A Learning-based Framework to Adapt Legged Robots On-the-fly to Unexpected Disturbances

Nolan Fey NOLANFEY@MIT.EDU
Department of Mechanical Engineering, Massachusetts Institute of Technology

He Li HLI25@ND.EDU
Nicholas Adrian NADRIAN@ND.EDU
Patrick Wensing PWENSING@ND.EDU
Department of Aerospace and Mechanical Engineering, University of Notre Dame

Michael Lemmon LEMMON@ND.EDU
Department of Electrical Engineering, University of Notre Dame

Editors: A. Abate, K. Margellos, A. Papachristodoulou

Abstract

State-of-the-art control methods for legged robots demonstrate impressive performance and robustness on a variety of terrains. Still, these approaches often lack an ability to learn how to adapt to changing conditions online. Such adaptation is especially critical if the robot encounters an environment with dynamics different than those considered in its model or in prior offline training. This paper proposes a learning-based framework that allows a walking robot to stabilize itself under disturbances neglected by its base controller. We consider an approach that simplifies the learning problem into two tasks: learning a model to estimate the robot's steady-state response and learning a dynamics model for the system near its steady-state behavior. Through experiments with the MIT Mini Cheetah, we show that we can learn these models offline in simulation and transfer them to the real world, optionally finetuning them as the robot collects data. We demonstrate the effectiveness of our approach by applying it to stabilize the quadruped as it carries a box of water on its back.

Keywords: Legged robots, Data-driven control, Adaptive control

1. Introduction

Recent advancements in legged locomotion controllers enable their widespread deployment in diverse environments in industry (Wensing et al., 2022). As these methods mature, there is a growing interest toward developing control algorithms that can overcome unanticipated conditions at test time. Most leading control approaches leverage model predictive control (MPC) (Di Carlo et al., 2018; Grandia et al., 2022; Meduri et al., 2023) or sim-to-real reinforcement learning (RL) (Lee et al., 2020; Kumar et al., 2021; Margolis et al., 2022), both of which may fail under disturbances not considered offline. Ideally, a locomotion controller could adapt on-the-fly when it encounters an environment outside the scope of its model (in the case of MPC) or its training distribution (in the case of RL). As a step toward this goal, this paper proposes a framework that enables a legged robot to learn how to stabilize itself under disturbances neglected by its base controller.

Rather than updating the base control policy or its model directly, our approach considers the response of the closed-loop system composed by the robot and its base controller. We separate the learning problem into two tasks: learning to estimate the robot's steady-state output and learning a local dynamics model near its steady-state behavior (Lemmon et al., 2022). In the context of legged

locomotion, we refer to the base velocity of the robot along an asymptotically stable periodic orbit (a.k.a. stable limit cycle and attractor) as the steady-state output.

Decomposing the problem into learning the system's steady-state behavior and a local dynamics model offers several benefits. The approach only requires that the base controller exhibits steady-state behavior when driven by a constant input. Thus, our framework is, in principle, applicable to model-based and model-free locomotion controllers alike. Once we have an estimate of the steady-state output of the system, we can subtract it from the true output to calculate the system's natural response. After removing its periodic component, the remaining output is hyperbolic in nature. It follows from the Hartman-Grobman theorem that the local behavior of a system near a hyperbolic equilibrium point can be captured by linearization techniques. With a local approximation of the dynamics, we can choose from a suite of available control options (i.e., linear quadratic regulator, MPC, passivity-based feedback, etc.) to force the system back to its steady-state behavior.

Past work in this area has shown that both a model for estimating the steady-state behavior and a local dynamics model can be learned through online regression techniques to adapt Raibert's 2D hopper to uneven terrain (Lemmon et al., 2022). We build upon this work to produce a more-general framework applicable to real-world robotic systems. Our contributions are as follows:

- We show that it is possible to train a multi layer perceptron (MLP) in simulation to predict the steady-state velocity of a robot to a given command, and that the model transfers to hardware.
- Similarly, we demonstrate sim-to-real transfer of a local dynamics model for the closed-loop system about its steady-state behavior. This step avoids an online training phase when the robot would be vulnerable to falling due to an unknown disturbance.
- We demonstrate finetuning these models online as the robot encounters new conditions.
- We demonstrate using these models in an MPC algorithm to stabilize the MIT Mini Cheetah while it carries a box of water.

Overall, we see our work as a step toward endowing robots with the capability of adapting their skills to new environments with unknown dynamics. Further, it is a strong example of improving an existing model-based controller with learning-based techniques.

1.1. Related Work

A number of methods for adapting robot control methods to new environments have been proposed in the literature. A traditional approach is to perform system identification to estimate parameters in models with known structure (Lee et al., 2024; An et al., 1985; Wensing et al., 2023). While this technique is critical for accurate simulation and model-based control, our framework takes a different perspective by estimating a local dynamics model of the closed-loop system composed of the robot and its base controller, rather than estimating parameters in a physics model. Thus, our approach addresses non-parametric dynamic uncertainty, which we argue is necessary for an uncertain world.

Instead of estimating the true parameters of the environment and the robot's dynamics, an alternative approach is to train a policy that is robust to a distribution of parameters (Tobin et al., 2017; Xie et al., 2021). While locomotion policies trained with domain randomization (DR) are robust, they are not necessarily optimal (Tan et al., 2018). In contrast, our method finetunes to the current environment to avoid sacrificing performance. Others have avoided the robustness-optimality

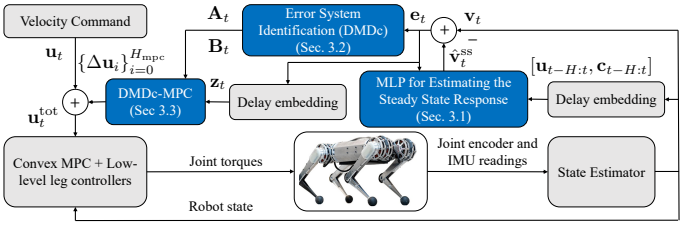

Figure 1: Proposed architecture. Main components in blue.

Algorithm 1 Adaptation Framework

Offline:

1: Train MLP to predict $\hat{\mathbf{v}}_t^{\text{ss}}$ (Sec. 3.1)
2: Train $\mathbf{A}_t, \mathbf{B}_t$ via DMDc (Sec. 3.2)

Repeat online:

3: Deploy DMDc-MPC (Sec. 3.3)
4: Finetune MLP, $\mathbf{A}_t, \mathbf{B}_t$ (Sec. 3.4)

tradeoff of DR by training a neural network to estimate a latent embedding of key environment parameters from a finite history of observations. They condition the control policy on this embedding to adapt in real-time (Lee et al., 2020; Kumar et al., 2021). Still, the test environment must be similar to those seen during training for this method to perform well. Our framework complements this approach by enabling adaptation to new scenarios not seen in prior training.

Another approach is to finetune the robot's base locomotion policy directly (Smith et al., 2021). Chen et al. (2023) consider this in addition to modulating between policies for different learned behaviors. While our framework also has mechanisms for finetuning, it is separate from the base policy, only assuming that it asymptotically approaches steady-state behavior in response to a constant input. This distinction makes our framework applicable to a more diverse set of controllers.

2. Overview

The system architecture developed in this work is shown in Figure 1. The main components are the MLP for estimating the steady-state velocity of the robot (Section 3.1), the error system identification for approximating the local dynamics near the steady-state behavior (Section 3.2), and the MPC for regulating the system back to steady-state (Section 3.3), which we refer to as DMDc-MPC to differentiate it from the base controller. The remaining components are the MIT Mini Cheetah robot (Katz et al., 2019) and its convex MPC (Di Carlo et al., 2018). The hardware results incorporate a state estimator (Bledt et al., 2018) for determining the base orientation and velocity. Algorithm 1 outlines our framework's training and deployment, distinguishing between offline and online steps.

The MLP is trained to predict the robot's steady-state velocity from a finite history of observations. Past work (Lemmon et al., 2022) demonstrates the use of a moment-matching model (Astolfi, 2010) for estimating the steady-state output. However, the formulation in Lemmon et al. (2022) is limiting because (1) it is not flexible to changes in the robot's input command and (2) it relies on a model for the disturbance generator. On real-world systems, a teleoperator may change the robot's velocity command frequently, and it's infeasible to anticipate all possible disturbances. We address the first issue by conditioning the MLP's prediction on the robot's velocity command. We train the model in simulation to predict the steady-state velocity of the Mini Cheetah controlled by its convex MPC base controller (Di Carlo et al., 2018) for a range of commands. Our MLP does not rely on a model of a disturbance, nor do our training environments include disturbances. Alternatively, we demonstrate through a sim-to-sim transfer experiment (Section 4) that the MLP can be finetuned when the robot encounters new conditions that change its steady-state behavior.

Similar to past work, we apply dynamic mode decomposition with control (DMDc) (Proctor et al., 2016) to learn a local dynamics model for the closed-loop system about its steady-state behavior. To learn how varying the control input affects the error system's state, it's necessary to

"kick" the control input to the robot's base controller by adding command perturbations. However, it may be unsafe to "kick" the control input in safety-critical scenarios, where risk for a fall is intolerable. We avoid this safety hazard by initializing the model in simulation through an offline training phase. From the start of deployment, we use the model in the DMDc-MPC to find a sequence of perturbations to the velocity command that minimizes the robot's deviation from its steady-state behavior over a finite horizon. We also finetune the model online through rank-1 updates to adapt to new disturbances. We validate our approach through simulation and hardware experiments with the Mini Cheetah, where it must stabilize itself under disturbances not encountered in its prior training.

3. Method

The goal of our learning-based framework is to stabilize the robot in the presence of a disturbance neglected by its base controller. An accurate estimate of the robot's steady-state behavior is needed as a reference to where a feedback controller will regulate the robot. The following section describes a method for estimating this behavior for a range of possible velocity commands to the robot's base controller. The feedback controller for stabilizing the robot relies on a model of the closed-loop system about its steady-state behavior. We describe how to learn such a model with DMDc (Section 3.2), and how we formulate the DMDc-MPC as the feedback controller in our framework (Section 3.3). We also describe how we finetune the MLP and the DMDc model online (Section 3.4).

3.1. Learning to Estimate the Robot's Steady-State Response

In this section, we describe the supervised learning procedure to train the MLP that estimates the steady-state velocity at time t, $\mathbf{v}_t^{\text{ss}} = [\boldsymbol{\omega}_t^{\text{ss}}, \boldsymbol{v}_t^{\text{ss}}]^T \in \mathbb{R}^6$, for a given command, $\mathbf{u}_t = \mathbf{v}_t^{\text{cmd}} \in \mathbb{R}^6$. We denote $\boldsymbol{\omega}_t^{\text{ss}} \in \mathbb{R}^3$ and $\boldsymbol{v}_t^{\text{ss}} \in \mathbb{R}^3$ as the angular and linear components of the steady-state velocity, respectively. The MLP's input is an observation history from the robot's base controller, $\mathbf{o}_{t-H:t} \in \mathbb{R}^{(H+1)\times N_o}$, where $H+1$ is the history length in timesteps and N_o is the dimension of the observation space. We perform stochastic gradient descent with a mean-squared error loss function between the model's output, $\hat{\mathbf{v}}_t^{\text{ss}}$, and the state estimate of the true velocity, $\mathbf{v}_t \in \mathbb{R}^6$.

The challenge in training the model is to avoid fitting to transient effects. Thus, it's critical only to train on data from when the system is in steady-state. We collect training data offline in simulation, recording the state estimates for a range of valid velocity commands. Each episode, the robot receives a random command. The system has a transient response to the command before settling to steady-state, so we drop the first few seconds of data from the training dataset. We continue collecting data for a fixed episode length or until a fall, in which case we discard the data.

Selecting the observation space, $\mathbf{o}_t$, involves a design tradeoff: providing the model enough information to estimate the steady-state velocity but not enough to fit to transient effects. We found that including the state estimate in the observation led to overfitting. Instead, we selected the history of observations as $\mathbf{o}_{t-H:t} = [\mathbf{u}_{t-H:t}, \mathbf{c}_{t-H:t}]$, where $\mathbf{c}_{t-H:t} \in [0,1]^{(H+1)\times 4}$ is a history of contact phase variables, which encode the contact status of each foot. The observation allows the network to capture oscillations due to the gait cycle. Each component i takes a value 0 when contact i is midway through a swing phase, and linearly increases with time such that it is 0.5 mid-stance. The components reset to 0 when contact i reaches mid-swing again.

We collected a training dataset in simulation (MIT Biomimetics Robotics Lab, 2019), and trained an MLP to map, $[\mathbf{u}_{t-H:t}, \mathbf{c}_{t-H:t}]$, to $[\boldsymbol{\omega}_t^{z,\text{ss}}, \boldsymbol{v}_t^{x,\text{ss}}, \boldsymbol{v}_t^{y,\text{ss}}]$, where $H = 10$ (100 ms). We

chose not to estimate the full $\mathbf{v}_t^{\text{ss}}$ because the convex MPC base controller only takes the desired yaw-rate, forward velocity, and lateral velocity as input. Each episode, we sampled a command from a gaussian distribution with mean $\text{diag}(0\,\text{rad/s}, 0.5\,\text{m/s}, 0\,\text{m/s})$ and variance $\text{diag}(0.33\,\text{rad}^2/\text{s}^2, 0.5\,\text{m}^2/\text{s}^2, 0.33\,\text{m}^2/\text{s}^2)$. We dropped the first two seconds of data to avoid fitting to transient effects. We terminated the episode after 10 s or if an emergency stop was triggered. The MLP had 3 hidden layers with 16, 64, and 16 neurons, as well as ReLU activations after the first two layers and layer norms after the last two layers. We trained for 15,000 iterations with the AdamW optimizer (Loshchilov and Hutter, 2017) and a batch size of 32. We set the initial learning rate to 1×10^{-3} and decreased it by a factor of 10 when the validation loss plateaued (Paszke et al.).

Tables 1 and 2 show the maximum mean absolute error (MAE) of each episode in the test set for MLP's trained with datasets of various sizes and observation history lengths. We noticed diminishing improvements when expanding the size of the training data. Extending the observation history length improved the model up to a point where the performance degraded.

We validated that the MLP transferred to the actual system by conducting an experiment where we suddenly changed the robot's velocity command and observed the MLP's output. Figure 2 is a plot of the forward velocity (blue), the MLP estimate (orange), and the command during the experiment (green). During steady-state, the MLP tracks the robot's state estimate, including oscillations due to the gait cycle. After command changes, the robot overshoots the command, but the MLP ignores the transient effects. The results validate our method of curating the training data to avoid fitting to transients and show that the MLP, trained in simulation, transfers to the real system.

The MLP prediction error from the velocity state estimate, $\mathbf{e}_t = \hat{\mathbf{v}}_t^{\text{ss}} - \mathbf{v}_t$, captures deviations from the steady-state behavior due to disturbances not considered during prior training. The next section discusses how to learn a dynamics model for this error signal.

Num. Episodes	Max Test MAE
100	0.1301
1000	0.0429
12000	0.0211

Table 1: MLP Test Loss by Training Set Size

Length	# Params	Max Test MAE
1	6627	0.0308
10	7635	0.0211
20	8755	0.0267

Table 2: MLP Test Loss by H

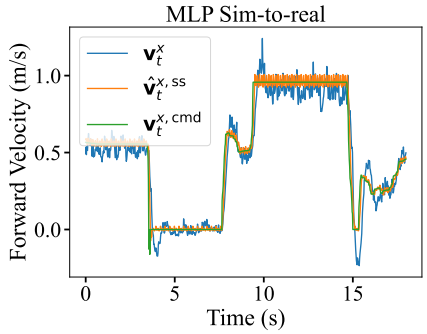

Figure 2: The MLP ignores the transient effects due to command changes and estimates the robot's steady-state velocity.

3.2. Learning a Local Dynamics Model

Our framework proposes learning a dynamics model to predict how $\mathbf{e}_t$ evolves over time, which is equivalent to learning a dynamics model for the robot about its steady-state behavior. Since we removed the steady-state behavior, the dynamics can be captured by a linear model. Prior work proposed applying DMDc to learn the dynamics model (Lemmon et al., 2022). We extend this work by showing how to learn this model in an offline simulation and transfer it to the real world, avoiding

the need to "kick" the robot's command on hardware. Additionally, we demonstrate training a single DMDc model that sufficiently captures the dynamics of $\mathbf{e}_t$ regardless of the robot's command $\mathbf{u}_t$.

We learn the error system dynamics through input-output data (Lemmon et al., 2022). We treat $\mathbf{e}_t$ as the error system's output and use a length N_e delay embedding as the system's state, $\mathbf{z}_t = [\mathbf{e}_{t-N_e}, \mathbf{e}_{t-(N_e+1)}, ..., \mathbf{e}_{t-1}, \mathbf{e}_t]^T$. Taken's embedding theorem (Takens, 1981) states that if the system is observable, a history of outputs can be used as the system's state provided N_e is sufficiently large. The system's input is $\Delta\mathbf{u}_t$, where the total velocity command is $\mathbf{u}_t^{\text{tot}} = \mathbf{u}_t + \Delta\mathbf{u}_t$.

We collect the input-output dataset, $\{\mathbf{z}_i, \Delta\mathbf{u}_i\}_{i=0}^{N-1}$, through an offline training period. We select commands for the base controller, $\{\mathbf{u}_k\}_{k=0}^{N_u-1}$, that adequately cover the space of possible inputs to the robot. For fixed intervals, I, we send one of these commands to the robot until we sequence through all of the commands. Simultaneously, we periodically "kick" the control input such that,

$$\mathbf{u}_t^{\text{tot}} = \sum_{k=0}^{N_u-1} (\mathbf{u}_k + \lambda_k^{t-kI}\mathbf{a}_k)\{kI < t \leq (k+1)I\}, \tag{1}$$

where $\{\mathbf{a}_k, \lambda_k\}$ are uniform random variables that characterize the magnitude and decay rate of the k^{th} perturbation, and $\{kI < t \leq (k+1)I\} = 1$ if the condition inside the brackets is true and zero otherwise. Each $\mathbf{a}_k$ has the same dimension as the control input, and each $\lambda_k < 1$.

We apply DMDc to compute matrices $\mathbf{A}$ and $\mathbf{B}$ that minimize the least-squares regression loss between $\mathbf{A}\mathbf{z}_t + \mathbf{B}\Delta\mathbf{u}_t$ and $\mathbf{z}_{t+1}$ on the training dataset. We form matrices

$$\boldsymbol{Z} = \begin{bmatrix} \boldsymbol{z}_0 & \boldsymbol{z}_1 & \dots & \boldsymbol{z}_{N-2} \\ \Delta\boldsymbol{u}_0 & \Delta\boldsymbol{u}_1 & \dots & \Delta\boldsymbol{u}_{N-2} \end{bmatrix}, \quad \boldsymbol{Z'} = \begin{bmatrix} \boldsymbol{z}_1 & \boldsymbol{z}_2 & \dots & \boldsymbol{z}_{N-1} \\ \Delta\boldsymbol{u}_1 & \Delta\boldsymbol{u}_2 & \dots & \Delta\boldsymbol{u}_{N-1} \end{bmatrix}, \tag{2}$$

and take the first n rows and n columns of $\boldsymbol{W} = \boldsymbol{Z'}(\boldsymbol{Z}^T\boldsymbol{Z})^{-1}\boldsymbol{Z}^T$ as the matrix $\boldsymbol{A}$, where $\mathbf{z}_t \in \mathbb{R}^n$, and the next m columns of the first n rows as the matrix $\boldsymbol{B}$, where $\Delta\mathbf{u}_t \in \mathbb{R}^m$.

Figure 3 visualizes the training data (top row) and model-fitting error (bottom row) from training an error system dynamics model for the Mini Cheetah. We followed the procedure above for data collection, sending $N_u = 8$ base commands, $\mathbf{u}_k$, and perturbations, $\{\mathbf{a}_k, \lambda_k\}$, over 4 minutes of simulation. We set $N_e = 10$, making $\mathbf{z}_t \in \mathbb{R}^{30}$. On the top row, the orange lines are the command perturbation, $\Delta\mathbf{u}_t$, while the blues lines are the MLP prediction errors $\mathbf{e}_t$. Despite the variety in base commands, the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ achieve a tight fit, other than the spikes near the peaks of the perturbations. We can improve the fit online through recursive updates (Section 3.4.1).

3.3. Closing the Loop with MPC

In this section, we close the loop on the MLP prediction error, $\mathbf{e}_t$, to regulate the system to its steady-state behavior. Equivalently, we want to design a feedback controller to regulate $\mathbf{e}_t$ to zero. We propose using our DMDc dynamics model within a finite-horizon LQR problem,

$$\begin{aligned} \min_{\mathbf{z},\Delta\mathbf{u}} \quad & \sum_{i=1}^{H_{\text{mpc}}} \mathbf{z}_i^T\mathbf{Q}\mathbf{z}_i + \Delta\mathbf{u}_i^T\mathbf{R}\Delta\mathbf{u}_i, \\ \text{subject to:} \quad & \mathbf{z}_{t+1} = \mathbf{A}\mathbf{z}_t + \mathbf{B}\Delta\mathbf{u}_t,\ \mathbf{z}_0 = \mathbf{z}_t,\ \Delta\mathbf{u}_0 = \Delta\mathbf{u}_t, \end{aligned}$$

where $\mathbf{z} = [\mathbf{z}_1, ..., \mathbf{z}_{H_{\text{mpc}}}]$, and $\Delta\mathbf{u} = [\Delta\mathbf{u}_1, ..., \Delta\mathbf{u}_{H_{\text{mpc}}}]$. We solve it in MPC fashion using CVXOPT (Andersen et al., 2023) and execute the control tape, $\{\Delta\mathbf{u}_i\}_{i=0}^{H_{\text{mpc}}}$, such that the total command

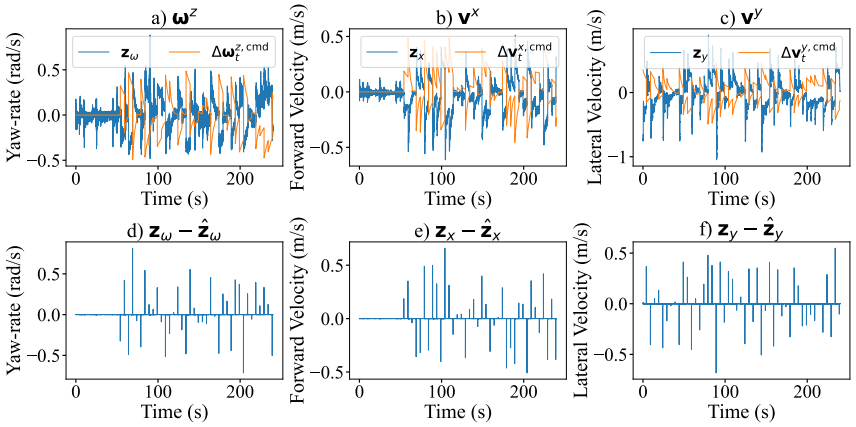

Figure 3: Training data (a,b,c) and fitting error (d,e,f) for the Mini Cheetah's DMDc model.

to the base controller is $\mathbf{u}_{t+i}^{\text{tot}} = \mathbf{u}_{t+i} + \Delta\mathbf{u}_i$. Our choice of MPC was motivated by its ability to ensure consistent performance across iterative calls to the adaptation framework. The robot follows the trajectory from the previous iteration as it waits for the next control tape.

3.4. Finetuning Models Online

During a deployment, the robot may encounter new environments different from those seen during training. Ideally, the robot can finetune to these conditions, rather than resort to further offline training. Our framework has two finetuning mechanisms: recursive updates to the error system dynamics model (Section 3.4.1) and MLP transfer learning (Section 3.4.2). Upon a sudden perturbation, the feedback controller (Section 3.3) will stabilize the robot. However, if a disturbance gradually changes the robot's steady-state behavior, then MLP transfer learning is necessary to capture this change. The recursive DMDc updates finetune the error system dynamics model as its equilibrium point shifts due to changes in the MLP.

3.4.1. RECURSIVE DMDC UPDATES

We propose performing rank-one updates to the DMDc model through a recursive least squares algorithm, which is a variant of the Kalman filter (Kalman, 1960). The recursive updates improve the model as the robot collects more data and finetune it to disturbances not seen during prior training. Since we pretrained the model offline, we can implement the feedback controller immediately and learn as the robot collects more data, avoiding the need to "kick" the robot's control input online.

Each iteration, we observe the transition $(\mathbf{z}_{t-1}, \mathbf{u}_{t-1}) \to \mathbf{z}_t$. We perform a rank one update as

$$\mathbf{K}_t = \frac{\mathbf{P}_{t-1}\boldsymbol{\psi}_{t-1}}{R_2 + \boldsymbol{\psi}_{t-1}^T \mathbf{P}_{t-1}\boldsymbol{\psi}_{t-1}}, \tag{3}$$

$$\mathbf{W}_t = \mathbf{W}_{t-1} + \mathbf{K}_t\big(\mathbf{z}_t - (\mathbf{A}_{t-1}\mathbf{z}_{t-1} + \mathbf{B}_{t-1}\Delta\mathbf{u}_{t-1})\big), \tag{4}$$

$$\mathbf{P}_t = \mathbf{P}_{t-1} + \mathbf{R}_1 - \mathbf{K}_t\boldsymbol{\psi}_{t-1}^T\mathbf{P}_{t-1}, \tag{5}$$

where $\mathbf{K}_t$ is the Kalman gain, $\mathbf{P}_t \in \mathbb{R}^{(n+m)\times(n+m)}$ is the estimated covariance matrix of the parameters, $\boldsymbol{\psi}_t = [\mathbf{z}_t, \mathbf{u}_t]^T$, $\mathbf{R}_1 \in \mathbb{R}^{(n+m)\times(n+m)}$ is the process noise covariance, and R_2 is the

measurement noise covariance. We take the first n rows and n columns of $\boldsymbol{W}_t$ as the matrix $\boldsymbol{A}_t$, and the next m columns of the first n rows as the matrix $\boldsymbol{B}_t$. We initialize $\mathbf{P}_0 = \boldsymbol{Z}\boldsymbol{Z}^T$, where $\boldsymbol{Z}$ is the data matrix collected during the offline training period, and we set $\boldsymbol{A}_0$ and $\boldsymbol{B}_0$ with the $\boldsymbol{A}$ and $\boldsymbol{B}$ matrices estimated through pretraining (Section 3.2). Due to this strong prior, we set $\mathbf{R}_1 = k\,\mathbf{I}_{(n+m)\times(n+m)}$ where $\mathbf{I}_{(n+m)\times(n+m)}$ is the $(n+m)\times(n+m)$ identity matrix and $\mathbf{R}_2/k \gg 1$.

3.4.2. MLP TRANSFER LEARNING THROUGH FEEDBACK

We propose transfer learning to finetune the MLP online. When the robot suddenly encounters a new disturbance, it may be impossible to collect a large enough dataset to retrain the last layer(s) of the MLP. Instead, we consider making rank-one updates to the MLP's last layer. We apply the same recursive least squares algorithm as in Section 3.4.1 (Equations 3-5), where $\boldsymbol{\psi}_t^{\text{MLP}}$ is the output of the MLP's second to last layer and $\mathbf{W}_t^{\text{MLP}}$ are the weights of its last layer. The weights update is

$$\mathbf{W}_t^{\text{MLP}} = \mathbf{W}_{t-1}^{\text{MLP}} + \mathbf{K}_t^{\text{MLP}}\big(\mathbf{v}_{t-1} - (\mathbf{W}_{t-1}^{\text{MLP}})^T\,\boldsymbol{\psi}_t^{\text{MLP}}\big), \tag{6}$$

which improves our MLP through feedback from the robot's state estimate. For ease of implementation, we initialize $\mathbf{W}_0^{\text{MLP}}$ to zero and $\mathbf{P}_0^{\text{MLP}} = 100\,\mathbf{I}_{N_e^{\text{MLP}}\times N_e^{\text{MLP}}}$, where N_e^{MLP} is the MLP embedding dimension. We set the noise covariances such that $\mathbf{R}_2^{\text{MLP}}/k > 1$, where $\mathbf{R}_1^{\text{MLP}} = k\,\mathbf{I}_{N_e\times N_e}$, to avoid sudden large changes in the equilibrium point of the DMDc model.

3.4.3. SIM-TO-SIM TRANSFER EXPERIMENT

We tested our finetuning methods through an experiment where we varied the simulation's gravity, such that $\mathbf{g} = [0, 1.5\sin(\frac{2\pi}{3}t), -9.81]^T\,\text{m/s}^2$. In prior training, the MLP and DMDc model only saw environments with $\mathbf{g} = [0, 0, -9.81]^T\,\text{m/s}^2$. We commanded the robot to walk forward at +1 m/s. Figure 4 visualizes the estimates of the robot's steady-state (blue) and true (orange) lateral velocity. Initially, we shut off the DMDc-MPC to show that the disturbance caused the robot to sway left and right, and we inference the pretrained MLP, which does not capture the oscillations in lateral velocity. At 10 s, we begin finetuning the MLP to estimate the robot's new steady-state behavior. We set $\mathbf{R}_1^{\text{MLP}} = 10^{-3}\mathbf{I}_{N_e\times N_e}$ and $\mathbf{R}_2^{\text{MLP}} = 10^3$, and the Kalman gain $\mathbf{K}_t^{\text{MLP}}$ converged to $\sim$0.025. As the filter converges, the MLP model adjusts its estimate to capture the oscillatory behavior. At 20 s, we turn on the DMDc-MPC and the recursive DMDc updates, with $\mathbf{R}_1 = 10^{-3}\mathbf{I}_{(n+m)\times(n+m)}$ and $\mathbf{R}_2 = 10^5$. The Kalman gain $\mathbf{K}_t$ converged to $\sim$0.0015. The DMDc-MPC reduces the amplitude of the oscillations, as the DMDc model finetunes about the new steady-state behavior.

4. Results

We validated our adaptation framework on an MIT Mini Cheetah (Katz et al., 2019), training the MLP and DMDc model in simulation (MIT Biomimetics Robotics Lab, 2019) and transferring them to hardware. We provide simulation results of improved disturbance recovery with our framework (Section 4.1). In Section 4.2, we compare the velocity tracking of the convex MPC base controller (Di Carlo et al., 2018) with and without our framework while carrying a box of water (Figure 5).

4.1. Improved Disturbance Recovery

Before hardware tests, we validated our adaptation framework in simulation by comparing the robot's disturbance rejection ability with and without our framework. Figure 6 shows the robot's

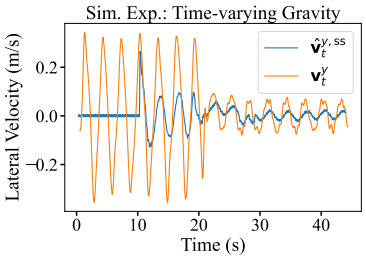

Figure 4: Recursive finetuning of the MLP activates at 10 s. The DMDc-MPC and recursive finetuning of DMDc models activates at 20 s.

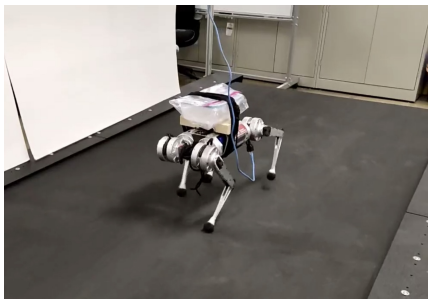

Figure 5: Mini Cheetah running forward on treadmill while carrying a box of water.

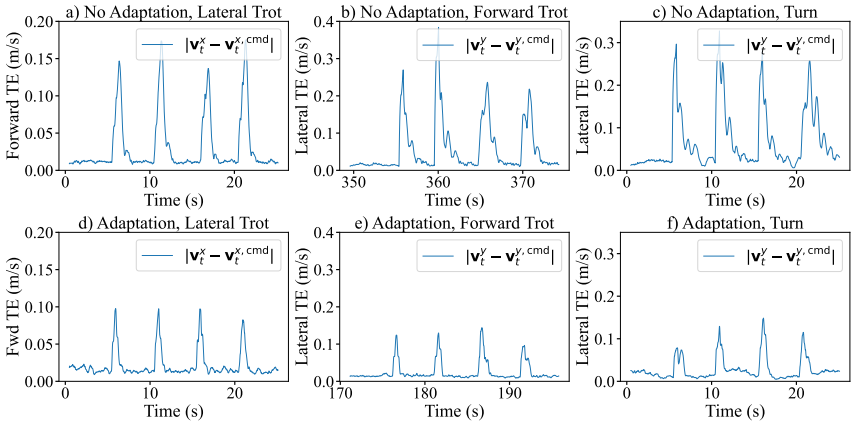

Figure 6: Tracking errors (TE) from disturbance rejection experiments. With our adaptation framework (d,e,f), the jump in tracking error after the impulse is smaller, and it decays faster. This held true for various operating points, including a lateral trot (a,d), a forward trot (b,e), and a turn (e,f).

absolute tracking error parallel to the direction of the disturbance force across various tests, including a lateral trot at 0.33 m/s (experiments a,d), a forward trot at 1.0 m/s (experiments b,e), and a turn with $\boldsymbol{\omega}_t^{z,\text{cmd}} = 0.33$ rad/s and $\boldsymbol{v}_t^{x,\text{cmd}} = 1.0$ m/s (experiments c,f). For all experiments, we kicked the robot at 5, 10, 15, and 20 s with an impulse, $\boldsymbol{J} \in \mathbb{R}^3$, over 1 ms though the COM of its base. In experiments a and d, we sent $\boldsymbol{J} = [+4.5, 0, 0]$ N·s at 5 and 10 s, and $\boldsymbol{J} = [-4.5, 0, 0]$ N·s at 15 and 20 s. In the other experiments, we sent $\boldsymbol{J} = [0, +4.5, 0]$ N·s at 5 and 10 s, and $\boldsymbol{J} = [0, -4.5, 0]$ N·s at 15 and 20 s. Experiments a, b, and c were baseline tests without adaptation, whereas experiments d, e, and f included our framework. Our adaptation framework dampened the amplitude and increased the decay rate of the tracking error after the disturbances. The results show that our framework improves the disturbance rejection ability of the robot's convex MPC controller.

4.2. Carrying Box of Water

We deployed our framework on hardware to verify it improves the stability of the robot while it carries a box of water. Due to onboard compute limitations, we run our adaptation framework on a laptop at 50 Hz and communicate with the robot over an ethernet cable.

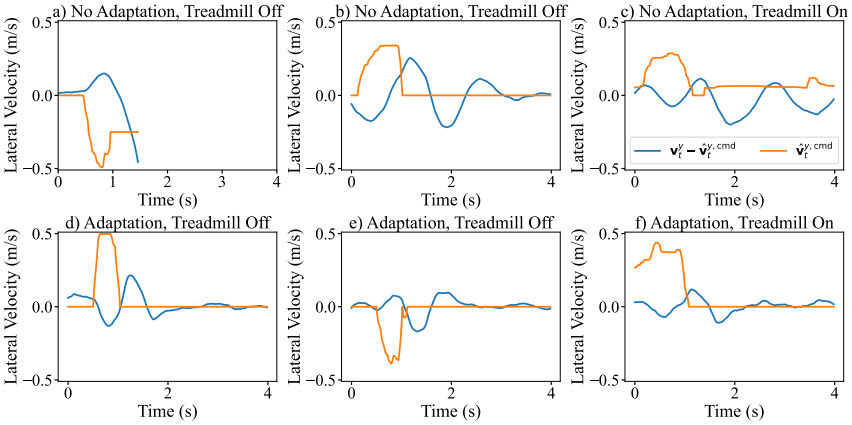

Figure 7: Hardware tests carrying a box of water. Without adaptation (a,b,c), the robot sways due to the control perturbations (orange). In experiment a, the robot fell. With adaptation (d,e,f), the robot's tracking error (blue) still oscillates, but with a smaller amplitude and a faster decay rate.

The convex MPC base controller, without our framework, could walk while carrying the box of water (Figure 5). However, when perturbed, the water would slosh back and forth, causing the robot to oscillate about its commanded velocity (Figure 7, a,b,c). We perturbed the robot by sending it sudden lateral velocity commands. Plots of the commanded lateral velocity and the tracking error are shown in Figure 7. The tracking error (blue) is filtered over a 0.5 s averaging window. In experiments a, b, d, and e the robot received zero velocity command, while in the experiments c and f, the robot received a forward velocity command of 0.8 m/s. Experiments a, b and c were without our adaptation framework. The robot swayed back and forth after the perturbation and in one case (experiment a) fell over. With our adaptation framework (experiments d,e,f), the amplitude of the oscillations were smaller and decayed at a faster rate. The results show that our framework improved the disturbance recovery of the robot's convex MPC base controller, despite not seeing conditions similar to the box of water in prior training.

5. Conclusion

This paper proposed a learning-based framework to adapt a legged robot to conditions neglected by its base controller. The framework separates the learning problem into two tasks: learning to estimate the steady-state response of the robot, and learning a local dynamics model near the system's steady-state behavior. Through experiments, we show that is is possible to train in an offline simulation (1) an MLP to predict the robot's steady-state velocity to various input commands and (2) a local dynamics model of the closed-loop system, and that these models transfer to the real-world. We also demonstrated that these models can be finetuned, as the DMDc-MPC stabilizes the robot. We applied our proposed approach to the MIT Mini Cheetah, both in simulation and on hardware. We demonstrate that our framework improves the disturbance rejection ability of the robot's convex MPC base controller, and that it stabilizes the robot as it carries a box of water on its back.

Future work may consider applying our approach to a robot controlled by an RL policy to stabilize it via feedback. In principle, the framework may apply to any system that approaches steady-state behavior when given a constant input.

Acknowledgments

NF and ML acknowledge partial support from the National Science Foundation under Grant Nos. 2141064 and CNS-2228092, respectively.

References

Chae H An, Christopher G Atkeson, and John M Hollerbach. Estimation of inertial parameters of rigid body links of manipulators. In *1985 24th IEEE Conference on Decision and Control*, pages 990–995. IEEE, 1985.

M. S. Andersen, J. Dahl, and L. Vandenberghe. CVXOPT: A python package for convex optimization, version 1.3.1. http://cvxopt.org, 2023.

Alessandro Astolfi. Model reduction by moment matching for linear and nonlinear systems. *IEEE Transactions on Automatic Control*, 55(10):2321–2336, 2010. doi: 10.1109/TAC.2010.2046044.

Gerardo Bledt, Matthew J. Powell, Benjamin Katz, Jared Di Carlo, Patrick M. Wensing, and Sangbae Kim. Mit cheetah 3: Design and control of a robust, dynamic quadruped robot. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2245–2252, 2018. doi: 10.1109/IROS.2018.8593885.

Annie S. Chen, Govind Chada, Laura Smith, Archit Sharma, Zipeng Fu, Sergey Levine, and Chelsea Finn. Adapt on-the-go: Behavior modulation for single-life robot deployment, 2023.

Jared Di Carlo, Patrick M. Wensing, Benjamin Katz, Gerardo Bledt, and Sangbae Kim. Dynamic locomotion in the mit cheetah 3 through convex model-predictive control. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–9, 2018. doi: 10.1109/IROS.2018.8594448.

Ruben Grandia, Fabian Jenelten, Shaohui Yang, Farbod Farshidian, and Marco Hutter. Perceptive locomotion through nonlinear model predictive control. 2022.

R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, 82(1):35–45, 03 1960. ISSN 0021-9223. doi: 10.1115/1.3662552. URL https://doi.org/10.1115/1.3662552.

Benjamin Katz, Jared Di Carlo, and Sangbae Kim. Mini cheetah: A platform for pushing the limits of dynamic quadruped control. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6295–6301, 2019. doi: 10.1109/ICRA.2019.8793865.

Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. Rma: Rapid motor adaptation for legged robots, 2021.

Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science Robotics*, 5(47):eabc5986, 2020. doi: 10.1126/scirobotics.abc5986. URL https://www.science.org/doi/abs/10.1126/scirobotics.abc5986.

Taeyoon Lee, Jaewoon Kwon, Patrick M. Wensing, and Frank C. Park. Robot model identification and learning: A modern perspective. *Annual Review of Control, Robotics, and Autonomous Systems*, 7(1):null, 2024. doi: 10.1146/annurev-control-061523-102310. URL https://doi.org/10.1146/annurev-control-061523-102310.

M. D. Lemmon, P. M. Wensing, V. Kurtz, and H. Lin. Learning to control robot hopping over uneven terrain. In *2022 American Control Conference (ACC)*, pages 520–525, 2022. doi: 10.23919/ACC53348.2022.9867630.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Gabriel B Margolis, Ge Yang, Kartik Paigwar, Tao Chen, and Pulkit Agrawal. Rapid locomotion via reinforcement learning, 2022.

Avadesh Meduri, Paarth Shah, Julian Viereck, Majid Khadiv, Ioannis Havoutis, and Ludovic Righetti. Biconmp: A nonlinear model predictive control framework for whole body motion planning. *IEEE Transactions on Robotics*, 39(2):905–922, 2023. doi: 10.1109/TRO.2022.3228390.

MIT Biomimetics Robotics Lab. Cheetah-software. https://github.com/mit-biomimetics/Cheetah-Software, 2019.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library.

Joshua L. Proctor, Steven L. Brunton, and J. Nathan Kutz. Dynamic mode decomposition with control. *SIAM Journal on Applied Dynamical Systems*, 15(1):142–161, 2016. doi: 10.1137/15M1013857. URL https://doi.org/10.1137/15M1013857.

Laura Smith, J. Chase Kew, Xue Bin Peng, Sehoon Ha, Jie Tan, and Sergey Levine. Legged robots that keep on learning: Fine-tuning locomotion policies in the real world, 2021.

Floris Takens. Detecting strange attractors in turbulence. In David Rand and Lai-Sang Young, editors, *Dynamical Systems and Turbulence, Warwick 1980*, pages 366–381, Berlin, Heidelberg, 1981. Springer Berlin Heidelberg. ISBN 978-3-540-38945-3.

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

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world, 2017.

Patrick M. Wensing, Michael Posa, Yue Hu, Adrien Escande, Nicolas Mansard, and Andrea Del Prete. Optimization-based control for dynamic legged robots, 2022.

Patrick M. Wensing, Günter Niemeyer, and Jean-Jacques E. Slotine. A geometric characterization of observability in inertial parameter identification, 2023.

Zhaoming Xie, Xingye Da, Michiel van de Panne, Buck Babich, and Animesh Garg. Dynamics randomization revisited:a case study for quadrupedal locomotion, 2021.