quadrotor experiments by improving latent vector reconstruction through the extraction of local temporal patterns. The sliding filters capture environmental changes regardless of their position in the sequence, which is useful in robotic motion, where changes in velocity or motion direction may occur at arbitrary points within a time sequence. Since each element in the state and action vectors represents a distinct physical quantity, treating them as separate channels further enhances feature extraction.

To improve performance across environments, we incorporate online fine-tuning of the learned dynamics model. As

new observations $\{(x_k, u_k, x_{k+1})\}$ become available during execution, we update the parameters in the dynamics model on the fly to reduce prediction errors. To avoid catastrophic forgetting as well as to balance exploration and exploitation, we use experience replay buffers to record all the observations for online learning and enable a robot to select between a random action and u_k^*. This continual learning process allows the model to refine its predictions and adapt to distributional shifts, especially for unseen historical data. Notably, we use the same training and testing time step for NODE throughout the experiments.

V. Quadrotor Navigation in Simulation

Data Collection. We collect datasets using simulators by randomly sampling initial states and applying random actions. Each trajectory consists of 50 state–action pairs. If the dynamics model trained on the dataset fails to accurately capture the true system behavior, the online dynamics learning approach discussed in Section IV-B becomes essential, as it enables the robot to explore and cover critical portions of the state–action space required to accomplish the task.

Setup. We use the quadrotor dynamics as the governing equation in our simulator [33]. The state of the dynamics model is defined as $[p, v, q]^T$, where p, v and q are the 3D position, velocity, and quaternion. As shown in Fig. 2, control action is defined as $[f_\Sigma, \omega]^T$, where f_Σ denotes the desired thrust and ω denotes the desired angular velocity in roll, pitch and yaw directions $[\omega_r, \omega_p, \omega_y]$. While the MPC determines high-level thrust and angular velocity commands, the low-level Proportional-Integral (PI) controller computes the necessary body torques; an inverse actuation matrix then maps this combined thrust and torque into individual motor speeds. To model a real-world quadrotor, we transform the continuous dynamics model into a discrete one with the sampling time set as $0.02\,\mathrm{s}$. Environmental variations are different wind fields acting on the quadrotor system. The wind field is modeled as a disturbance force along the x-direction. During data collection, we sample the disturbance force in the range of $[-1, 1]\,\mathrm{N}$, where N denotes Newtons, and collect trajectories under each wind field.

Target tasks. In the goal reaching and hovering task, the domain is a box with half-size $0.2\,\mathrm{m}$. The quadrotor starts at each vertex of the box, and the goal is located at the center. The objective is to control the quadrotor to reach and hover at the goal. We allow each controller $5\,\mathrm{s}$ to execute actions, then calculate the average position error (RMSE) between the quadrotor position and goal over the final second. In the path tracking task, the quadrotor begins at the same positions as in the goal reaching tasks and follows a circular path, evaluated with the same RMSE metric. Test environments include two wind conditions, spanning both within and beyond the training range. (i) time-varying winds fluctuating sinusoidally around nominal force of $1.5\,\mathrm{N}$ with the amplitude of $0.5\,\mathrm{N}$, updated every 10 control steps; and (ii) a spatially dissipating wind field generated by a fan-like source located at $(-0.2,\ 0,\ 0)$, producing airflow in x direction. The wind magnitude decays with distance in both lateral and longitudinal directions from the source, with a peak force of $2\,\mathrm{N}$. The disturbance is further perturbed by additive Gaussian noise whose variance scales with the local wind magnitude, yielding a random time-varying component resembling Brownian motion [33].

The MPC cost function for a quadrotor performing a goal reaching and hovering or path tracking task is defined as $J = \sum_{k=0}^{H} \left(w_p \left\| p_k - p_k^{\text{ref}} \right\|^2 + w_q \left(1 - \left(q_k^\top q_k^{\text{ref}} \right)^2 \right) \right)$, where p_k^{ref} is the reference (goal or trajectory) position at time k, the reference quaternion at time k is denoted by q_k^{ref}, and the weighting factors for the position and orientation errors are denoted by w_p and w_q, respectively.

MPC performance. We compare our method against the baseline and ablation methods listed in Table II. Table III shows the control performance. We also apply online dynamics learning in a constant wind field, as described in Section IV-B, to fine-tune the dynamics model. From the results, we observe that our approach successfully completes each navigation task under environmental conditions that vary across space and time, achieving lower tracking errors than the model-based baselines. While there is a slight performance drop from Phase 1 to Phase 2, the adaptive module in Phase 2 still reconstructs environmental factors and achieves adaptation. The quadrotor trajectories in the path tracking task (Fig. 2) show that the meta-learning-based and fixed-dynamics models perform best among the baselines, successfully guiding the quadrotor near the target trajectory. However, these top baselines still cannot accurately track the reference, whereas our method achieves the best overall tracking performance. Table III also compares model-free methods (RMA and DATT) with our model-based approach. These two categories use fundamentally different training pipelines: PPO, a model-free and on-policy method, trains the policy directly from data sampled by the current policy, whereas our approach first collects offline data using random policies to train a dynamics model, and then fine-tunes the model using online data generated by the MPPI controller. Because of these differences, it is difficult to compare the methods using exactly the same training dataset. Nevertheless, we report the results in Table III and show that our approach achieves comparable performance.

The results in Fig. 3 demonstrate the generalization capability of our model-based method compared to the model-free baseline, DATT, under increasing distribution shift in wind field with constant force. As the scaling factor α increases beyond the training range $[-1, 1]\,\mathrm{N}$, both goal-reaching and path-tracking errors grow; however, our method exhibits significantly more graceful degradation, particularly in the goal-reaching task. While errors exceeding $0.2\,\mathrm{m}$ are considered failures, DATT fails at around $\alpha \approx 1$ for goal reaching, whereas our method remains robust up to approximately $\alpha \approx 3$. For path tracking, DATT achieves stronger performance—likely because it is explicitly trained for this task—yet both methods remain below the failure threshold even at $\alpha = 3$. This contrast highlights that our approach is not specialized for a single task but instead maintains consistent performance across different objectives. Moreover, the improved robustness can be attributed to the use of learned dynamics from transitions, whereas DATT directly learns a

TABLE II
BASELINES AND ABLATIONS. CADM REFERS TO CONTEXT-AWARE DYNAMICS MODEL [9]. PPO REFERS TO PROXIMAL POLICY OPTIMIZATION [35].

Type	Method	Description
Baselines	Meta-learning dynamics	We adopt the concept of the meta-learning–based approach in [5] to design a context encoder using variational inference.
	CaDM	End-to-end adaptive dynamics with forward and backward losses to extract environmental factors [9].
	RMA	Building on RMA [6], we enable trajectory tracking by embedding desired future positions into the PPO policy [35].
	DATT	Model-free method using L1 adaptation to derive the environmental factors, trained with PPO [33].
Ablation	MLP (Phase 1)	MLP-based dynamics model using privileged information, trained autoregressively with L2 loss over a fixed horizon.
	MLP (Phase 2)	MLP-based dynamics trained under the same two-phase framework using historical information.
	Fixed NODE	NODE-based model without online adaptation, relying only on initial environmental factors during inference.
Proposed	Ours (Phase 1)	NODE-based model trained with privileged information as an upper bound, solved via *Forward Euler* integration.
	Ours (Phase 2)	Context-aware NODE model with an encoder that reconstructs z_k from historical state-action data.

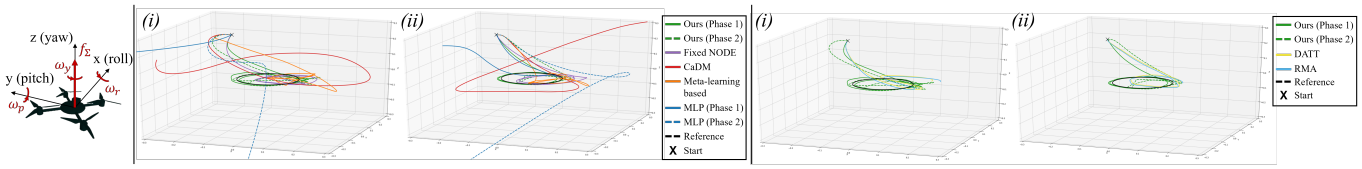


Fig. 2. Path tracking trajectories for the quadrotor under *(i)* time-varying winds, and *(ii)* spatially dissipating wind with random time-varying noise.

TABLE III
PERFORMANCE OF THE QUADROTOR UNDER *(i)* TIME-VARYING WINDS, AND *(ii)* SPATIALLY DISSIPATING WIND WITH RANDOM TIME-VARYING NOISE. RMSE (m).

	i		*ii*	
	Goal reaching	Path tracking	Goal reaching	Path tracking
MLP(Phase 1)	> 0.2	> 0.2	> 0.2	> 0.2
MLP(Phase 2)	> 0.2	> 0.2	> 0.2	> 0.2
CaDM	> 0.2	> 0.2	> 0.2	> 0.2
Meta-learning based	0.092±0.023	0.143±0.074	0.099±0.021	0.119±0.066
Fixed NODE	0.066±0.011	0.055±0.019	0.065±0.008	0.049±0.013
RMA	0.107±0.012	0.099±0.014	0.065±0.015	0.062±0.012
DATT	0.083±0.016	0.070±0.013	0.051±0.008	0.048±0.009
Ours(Phase 1)	**0.010±0.004**	**0.036±0.020**	**0.012±0.007**	0.034±0.012
Ours(Phase 2)	0.031±0.016	0.049±0.026	0.022±0.014	**0.030±0.018**

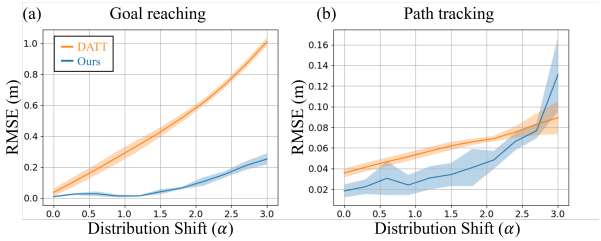


Fig. 3. Performance degradation under distribution shift. (a) Goal-reaching and (b) path-tracking errors are plotted as functions of the distribution shift parameter α. The training regime corresponds to $\alpha \in [0, 1]$, while $\alpha > 1$ represents out-of-range conditions with increasing wind disturbance.

policy, making it more sensitive to distribution shifts beyond the training regime. Notably, our method is trained with $9M$ transition samples to learn the dynamics, whereas DATT requires $25M$ environment interactions to learn the policy, highlighting the better sampling efficiency of our approach. On the other hand, model-free methods offer significantly faster inference: PPO runs at roughly $1\,\mathrm{ms}$ per step—an order of magnitude faster than our $17\,\mathrm{ms}$—making them attractive for applications requiring very high control rates or limited onboard compute. To further evaluate the effectiveness of online learning, we conduct experiments under wind disturbances exceeding $3\,\mathrm{N}$. After 8 hours of offline learning, the policy achieves a position RMSE of $0.1649 \pm 0.0221\,\mathrm{m}$ on path-tracking tasks. With an additional 2 hours of online finetuning, the RMSE is reduced to $0.0646 \pm 0.0153\,\mathrm{m}$.

VI. REAL-WORLD EXPERIMENTS

A. Setup

To demonstrate the real-world applicability of our algorithm, we evaluate the proposed framework on a mobile robot platform and a planar pushing platform, as shown in Fig. 4. The detailed setup and control problem definition for both platforms can be found in Table IV.

Mobile Robot Platform A Sphero BOLT robot is operated on a pool table and a Luxonis Oak-D Pro camera is mounted above the table to detect robot state. This platform evaluates adaptability to different surface textures.

Planar Pushing Platform A Fanuc LR Mate 200iD manipulator with a cylindrical pusher is used to push a box toward a target, while an Intel RealSense D435i camera tracks its pose via an ArUco marker. The robot performs a slow, quasi-static push from the box's long edge, and varying the internal weight shifts the CoM, altering the dynamics in ways that are not observable from the camera, making this a challenging testbed for CoM variation.

B. Data Collection

Mobile Robot Platform Since the low-level controller in the Sphero BOLT robot is embedded in the robot's computational board (unknown to users) and the uncertainty of the robot's dynamics is complicated, it is hard to build a simulation environment that has small sim-to-real gap to the real-world platform. Therefore, we choose to collect data directly on the real-world platform by commanding the robot with random actions from a randomly placed location on the pool table. In sum, we collect around 2,000 pairs of (state, action, next state) for each texture. For privileged information in Phase 1 training, we assign relative friction coefficients to each texture. The friction coefficient is measured by traveling distance and

TABLE IV
CONTROL PROBLEM DEFINITION FOR REAL-WORLD PLATFORMS: MOBILE ROBOT AND PLANAR PUSHING.

	Mobile Robot Platform	**Planar Pushing Platform**
State	$\mathbf{s} = [x, y, dx, dy, \dot{x}, \dot{y}]$: 2D position (x, y), heading direction (dx, dy) as cosine/sine, linear velocity $(\dot{x}, \dot{y})$	$\mathbf{s} = [x, y, \theta]$: object position (x, y), object orientation θ
Action	$\mathbf{a} = [u_{\text{forward}}, u_{\text{turn}}]$: forward velocity command u_{forward}, steering angle command u_{turn}	$\mathbf{a} = [u_{len}, u_{angle}, u_{loc}]$: pushing length u_{len}, pushing direction u_{angle} with zero defined as the direction normal to the pushing surface, pushing location u_{loc} with zero defined as the center of the pushing surface
State Estimation	*Position:* Blob detection on two colored LED patches; robot center = average of patch centers. *Heading:* (dx, dy) the vector from one patch center to another. *Velocity:* Finite-difference estimate of positions over two frames.	*Position and Orientation:* Pose estimation of ArUco marker.
Control Timing	Camera: 60 fps ($\Delta t = 0.0167$ s); Control: 15 Hz (4× decimation); Inference time: < 0.01 s (RTX 3090 GPU)	Camera: 8 fps ($\Delta t = 0.125$ s); Control: N/A (controller will wait for the action to complete before starting a new action); Inference time: < 0.01 s (RTX 3090 GPU)
Environment	Pool table surface with randomized friction layouts (paper patches = low friction; table surface = high friction)	A box with a randomly placed internal weight. CoM denotes the location of the internal weight.

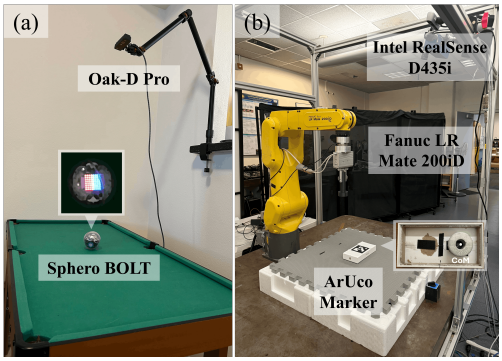


Fig. 4. Real-world setups for (a) Sphero navigation and (b) manipulator pushing tasks.

velocity profile along the path when the robot is applied with the same action on each texture. Since we do not collect data for situations where the robot transitions between paper and pool table textures, we rely on the learned dynamics to generalize to these mixed-friction scenarios.

Planar Pushing Platform Due to the quasi-static nature of the pushing action and high precision of the manipulator, we can build a simulator with minimal sim-to-real gap by doing system identification on real-world pushing trajectories. We build the simulation in MuJoCo and collect 116,000 pairs of (state, action, next state) for each of three different weight placements: centered, offset by $+1.8\,\text{cm}$, and offset by $+3.6\,\text{cm}$. After the Phase 1 model is trained, we combine it with MPC to collect rollout data for training the adaptive module in Phase 2.

C. Results on Mobile Robot Platform

We evaluate the platform on two tasks: goal reaching and path tracking. In the goal reaching task, each layout is tested from a random initial position, with the table center serving as the goal. The robot must reach and hover at the goal, and performance is measured by the minimum distance between its trajectory and the goal. For the path tracking task, we test two reference tracks: a circular path and a triangular path that provides a more aggressive reference. Both reference paths begin from the same initial position. The robot is required to track the reference path, and performance is evaluated using the mean Euclidean distance between the trajectory and the nearest point on the path. We design two cost functions for the tasks. J_1 is for goal reaching and J_2 is for path tracking. The two definitions are expressed as $J_1 = \sum_{k=0}^{H} \left[w_p \left\| p_k - p_k^{\text{ref}} \right\|^2 + w_\theta \left(\theta_k - \theta_k^{\text{pp}} \right)^2 \right]$ and $J_2 = \sum_{k=0}^{H} \left[w_p \left\| p_k - p_k^{\text{ref}} \right\|^2 + w_v \left(v_k - v_k^{\text{ref}} \right)^2 + w_\theta \left(\theta_k - \theta_k^{\text{ref}} \right)^2 \right]$, where $p_k = [x_k, y_k]^T$ denotes the position, with reference p_k^{ref}; $v_k = \|[\dot{x}_k, \dot{y}_k]\|_2$ denotes the velocity with reference v_k^{ref}; and $\theta_k^{\text{pp}} = \arctan 2(y_{\text{goal}} - y_k, x_{\text{goal}} - x_k)$ denotes the pure-pursuit heading toward the goal with reference θ_k^{ref}. The positive scalar weights for position, heading, and velocity errors are denoted by w_p, w_θ, and w_v, respectively.

Table V reports results on two friction layouts, each averaged over three runs from similar start poses. Since the fixed NODE-based dynamics does not adapt to environment changes, it uses the friction coefficient at the start location and treats environmental changes as disturbances. The results show that our model achieves lower errors, indicating effective adaptation to spatially varying friction. It also yields a smaller standard deviation, demonstrating greater robustness and higher repeatability under real-world uncertainty. Our model is deployable on real-world systems and outperforms fixed NODE-based dynamics. Our model also handles surface boundary crossings more reliably, whereas the fixed NODE often gets stuck or loses track. The videos are available here, and the trajectories for path tracking are shown in Fig. 5.

We note that the reported control frequency of $15\,\text{Hz}$ is not limited by the computational cost of the proposed method. In our system, the combined MPC and NODE forward simulation requires less than $0.01\,\text{s}$ per control step, corresponding to a potential control rate exceeding $100\,\text{Hz}$. The primary bottleneck arises from Bluetooth communication latency when transmitting commands to the robot. Therefore, the observed control frequency reflects system-level communication constraints rather than the computational efficiency of the MPC or NODE components.

D. Results on Planar Pushing Platform

The task is to command the manipulator to push the box towards the same goal position from the start pose. The performance is measured by the minimum distance between the rollout trajectory and the goal position. The cost design is the L2 norm of the distance to the goal. Table VI shows the controller performance on different locations of the mass in the box. **Training** refers to the in-distribution CoM ($+3.6\,\text{cm}$ offset) seen during training. **Moderate** refers to the

TABLE V
PERFORMANCE OF REAL-WORLD MOBILE ROBOT NAVIGATION TASKS ACROSS DIFFERENT FRICTION LAYOUTS. RMSE (cm).

	Goal reaching		Circle tracking		Triangle tracking	
	Layout 1	Layout 2	Layout 1	Layout 2	Layout 1	Layout 2
Fixed NODE	2.127 ± 1.233	2.725 ± 1.176	4.567 ± 2.106	2.858 ± 0.744	5.404 ± 0.193	5.124 ± 0.318
Ours	**0.766 ± 0.303**	**0.372 ± 0.082**	**2.077 ± 0.340**	**2.417 ± 0.124**	**3.315 ± 0.027**	**2.681 ± 0.113**

TABLE VI
PERFORMANCE OF REAL-WORLD PLANAR PUSHING TASKS WITH VARYING COM LOCATIONS.

	RMSE (cm)					Computation time (ms)	# of (state, action, next state)
	Training	Moderate	Extreme	Time varying	Disturbance		
Fixed NODE	4.781 ± 0.571	4.587 ± 0.516	**2.285 ± 0.411**	3.368 ± 0.643	4.174 ± 0.740	6.637 ±0.145	**348K**
RMA	0.471 ± 0.120	1.265 ± 0.205	7.072 ± 0.175	**0.491 ± 0.035**	9.548 ± 1.932	**0.398 ± 0.069**	600K
Ours	**0.447 ± 0.256**	**1.129 ± 0.302**	2.408 ± 0.796	1.040 ± 0.314	**1.713 ± 1.918**	7.799 ± 0.236	355K

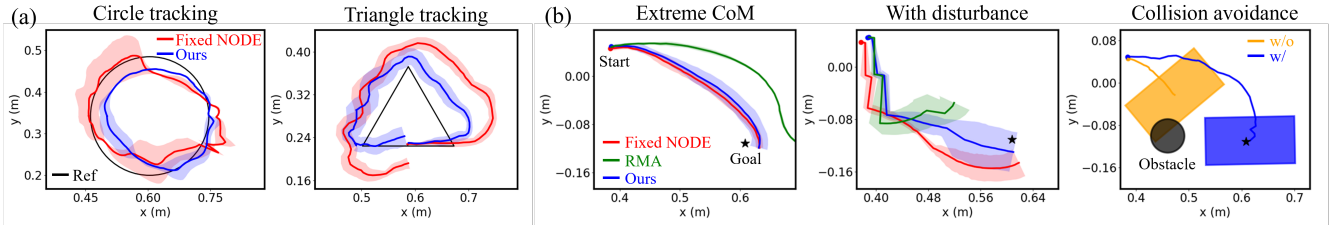


Fig. 5. Trajectories for (a) mobile robot navigation and (b) planar pushing tasks.

configuration ($+6.0\,\mathrm{cm}$ offset) where the inner weight touches the wall of the box and is not seen in the training data. **Extreme** refers to an unseen configuration with a $-2.5\,\mathrm{cm}$ offset on the opposite half of the box. This setting is considered extreme because its dynamic behavior is completely reversed compared to the positive-offset configurations. **Time varying** refers to a dynamic configuration where the weight starts at the center and shifts to a random positive offset after 10 control steps. **Disturbance** refers to the standard $+3.6\,\mathrm{cm}$ offset configuration, but introduces external perturbations by applying a negative y-axis force to the box at control steps 1, 5, and 20. All metrics are obtained from three trials starting from similar initial poses. For the fixed NODE-based approach, we use the learned dynamics assuming the CoM is at the center of the box, which is a reasonable assumption when the interior of the box is not observable.

The results in Table VI highlight the benefit of incorporating adaptation into the learned dynamics model. Compared to the Fixed NODE baseline, our method consistently achieves lower RMSE in most scenarios, demonstrating that online adaptation improves performance under distribution shifts. In the extreme setting, Fixed NODE exhibits comparable performance to our method; this can be attributed to its assumption that the internal weight is centered, which happens to be close to the $-2.5\,\mathrm{cm}$ CoM configuration. In contrast, our adaptation module does not explicitly converge to this exact value, as $-2.5\,\mathrm{cm}$ lies far outside the training distribution. When compared with the model-free RMA approach, our method achieves comparable accuracy in the training and moderate regimes, while significantly outperforming RMA in more challenging scenarios. In particular, our method reduces error by over 60% in the extreme case and over 80% under disturbance, indicating improved robustness to severe distribution shifts. Notably, this performance gain is achieved with approximately 40% fewer training samples, highlighting the superior sample efficiency of our model-based approach. It is important to note that the model-free method relies on carefully engineered reward functions to achieve the performance in Table VI. In our implementation, the reward is defined as $r = -d + \alpha\Delta d + \beta e^{-\gamma d}$, where d is the distance to the goal in world xy plane, Δd denotes the step-wise progress, and $(\alpha, \beta, \gamma) = (5.0, 2.0, 50)$ are weighting coefficients. We observe that using a simpler reward such as $r = -d$ leads to a performance degradation of approximately $3\times$, underscoring the sensitivity of model-free methods to reward design. From the results, we observe that our model-based approach does not require a tedious cost design process to achieve strong goal-reaching accuracy. RMA performs slightly better in the time-varying setting, where the inner weight is randomly placed over time; the sampled weight locations in this experimental setup may be more favorable to RMA, as they are more likely to fall within its training distribution. Additionally, RMA benefits from over $10\times$ faster inference compared to our method. However, our framework offers greater flexibility through the online optimization process. Fig. 5 shows that we enable collision avoidance in real-world deployments without retraining by incorporating an exponential barrier function of the form $c_{\text{obs}} = \lambda e^{-\kappa d_{\text{obs}}}$, where d_{obs} is the distance to the obstacle in world xy plane and (λ, κ) are design parameters, whereas RMA would require additional training to achieve the same capability. The videos are available here, and the trajectories are shown in Fig. 5.

VII. CONCLUSION & FUTURE WORKS

In this paper, we propose a context-aware Neural ODE dynamics model that integrates NODE with a two-phase training

process to reconstruct environmental factors and adjust state predictions accordingly. Combined with MPC, experimental results show that our method achieves superior performance compared to existing dynamics models on quadrotor simulation platforms subject to a range of wind fields. We also demonstrate the effectiveness of applying the framework to real-world systems, including a Sphero BOLT robot and a Fanuc manipulator. A limitation is that performance degrades when training and testing time steps are mismatched, even though larger time steps are desirable in MPC for improved efficiency; this could be addressed in future work by training a time-conditioned NODE.

ACKNOWLEDGMENT

The work was supported by FANUC, the Finnish Center for Artificial Intelligence (FCAI), and CSC – IT Center for Science.

REFERENCES

[1] Y. Seo, K. Lee, I. Clavera Gilaberte, T. Kurutach, J. Shin, and P. Abbeel, "Trajectory-wise multiple choice learning for dynamics generalization in reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 12 968–12 979, 2020.

[2] A. Nagabandi, I. Clavera, S. Liu, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn, "Learning to adapt in dynamic, real-world environments through meta-reinforcement learning," *arXiv preprint arXiv:1803.11347*, 2018.

[3] J. Liang, Z. Wang, Y. Cao, J. Chiun, M. Zhang, and G. A. Sartoretti, "Context-aware deep reinforcement learning for autonomous robotic navigation in unknown area," in *Conference on Robot Learning*. PMLR, 2023, pp. 1425–1436.

[4] X. Zhang, S. Liu, P. Huang, W. J. Han, Y. Lyu, M. Xu, and D. Zhao, "Dynamics as prompts: In-context learning for sim-to-real system identifications," *IEEE Robotics and Automation Letters*, 2025.

[5] S. Belkhale, R. Li, G. Kahn, R. McAllister, R. Calandra, and S. Levine, "Model-based meta-reinforcement learning for flight with suspended payloads," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, p. 1471–1478, Apr. 2021. [Online]. Available: http://dx.doi.org/10.1109/LRA.2021.3057046

[6] A. Kumar, Z. Fu, D. Pathak, and J. Malik, "Rma: Rapid motor adaptation for legged robots," *arXiv preprint arXiv:2107.04034*, 2021.

[7] D. Zhang, A. Loquercio, X. Wu, A. Kumar, J. Malik, and M. W. Mueller, "Learning a single near-hover position controller for vastly different quadcopters," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 1263–1269.

[8] H. Qi, A. Kumar, R. Calandra, Y. Ma, and J. Malik, "In-hand object rotation via rapid motor adaptation," in *Conference on Robot Learning*. PMLR, 2023, pp. 1722–1732.

[9] K. Lee, Y. Seo, S. Lee, H. Lee, and J. Shin, "Context-aware dynamics model for generalization in model-based reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2020, pp. 5757–5766.

[10] B. Evans, A. Thankaraj, and L. Pinto, "Context is everything: Implicit identification for dynamics adaptation," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 2642–2648.

[11] S. Greydanus, M. Dzamba, and J. Yosinski, "Hamiltonian neural networks," *Advances in neural information processing systems*, vol. 32, 2019.

[12] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. Curran Associates, Inc., 2018.

[13] Y. Lipman, R. T. Chen, H. Ben-Hamu, M. Nickel, and M. Le, "Flow matching for generative modeling," *arXiv preprint arXiv:2210.02747*, 2022.

[14] X. N. Zhang, Y. Pu, Y. Kawamura, A. Loza, Y. Bengio, D. Shung, and A. Tong, "Trajectory flow matching with applications to clinical time series modelling," *Advances in Neural Information Processing Systems*, vol. 37, pp. 107 198–107 224, 2024.

[15] M. Cranmer, S. Greydanus, S. Hoyer, P. Battaglia, D. Spergel, and S. Ho, "Lagrangian neural networks," *arXiv preprint arXiv:2003.04630*, 2020.

[16] C. Yildiz, M. Heinonen, and H. Lähdesmäki, "Continuous-time model-based reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2021, pp. 12 009–12 018.

[17] M. Kasaei, K. K. Babarahmati, Z. Li, and M. Khadem, "A data-efficient neural ode framework for optimal control of soft manipulators," in *Proceedings of The 7th Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, J. Tan, M. Toussaint, and K. Darvish, Eds., vol. 229. PMLR, 06–09 Nov 2023, pp. 2700–2713. [Online]. Available: https://proceedings.mlr.press/v229/kasaei23a.html

[18] T. Duong and N. Atanasov, "Adaptive control of se(3) hamiltonian dynamics with learned disturbance features," 2022. [Online]. Available: https://arxiv.org/abs/2109.09974

[19] D. Hanover, P. Foehn, S. Sun, E. Kaufmann, and D. Scaramuzza, "Performance, precision, and payloads: Adaptive nonlinear mpc for quadrotors," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, p. 690–697, Apr. 2022. [Online]. Available: http://dx.doi.org/10.1109/LRA.2021.3131690

[20] R. Sinha, J. Harrison, S. M. Richards, and M. Pavone, "Adaptive robust model predictive control with matched and unmatched uncertainty," in *2022 American Control Conference (ACC)*, 2022, pp. 906–913.

[21] T. Z. Jiahao, K. Y. Chee, and M. A. Hsieh, "Online dynamics learning for predictive control with an application to aerial robots," in *Proceedings of The 6th Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, K. Liu, D. Kulic, and J. Ichnowski, Eds., vol. 205. PMLR, 14–18 Dec 2023, pp. 2251–2261. [Online]. Available: https://proceedings.mlr.press/v205/jiahao23a.html

[22] S. M. Richards, N. Azizan, J.-J. Slotine, and M. Pavone, "Adaptive-control-oriented meta-learning for nonlinear systems," 2021. [Online]. Available: https://arxiv.org/abs/2103.04490

[23] Y. Li, H. He, J. Wu, D. Katabi, and A. Torralba, "Learning compositional koopman operators for model-based control," *arXiv preprint arXiv:1910.08264*, 2019.

[24] G. Shi, K. Azizzadenesheli, M. O'Connell, S.-J. Chung, and Y. Yue, "Meta-adaptive nonlinear control: Theory and algorithms," 2021. [Online]. Available: https://arxiv.org/abs/2106.06098

[25] A. Nagabandi, C. Finn, and S. Levine, "Deep online learning via meta-learning: Continual adaptation for model-based rl," *arXiv preprint arXiv:1812.07671*, 2018.

[26] W. Xiao, H. Xue, T. Tao, D. Kalaria, J. M. Dolan, and G. Shi, "Anycar to anywhere: Learning universal dynamics model for agile and adaptive mobility," 2024. [Online]. Available: https://arxiv.org/abs/2409.15783

[27] S. Guttikonda, J. Achterhold, H. Li, J. Boedecker, and J. Stueckler, "Context-conditional navigation with a learning-based terrain- and robot-aware dynamics model," 2024. [Online]. Available: https://arxiv.org/abs/2307.09206

[28] J. Vertens, N. Dorka, T. Welschehold, M. Thompson, and W. Burgard, "Improving deep dynamics models for autonomous vehicles with multimodal latent mapping of surfaces," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023, pp. 4299–4306.

[29] M. O'Connell, G. Shi, X. Shi, K. Azizzadenesheli, A. Anandkumar, Y. Yue, and S.-J. Chung, "Neural-fly enables rapid learning for agile flight in strong winds," *Science Robotics*, vol. 7, no. 66, May 2022. [Online]. Available: http://dx.doi.org/10.1126/scirobotics.abm6597

[30] F. Xie, G. Shi, M. O'Connell, Y. Yue, and S.-J. Chung, "Hierarchical meta-learning-based adaptive controller," 2023. [Online]. Available: https://arxiv.org/abs/2311.12367

[31] J. Levy, J. Gibson, B. Vlahov, E. Tevere, E. Theodorou, D. Fridovich-Keil, and P. Spieler, "Meta-learning online dynamics model adaptation in off-road autonomous driving," 2025. [Online]. Available: https://arxiv.org/abs/2504.16923

[32] S. Zhao, Y. Xu, M. Kasaei, M. Khadem, and Z. Li, "Neural ode-based imitation learning (node-il): Data-efficient imitation learning for long-horizon multi-skill robot manipulation," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2024, pp. 8524–8530.

[33] K. Huang, R. Rana, A. Spitzer, G. Shi, and B. Boots, "Datt: Deep adaptive trajectory tracking for quadrotor control," *arXiv preprint arXiv:2310.09053*, 2023.

[34] G. Williams, A. Aldrich, and E. A. Theodorou, "Model predictive path integral control: From theory to parallel computation," *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 2, pp. 344–357, 2017.

[35] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.