Online Learning of Unknown Dynamics for Model-Based Controllers in Legged Locomotion

Yu Sun[1], Wyatt L. Ubellacker[2], Wen-Loong Ma[2], Xiang Zhang[1], Changhao Wang[1], Noel V. Csomay-Shanklin[2], Masayoshi Tomizuka[1], Koushil Sreenath[1], and Aaron D. Ames[2]

***Abstract*—The performance of a model-based controller can severely suffer when its model inaccurately represents the real world dynamics. We propose to learn a time-varying, locally linear residual model along the robot's current trajectory, to compensate for the prediction errors of the controller's model. Supervised learning is performed online, as the robot is running in the unknown environment, using data collected from its immediate past. We theoretically investigate our method in its general formulation, then apply it to a bipedal controller derived from the full-order dynamics of virtual constraints, and a quadrupedal controller derived from a simplified model of contact forces. For a biped in simulation, our method consistently outperforms the baseline and a recent learning-based method. We also experiment with a 12 kg quadruped in simulation and real world, where the baseline fails to walk with 10 kg of payload but our method succeeds.**

***Index Terms*—Model Learning for Control, Legged Robots**

I. INTRODUCTION

MANY popular frameworks for controller design are based on the robot's model of dynamics. In the real world, however, this model can often turn out to be inaccurate, due to, for example, misspecification of the robot's physical parameters, mechanical wear and tear, and deployment-time interventions such as additional payload. While a well designed controller is robust to small inaccuracies in the dynamics, large deviations may significantly degrade its performance.

Our goal is to make corrections to the model behind the controller during deployment, through online learning using onboard sensors. Since the nature of a model is to predict the future given the past, data for supervised learning of dynamics can be collected automatically without human supervision, as time goes on and the future is revealed.

Because data are generated along the controller's trajectory that we are trying to improve, they might not contain enough information about the entire system. Nevertheless, we find it sufficient to limit the scope of learning to a *local* neighborhood of the current point in the current trajectory, instead of the entire system, if the learned model is updated in real time as the trajectory evolves.

Manuscript received: February 24, 2021; Revised May 14, 2021; Accepted June 13, 2021.

This paper was recommended for publication by Editor Dana Kulic upon evaluation of the Associate Editor and Reviewers' comments. This work was supported by DOW Chemical Project 227027AT, NSF NRI Award 1924526, NSF DCSDVTECH Award 1923239, and NSF Grant CMMI-1944722.

[1]University of California, Berkeley, CA. yusun, xiang_zhang_98, changhaowang, tomizuka, koushils@berkeley.edu

[2]California Institute of Technology, Pasadena, CA. wubellac, wma, noelcs, ames@caltech.edu

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

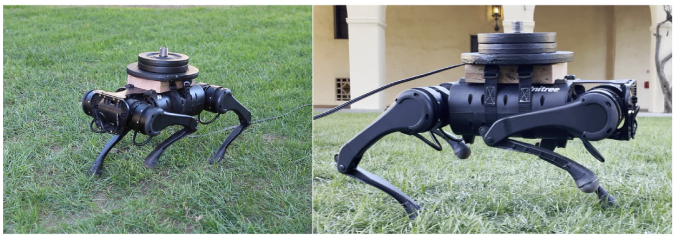

Fig. 1. The 12 kg A1 robot carrying 10 kg of payload with our method, tested for trotting in place and walking forward.

Fortunately, even globally complex systems, such as the highly nonlinear hybrid systems for legged locomotion, can be locally simple. Therefore, we also find it sufficient to learn with only a *time-varying, locally linear model*, which is computationally feasible to be updated in real time.

We first develop the intuition of online learning into a method for controllers that drive the outputs to the desired behavior based on control-affine models. We then analyze this method's theoretical properties, and evaluate it in two applications for legged locomotion.

A. Related Work

1) System identification: Given a system with known form but unknown parameters, system identification (sysID) estimates these parameters from signals given by the system ([1]). Recent papers have applied sysID for inertial parameters of a humanoid ([2], [3]). The parameters are assumed to be constant in time, and estimation is performed before the deployment of a controller. Thinking of identification as training and deployment as testing, sysID trains a model before deployment, and keeps the model fixed during testing. Since the goal is to model the system's behavior globally across the entire state space, sysID usually requires driving the system to diverse enough states, using diverse enough inputs. This requirement is known as persistence of excitation in control theory, and might be difficult to satisfy without many samples from the plant. In contrast, we only model the system's behavior locally, around the small neighborhood of our current state, learning a linear model even for complex systems with relatively few samples.

2) Learning dynamics: There is also a developing community in machine learning, modeling dynamics of the environment from interactions and observations ([4], [5], [6], [7], [8], [9], [10], [11]). It has roughly the same goal as sysID, but often uses powerful tools from deep learning, and does not assume any specific form of the system; here, learning often

produces a general prediction model. We diverge from this community in the global vs. local aspect (like from sysID), but embrace its philosophy of learning a general model with parameters that might not be interpretable.

3) Adaptive control: The intuition of adaptive control is to change the controller's parameters during deployment ([12]). Online system identification ([13]) is the most relevant subfield, since it directly concerns the model behind the controller. It has been successfully applied in manipulation ([14], [15], [16], [17], [18]), and for the location and inertial parameters of the center of mass of a quadruped ([19]). For online sysID, the parameters considered are very specific, and estimation relies on the physics of the model and the particular controller for the application. Our work considers parameters in a much more general sense closer to that of the machine learning community. Our parameters are functions of the state, thus are inherently time-varying and abstract. In fact, in the control-affine form, every term of our dynamics is updated in real time as the state evolves. Furthermore, unlike sysID (online or not) whose goal is to identify the parameters, our goal is simply to give accurate predictions for the next timestep, again closer to the goal of learning. This allows our method to not rely on the specific meanings of the parameters and instead work with general model-based controllers. Another relevant subfield is L1 adaptive control ([20], [21]), which, like our work, concerns the residual dynamics, but does not use learning.

4) Online learning: Our work performs supervised learning online, which has long been a subject of research in machine learning ([22], [23], [24]). The two central questions are: where does the label come from, and how is learning evaluated. Traditionally ([25]), learning has been evaluated with regret, and labels can come from a potentially adversarial oracle. Recently, the computer vision community has been using self-supervised tasks to provide labels ([26], [27], [28], [29], [30]), and the continual learning community has been evaluating with forward and backward predictions ([4]) c.f. Subsection II-B.

B. Conventions

In this paper, vectors $(\mathbf{a}, \boldsymbol{\alpha})$ are bold and lowercase, matrices $(\mathbf{A}, \boldsymbol{\Omega})$ are bold and uppercase, scalars and functions (of all type signatures) are not bold. We assemble matrices and vectors like in MATLAB: $[\mathbf{A}, \mathbf{B}]$ concatenates $\mathbf{A}$ and $\mathbf{B}$ horizontally with a comma, and $[\mathbf{A}; \mathbf{B}]$ concatenates them vertically with a semicolon. $\mathbf{0}_n$ denotes the $n \times n$ matrix of zeros, and $\mathbf{1}_n$ denotes the $n \times n$ identity matrix. Also, $\|\cdot\|$ denotes the 2-norm for vectors (Euclidean norm) and matrices (spectral norm), unless stated otherwise. We express quantities in the nominal dynamics $\bar{\alpha}$ with a bar, in the residual dynamics $\tilde{\alpha}$ with a tilde, and in the true (plant) dynamics α without anything on top.

II. METHOD

A. Unknown Dynamics and Linear Residual Models

Given a robotic system that is characterized by rigid-body dynamics, we denote $\mathbf{x} \in \mathbb{R}^n$ as its state, $\mathbf{u} \in \mathbb{R}^m$ its vector of control inputs, and $\mathbf{y} \in \mathbb{R}^d$ its vector of outputs. The output dynamics can almost always be written as a second-order system of the following form ([31]), known as control-affine ([32]):

$$\ddot{\mathbf{y}} = \bar{\alpha}(\mathbf{x})\mathbf{u} + \bar{\beta}(\mathbf{x}). \tag{1}$$

We consider model-based controllers whose goal is to drive the vector of tracking errors $\boldsymbol{\eta} = [\mathbf{y}; \dot{\mathbf{y}}]$ to zero.

The bars on top of the variables imply that they come from our assumed *nominal model*, which in reality can never be completely accurate. The unknown real-world dynamics are called the *true (plant) model*, denoted without the bars as α, β. We often use an alternative set of notations to write equation (1) simply as:

$$\ddot{\mathbf{y}} = \bar{\boldsymbol{\alpha}}\mathbf{u} + \bar{\boldsymbol{\beta}}, \tag{2}$$

in order to emphasize the role of $\bar{\boldsymbol{\alpha}}$ and $\bar{\boldsymbol{\beta}}$ as time-varying parameters of the output dynamics.

To make corrections to the nominal model, we incorporate two residual parameters and obtain the following form:

$$\ddot{\mathbf{y}} = (\bar{\boldsymbol{\alpha}} + \tilde{\boldsymbol{\alpha}})\,\mathbf{u} + \left(\bar{\boldsymbol{\beta}} + \tilde{\boldsymbol{\beta}}\right), \tag{3}$$

where $\tilde{\boldsymbol{\alpha}}$ is called the *weight* and $\tilde{\boldsymbol{\beta}}$ is called the *bias*. They are written as time-varying parameters, and have the same dimensions as $\bar{\boldsymbol{\alpha}}$ and $\bar{\boldsymbol{\beta}}$ respectively. The tildes on top of them emphasize that they are estimated from data.

To better understand these residual parameters, we manipulate equation (3) into:

$$\ddot{\mathbf{y}} - \left(\bar{\boldsymbol{\alpha}}\mathbf{u} + \bar{\boldsymbol{\beta}}\right) = \tilde{\boldsymbol{\alpha}}\mathbf{u} + \tilde{\boldsymbol{\beta}}. \tag{4}$$

Intuitively, the above equation says that the goal of learning is to make the *residual model* on the right-hand side account for the prediction errors of the nominal model on the left-hand side. It also reveals the role of labels vs. covariates, as we explain next in the context of learning.

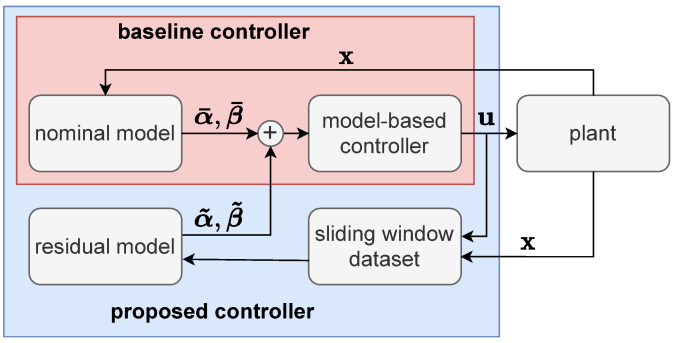

Fig. 2. **Block diagram of our method.** $\bar{\boldsymbol{\alpha}}$ and $\bar{\boldsymbol{\beta}}$ are time-varying parameters of the nominal model for a system's output dynamics, assumed to be control-affine. As the model-based controller is running, data are collected into the sliding window dataset, and supervised learning is performed to estimate residual parameters $\tilde{\boldsymbol{\alpha}}$ and $\tilde{\boldsymbol{\beta}}$; they are then used to improve the model behind the controller. See Section II for more details.

B. Data Collection and Online Learning

For real systems, sensor data can only be collected at discrete sampling intervals. We denote each sampling timestep by an integer subscript, which converts equation (4) into:

$$\ddot{\mathbf{y}}_t - \left(\bar{\boldsymbol{\alpha}}_t\mathbf{u}_t + \bar{\boldsymbol{\beta}}_t\right) = \tilde{\boldsymbol{\alpha}}_t\mathbf{u}_t + \tilde{\boldsymbol{\beta}}_t. \tag{5}$$

Note that we are merely sampling a continuous system at discrete timesteps, so continuous-time concepts such as acceleration are still well defined. We collect a dataset of the form $\mathcal{D} := \{\text{label}_s, \text{covariate}_s\}_{s=t-k,\dots,t-1}$, where s is the index of discrete timesteps, and k denotes the fixed size of the sliding time window. From equation (5), we have

$$\mathcal{D} = \left\{\ddot{\mathbf{y}}_s - \left(\bar{\boldsymbol{\alpha}}_s \mathbf{u}_s + \bar{\boldsymbol{\beta}}_s\right), \mathbf{u}_s\right\}_{s=t-k,\dots,t-1}.$$

Given a dataset, our method solves regularized least squares a.k.a. ridge regression on the labels and covariates. The weight of the solution is $\tilde{\boldsymbol{\alpha}}_t$, and bias is $\tilde{\boldsymbol{\beta}}_t$. Note that in textbook-style least squares, the weight is a vector, and the label and bias are scalars; for our learning problem, the weight is a matrix in $\mathbb{R}^{d\times m}$, and the label and bias are vectors in $\mathbb{R}^d$. But we can simply reduce this to d independent vector-scalar least squares problems. The same regularization is added independently to these d problems, since they share the same covariates; thus inversion of the covariance matrix, the most computationally costly step, is only performed once.

The solved parameters are then immediately used by the model-based controller to produce $\mathbf{u}_t$. In both of our later examples, the baseline controller solves for $\mathbf{u}_t$ in an application-specific optimization problem with the assumed nominal parameters $\bar{\boldsymbol{\alpha}}_t$ and $\bar{\boldsymbol{\beta}}_t$. We simply substitute these with $\bar{\boldsymbol{\alpha}}_t + \tilde{\boldsymbol{\alpha}}_t$ and $\bar{\boldsymbol{\beta}}_t + \tilde{\boldsymbol{\beta}}_t$ respectively, as shown in Figure 2.

Learning is performed online, as the controller is running with the learned parameters. At the beginning, all residual parameters are initialized to zero, because there is not enough data to learn them. Once we are k steps into the trajectory, we have enough data to form $\mathcal{D}$ as above and solve for the residual parameters; informed by them, the controller generates an improved trajectory, which in turn generates new data that are more relevant as time goes on.

The fact that $\mathcal{D}$ only keeps the k most recent data points implements a natural forgetting mechanism. In reinforcement learning terms, $\mathcal{D}$ is called the *replay buffer*, which stores the off-policy data that are not generated by the current controller; in our case, data in $\mathcal{D}$ are generated by the old controllers using the residual parameters from previous timesteps. Because we learn small, local models, we *encourage forgetting* so that our model capacity can be used only for the neighborhood of our current state. This is in contrast to the vast literature in reinforcement learning [33], [24], [34], [4], where the goal is to learn a large, global model; there the replay buffer contains as much historical data as possible, and various techniques are implemented to *discourage forgetting*.

Our method can also be viewed as bootstrapping from a "bad" controller based on an inaccurate model to a better one. This might not be feasible, however, if the initial model deviates too much from the plant. For example, if the nominal model is so far off that the robot loses balance immediately, no useful information will be contained in the data collected. Fortunately, when deviations happen gradually over time, there will more likely be enough information for learning to maintain a controller that keeps generating useful data. We study this phenomenon empirically in Section IV.

C. Theoretical Analysis

Suppose the true (plant) output dynamics is control-affine:

$$\ddot{\mathbf{y}}_t = \boldsymbol{\alpha}_t \mathbf{u}_t + \boldsymbol{\beta}_t. \tag{6}$$

We prove that our method stabilizes the tracking errors under two assumptions. The main theorem illustrates our intuition of learning in a local time window under smoothly varying dynamics, and characterizes the role of k, our window size.

Denote errors in the nominal model's prediction as

$$\hat{\ddot{\mathbf{y}}}_t := \ddot{\mathbf{y}}_t - \left(\bar{\boldsymbol{\alpha}}_t \mathbf{u}_t + \bar{\boldsymbol{\beta}}_t\right) = \hat{\boldsymbol{\alpha}}_t \mathbf{u}_t + \hat{\boldsymbol{\beta}}_t, \tag{7}$$

with $\hat{\boldsymbol{\alpha}}_t := \boldsymbol{\alpha}_t - \tilde{\boldsymbol{\alpha}}_t$, and $\hat{\boldsymbol{\beta}}_t := \boldsymbol{\beta}_t - \tilde{\boldsymbol{\beta}}_t$.

Denote the prediction of the residual model as

$$\tilde{\ddot{\mathbf{y}}}_t := \tilde{\boldsymbol{\alpha}}_t \mathbf{u}_t + \tilde{\boldsymbol{\beta}}_t. \tag{8}$$

Assumption 1: The model-based controller can stabilize the tracking errors $\boldsymbol{\eta} = [\mathbf{y}; \dot{\mathbf{y}}]$ if for some $\epsilon > 0$,

$$\left\| \ddot{\mathbf{y}}_t - \left(\left(\bar{\boldsymbol{\alpha}}_t + \tilde{\boldsymbol{\alpha}}_t\right) \mathbf{u}_t + \left(\bar{\boldsymbol{\beta}}_t + \tilde{\boldsymbol{\beta}}_t\right) \right) \right\| < \epsilon. \tag{9}$$

Assumption 2: $\|\hat{\boldsymbol{\alpha}}_{t+1} - \hat{\boldsymbol{\alpha}}_t\| < \delta_\alpha$, $\|\hat{\boldsymbol{\beta}}_{t+1} - \hat{\boldsymbol{\beta}}_t\| < \delta_\beta$.

In words, Assumption 1 says that the proposed model-based controller works when the proposed (nominal plus residual) model is relatively accurate; Assumption 2 says that the deviations in dynamics are relatively smooth (in the space of parameters) over time.

In addition, we denote the motor torque saturation as $\|\mathbf{u}\| < B$. Denote $\mathbf{u}'_t = [\mathbf{u}_t; 1] \in \mathbb{R}^{m+1}$, and

$$\mathbf{U}' = \left[[\mathbf{u}_1; 1]^\top; \dots; [\mathbf{u}_k; 1]^\top \right] \in \mathbb{R}^{k\times(m+1)}. \tag{10}$$

We set $k \geq m+1$, so $\sigma_{\min}(\mathbf{U}') > 0$, i.e. the covariance matrix of ordinary least squares (OLS) has rank $m+1$.

Theorem 1: Given the above assumptions, if

$$\frac{(B+1)^2\sqrt{d}}{\sigma_{\min}(\mathbf{U}')} k\sqrt{k}(\delta_\alpha + \delta_\beta) < \epsilon, \tag{11}$$

then the model-based controller stabilizes $\boldsymbol{\eta}$.

Note that any claim of stability in Theorem 1 is completely inherited from the baseline controller, when Assumption 1 holds. Our method is agnostic to the exact type of stability e.g. exponential / asymptotic, which depends on the underlying baseline, and is orthogonal to the theory we develop.

In Theorem 1, B, d, δ_α and δ_β are constants determined by the application. ϵ is the model-based controller's tolerance for model inaccuracy, also independent of our method. The only quantity we tune is k, the window size, which strongly affects $\sigma_{\min}(\mathbf{U}')$. With a large k, we pay a factor of $k\sqrt{k}$, intuitively due to the lag in our dataset. With a small k, we pay for the decrease in $\sigma_{\min}(\mathbf{U}')$, as $\tilde{\boldsymbol{\alpha}}$ and $\tilde{\boldsymbol{\beta}}$ become more sensitive to noise. The user should tune k to find a sweet spot in the middle. In practice, we use regularized least squares instead of OLS, so $\sigma_{\min}(\mathbf{U}')$ is always > 0 and more noise tolerant, making the balance less delicate w.r.t. choice of k. We use $k = 100$ in both of our applications (100 and 200 ms respectively).

Before proving Theorem 1, we state two lemmas, whose proofs are given in Subsection A of the appendix.

Lemma 1: For $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $\mathbf{b} \in \mathbb{R}^m$, if $\|\mathbf{A}\| \leq \delta_A$ and $\|\mathbf{b}\| \leq \delta_b$, then $\|[\mathbf{A}, \mathbf{b}]\| \leq \delta_A + \delta_b$.

Lemma 2: Let $\mathbf{y}_t \in \mathbb{R}^d$, $\mathbf{u}_t \in \mathbb{R}^m$ and $\mathbf{A}_t \in \mathbb{R}^{d\times m}$. Let $\mathbf{y}_t = \mathbf{A}_t\mathbf{u}_t$ for $t = 1, ..., k$, and $\tilde{\mathbf{A}}$ be the OLS estimator of the dataset $\{(\mathbf{y}_1, \mathbf{u}_1), ..., (\mathbf{y}_k, \mathbf{u}_k)\}$. If for $t = 1, ..., k+1$, $\|\mathbf{A}_{t+1} - \mathbf{A}_t\| < \delta_A$, and $\|\mathbf{u}_t\| < B$, then

$$\left\|\mathbf{A}_t - \tilde{\mathbf{A}}_t\right\| < \frac{B\sqrt{d}}{\sigma_{\min}(\mathbf{U})} k\sqrt{k}\delta_A, \tag{12}$$

where $\mathbf{U} = [\mathbf{u}_1^T; ...; \mathbf{u}_k^T] \in \mathbb{R}^{k\times m}$.

Proof of Theorem 1: By triangle inequality, we have $\|\mathbf{u}'_t\| < B + 1$. Also define $\hat{\mathbf{A}}_t = [\hat{\boldsymbol{\alpha}}_t, \hat{\boldsymbol{\beta}}_t] \in \mathbb{R}^{d\times m+1}$, and similarly $\tilde{\mathbf{A}}_t = [\tilde{\boldsymbol{\alpha}}_t, \tilde{\boldsymbol{\beta}}_t]$. Combining Assumption 2 and Lemma 1, we have $\|\hat{\mathbf{A}}_{t+1} - \hat{\mathbf{A}}_t\| < \delta_\alpha + \delta_\beta$. Now

$$\left\|\ddot{\mathbf{y}}_t - \left(\left(\bar{\boldsymbol{\alpha}}_t + \tilde{\boldsymbol{\alpha}}_t\right)\mathbf{u}_t + \left(\bar{\boldsymbol{\beta}}_t + \tilde{\boldsymbol{\beta}}_t\right)\right)\right\| \tag{13}$$

$$= \left\|\hat{\ddot{\mathbf{y}}}_t - \tilde{\ddot{\mathbf{y}}}_t\right\| = \left\|\left(\hat{\boldsymbol{\alpha}}_t - \tilde{\boldsymbol{\alpha}}_t\right)\mathbf{u}_t + \left(\hat{\boldsymbol{\beta}}_t - \tilde{\boldsymbol{\beta}}_t\right)\right\| \tag{14}$$

$$= \left\|\left(\hat{\mathbf{A}}_t - \tilde{\mathbf{A}}_t\right)\mathbf{u}'_t\right\| \leq (B+1)\left\|\hat{\mathbf{A}}_t - \tilde{\mathbf{A}}_t\right\|. \tag{15}$$

By definition, $\tilde{\mathbf{A}}_t$ is the least squares solution on $\mathcal{D}$. We then apply Assumption 1 and Lemma 2 to finish the proof.

III. APPLICATIONS

We now apply our method to two model-based controllers, derived from two different perspectives for different robotic platforms: a Lyapunov perspective to control the full-order dynamics of bipedal robots, and a simplified dynamics based control architecture for robust quadrupedal locomotion. We focus on identifying the components of our method in the context of each controller, without elaborating on derivations of the nominal dynamics.

A. CLF-QP for Bipedal Locomotion

Let $\mathbf{q}$ be the robot's configuration, and $\mathbf{x} = [\mathbf{q}; \dot{\mathbf{q}}]$ be the robot's state. We define $\mathbf{y} = h(\mathbf{x})$, where h is called the *virtual constraints* ([35]). For a biped, stabilizing $\boldsymbol{\eta} = [\mathbf{y}; \dot{\mathbf{y}}]$ means, for example, that the torso maintains a constant posture, and the legs walk in a scissor-symmetric gait.

The nominal output dynamics, whose derivation we omit, can then be written in the familiar form of equation (1), using Lie-derivatives of the nominal dynamics in the state space as $\bar{\alpha}$ and $\bar{\beta}$:

$$\ddot{\mathbf{y}} = \underbrace{\frac{d}{dt}\left(\frac{\partial h}{\partial \mathbf{q}}\right)\dot{\mathbf{q}} - \frac{\partial h}{\partial \mathbf{q}}\bar{\mathbf{D}}\left(\bar{\mathbf{C}}\dot{\mathbf{q}} + \bar{\mathbf{g}}\right)}_{\bar{\beta}(\mathbf{x})} + \underbrace{\frac{\partial h}{\partial \mathbf{q}}\bar{\mathbf{D}}\mathbf{B}}_{\bar{\alpha}(\mathbf{x})}\mathbf{u}, \tag{16}$$

where $\bar{\mathbf{D}}$ is the inverse of the mass-inertia matrix, $\bar{\mathbf{C}}$ is the Coriolis matrix and $\bar{\mathbf{g}}$ is the gravity vector. While $\bar{\alpha}(\mathbf{x})$ might not be square ($d = m$) in general, this particular bipedal controller has the same number of virtual constraints as actuated joints. Now the control law $\mathbf{u} = \bar{\alpha}(\mathbf{x})^{-1}\left(-\bar{\beta}(\mathbf{x}) + \mathbf{v}\right)$, a.k.a. *input-output (I/O) linearization*, produces $\ddot{\mathbf{y}} = \mathbf{v}$.

We can then design $\mathbf{v}$ to stabilize the output dynamics using *control Lyapunov functions (CLFs)*, a common tool in control theory for providing stability guarantees in legged locomotion ([36]). Because $\dot{\boldsymbol{\eta}}$ is linear in $\boldsymbol{\eta}$ and $\mathbf{v}$, it is straightforward to find a CLF by solving the Lyapunov equation $V(\boldsymbol{\eta})$ ([37]). It is then a well known fact that $\dot{V}(\boldsymbol{\eta}, \mathbf{v}) < -c\mathbf{V}$ implies exponential stability of $\boldsymbol{\eta}(t)$, with a constant $c > 0$. This motivates the following CLF-based quadratic program (CLF-QP) to solve for $\mathbf{v}$:

$$\begin{aligned} \mathbf{v}(\mathbf{x}) = \underset{\mathbf{v}}{\operatorname{argmin}} \quad & \mathbf{u}^\top\mathbf{u} \\ \text{s.t.} \quad & \mathbf{C1}.\ \dot{V}(\boldsymbol{\eta}, \mathbf{v}) < -c\mathbf{V} \\ & \mathbf{C2}.\ \mathbf{u} = \bar{\alpha}(\mathbf{x})^{-1}\left(-\bar{\beta}(\mathbf{x}) + \mathbf{v}\right), \\ & \mathbf{C3}.\ \mathbf{u}_{\min} \preceq \mathbf{u} \preceq \mathbf{u}_{\max}, \end{aligned} \tag{17}$$

where $\mathbf{u}_{\min}$ and $\mathbf{u}_{\max}$ are bounds of the torque saturation constraints. Since the output dynamics is already in the form of equation (1), it is straightforward to apply our method to obtain $\tilde{\boldsymbol{\alpha}}$ and $\tilde{\boldsymbol{\beta}}$. We can then modify the $\mathbf{C2}$ in the optimization problem (17) to have

$$\mathbf{u} = \left(\bar{\boldsymbol{\alpha}} + \tilde{\boldsymbol{\alpha}}\right)^{-1}\left(-\left(\bar{\boldsymbol{\beta}} + \tilde{\boldsymbol{\beta}}\right) + \mathbf{v}\right). \tag{18}$$

In Section IV we show that this simple modification leads to significant improvements under uncertain dynamics.

B. MPC with Contact Force for Quadrupedal Locomotion

To control a quadrupedal system walking stably under large disturbance (such as heavy loads), we take the *model predictive control* (MPC) approach using the simplified dynamics from [38] as our baseline controller.

For quadrupedal dynamics, let $\mathbf{p}, \dot{\mathbf{p}}, \ddot{\mathbf{p}} \in \mathbb{R}^6$ be the position, velocity and acceleration of the robot's center of mass (CoM). Let $\mathbf{f}_i \in \mathbb{R}^3$ be the ground reaction force at the robot's i^{th} foot, with $i \in \{1, 2, 3, 4\}$. We also denote $\mathbf{f} = [\mathbf{f}_1; \mathbf{f}_2; \mathbf{f}_3; \mathbf{f}_4] \in \mathbb{R}^{12}$. The nominal dynamics of the CoM is given by

$$\ddot{\mathbf{p}} = \bar{\mathbf{D}}\mathbf{G}\mathbf{f} - \bar{\mathbf{g}}, \tag{19}$$

where $\bar{\mathbf{g}} \in \mathbb{R}^6$ is the gravity vector, $\bar{\mathbf{D}} \in \mathbb{R}^{6\times 6}$ is the inverse mass matrix, and $\mathbf{G} \in \mathbb{R}^{6\times 12}$ is called the grasp map, which depends on the robot's state and is assumed to be accurate.

The goal of the model-based controller is to have $\mathbf{p}$ and $\dot{\mathbf{p}}$ track the desired position and velocity $\mathbf{p}_d$ and $\dot{\mathbf{p}}_d$, generated from user command. In Sec.II notations, $\mathbf{y} = \mathbf{p} - \mathbf{p}_d$, we want to stabilize $\boldsymbol{\eta} = [\mathbf{y}, \dot{\mathbf{y}}]$ around zero. This is achieved by having $\ddot{\mathbf{p}}$ track some desired acceleration $\ddot{\mathbf{p}}_d$, generated from PD control on $\mathbf{p}_d$ and $\dot{\mathbf{p}}_d$. The model-based controller then uses equation (19) to solve for $\mathbf{f}$:

$$\begin{aligned} \underset{\mathbf{f}}{\operatorname{argmin}} \quad & \|\bar{\mathbf{D}}\mathbf{G}\mathbf{f} - \bar{\mathbf{g}} - \ddot{\mathbf{p}}_d\|_{\mathbf{Q}} + \|\mathbf{f}\|_{\mathbf{R}} \\ \text{s.t.} \quad & \text{stance and swing leg constraints,} \\ & \text{friction pyramid condition.} \end{aligned} \tag{20}$$

where more details can be found in [39], Following the outline in Section II, we modify (19) to incorporate the linear residual model:

$$\ddot{\mathbf{p}} = \left(\bar{\mathbf{D}} + \tilde{\mathbf{D}}\right)\mathbf{G}\mathbf{f} - \left(\bar{\mathbf{g}} + \tilde{\mathbf{g}}\right), \tag{21}$$

where $\tilde{\mathbf{D}}$ is the weight, and $-\tilde{\mathbf{g}}$ is the bias.

Note that the nominal dynamics in (19) has no Coriolis terms, a simplification often adopted in the literature for model-based controller design of quadrupeds with small angular velocity. While this simplification has been validated in many implementations, it is never completely accurate. Therefore, even if $\bar{\mathbf{D}} = \mathbf{D}$ and $\bar{\mathbf{g}} = \mathbf{g}$ i.e. they are both accurate parameters, (19) is still an inaccurate description of the plant. We make no distinction, philosophically or algorithmically, between unknown dynamics e.g. payload, and unmodeled dynamics e.g. the Coriolis terms discarded by design. Our true output dynamics can take any general form. Also note that Assumption 1 is in fact not satisfied by our baseline controller due to its simplifications e.g. massless legs. In this case, stability is left to empirical validation.

Moving on, we sample equation (21) at discrete timesteps:

$$\ddot{\mathbf{p}}_t - \left(\bar{\mathbf{D}}\mathbf{G}_t\mathbf{f}_t - \bar{\mathbf{g}}\right) = \tilde{\mathbf{D}}_t\mathbf{G}_t\mathbf{f}_t - \tilde{\mathbf{g}}_t, \tag{22}$$

and form the dataset as

$$\mathcal{D}_q = \left\{\ddot{\mathbf{p}}_s - \left(\bar{\mathbf{D}}\mathbf{G}_s\mathbf{f}_s - \bar{\mathbf{g}}\right), \mathbf{G}_s\mathbf{f}_s\right\}. \tag{23}$$

After solving for $\tilde{\mathbf{D}}$ and $\bar{\mathbf{g}}$, we use them to modify the objective function in equation (20) as:

$$\min_{\mathbf{f}} \quad \left\| \left(\bar{\mathbf{D}} + \tilde{\mathbf{D}}\right)\mathbf{G}\mathbf{f} - (\bar{\mathbf{g}} + \tilde{\mathbf{g}}) - \ddot{\mathbf{p}}_d \right\|_{\mathbf{Q}} + \|\mathbf{f}\|_{\mathbf{R}}. \tag{24}$$

By definition, $\bar{\mathbf{D}} + \tilde{\mathbf{D}}$ must be positive definite; this is also necessary for the optimization problem above to make sense. For computational efficiency, we solve for $\tilde{\mathbf{D}}$ unconstrained, and find that our least squares solution in fact always gives $\bar{\mathbf{D}} + \tilde{\mathbf{D}}$ positive definite for our experiments.

IV. RESULTS

Video of our experiments is available at https://youtu.be/Je_2Y-FQpKw ([40]). Simulations are performed in the PyBullet ([41]) physics engine.

A. Simulation for Bipedal Walking

Our baseline controller discussed in Subsection III-A is taken from [33], which introduces its own setting and method for unknown dynamics. We perform simulation in their setting, and make comparison with their method.

The problem setting is based on RABBIT ([42]), an underactuated planar five-link bipedal robot with seven degree-of-freedom; virtual constraints and controller design are based on [37]. Model uncertainty is introduced in [33] by scaling the mass of each link by a factor of two in the real environment. The baseline CLF-QP controller falls in a few steps in this setting, due to the significant difference in dynamics between the nominal and true model.

By querying the plant, [33] uses model-free reinforcement learning (RL) to train a policy that directly adds on the original control inputs $\mathbf{u}$, without reasoning about the unknown dynamics in the model space. Specifically, the commanded control inputs take the form $\mathbf{u} + u_{\boldsymbol{\theta}}(\mathbf{x})$, where u is a neural network policy with parameters $\boldsymbol{\theta}$. Their reward is designed

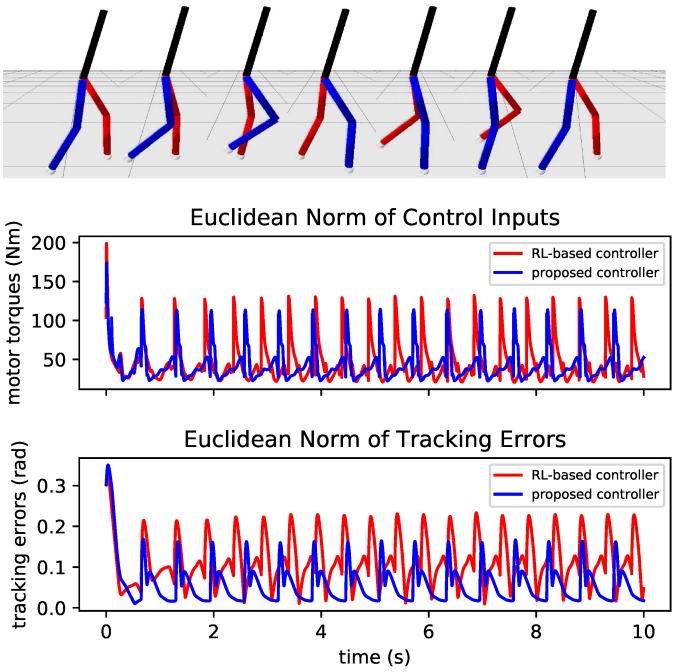

Fig. 3. **Bipedal walking with mass of each link scaled by two.** Both our method and that of [33] walk stably. Their RL-based method trains on 20,000 samples from the real environment before deployment. Our method trains completely online and does not sample from or anticipate the real environment, treating it as truly unknown until the robot is deployed, and results in smaller impulses of control inputs and better tracking performance. The top panel visualizes the gait generated by our method.

to encourage $\dot{V} < -cV$, where the value of V is obtained by simulating in the plant. After 20,000 samples from the plant simulated using the true dynamics, their method trains a policy which walks in the true dynamics without falling.

Our method walks stably in the same setting, training completely online without querying the plant at all before deployment. In fact, Fig. 3 shows that our method enjoys smaller impulses of control inputs and better tracking performance than the RL-based method, even though the latter had privileged access to the plant before deployment to optimize exactly for these metrics.

Online learning enables us to treat the plant as truly unknown, in terms of both data and mathematical representation, while only the latter is unknown for methods that train offline like in [33]. This philosophical difference prevents our controller from overfitting on the training environment. In particular, our controller still walks stably under the original dynamics without scaling, where the policy trained with the scaled links fails, because it overfits to the scaled dynamics.

In addition, our controller walks stably in all environments below, where the baseline and the RL-based method cannot:

1) scaling the control inputs by half, in order to simulate transmission inefficiencies and motor wear and tear;
2) scaling the mass of the torso by four, in order to simulate payload on the back of the humanoid;
3) scaling the mass of the right leg by four, as an example of asymmetric changes in dynamics.

We keep the same hyper-parameters for all the experiments above, including a windows size of 100 ms (where $k = 100$ and each timestep is 1 ms). The robot is still able to walk

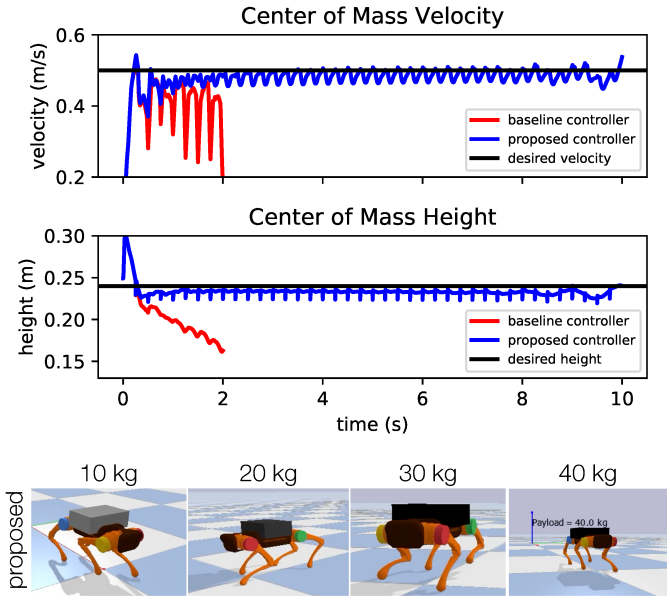

Fig. 4. **Quadruped walking with payload in simulation.** We start with an empty payload, and increase its mass by 5 kg / s once simulation begins. The baseline has completely fallen in 2 s, but the proposed method still walks stably after 10 s (50 kg). The bottom visualizations are captured when the payload reaches the specified mass. The torque limit is reached at 25 kg.

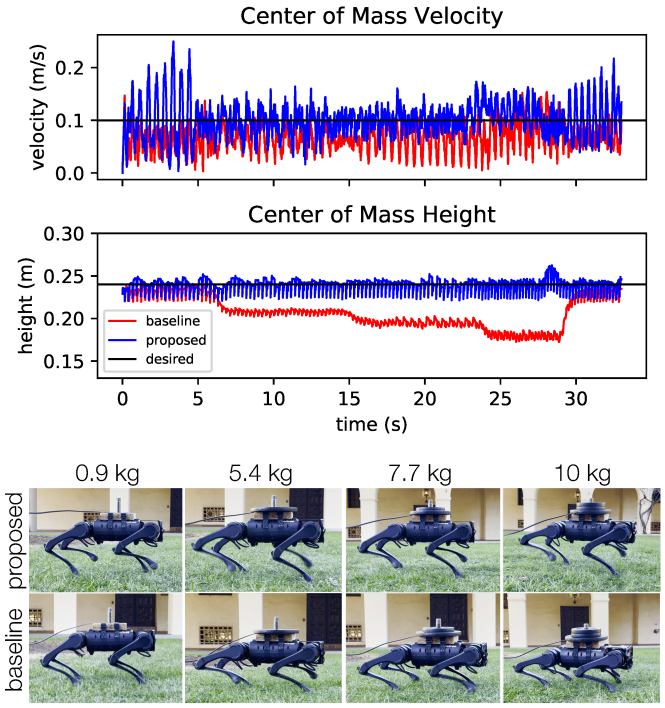

Fig. 5. **Quadruped walking with payload in the real world.**

under the scaled dynamics with a window size of 10 or 1000, but has higher norm of control inputs and tracking errors.

B. Simulation for Quadrupedal Robot

Our baseline controller, as discussed in Subsection III-B, is based on [38] and used subsequently in [39] and [43]. Our implementation is modified from the publicly available code of [43] on an Unitree A1 quadruped, and keeps their original parameters unless stated otherwise. The A1 weighs 12 kg and has 12 motors, three for each leg, with the stated torque limit of 35.5 Nm. We experiment in PyBullet using Unitree's URDF description, and also on a real robot. In both simulation and real world, we use a window size of $k = 100$ (like for the biped); the controller runs at a frequency of 500 Hz, making the dataset window 200 ms.

We command our robot to walk with linear velocity of 0.5 m / s in the x-direction, while maintaining CoM height of 0.24 m. Both the baseline and the proposed method can walk stably without payload, while tracking the desired velocity and height. With 6 kg of payload, however, the baseline can barely walk at 2 / 3 the desired velocity, and sags to 2 / 3 the desired height; the robot falls with 7 kg.

The proposed method walks stably with 12 kg of payload (same as its body mass), while tracking the desired velocity up to 0.05 m / s, and the desired height up to 0.01 m; all motors torques are less than 35.5 Nm. With more than 12 kg, however, tracking becomes less accurate, and with 15 kg the robot falls. Since the payload is carried from very the beginning of simulation, the robot visibly sags for the first fifth of a second, as we collect data before we can estimate the residual parameters. With 12 kg it soon recovers from the sag, but for larger payloads it struggles to get back.

Next, we experiment with gradually changing dynamics. We start with an empty payload, and increase its mass by 5 kg / s, that is, 0.001 kg per timestep, once simulation begins. The tracking errors are shown in Figure 4. The baseline falls within 2 s. We have tried to improve the baseline by tuning the PD gains for $\ddot{\mathbf{p}}_d$, but found it ineffective. This observation is reasonable, since larger gains only make $\ddot{\mathbf{p}}_d$ more aggressive, but cannot help if the model-based controller fails to achieve it using the nominal dynamics. The proposed method walks stably even when the payload reaches 50 kg. Motor torques reach the specified limit at 25 kg (5 s), but the URDF allows simulation to keep running.

C. Hardware Experiments for Quadrupedal Robot

To facilitate hardware testing, we fit the Unitree A1 quadruped with a loading rig designed to hold up three standard 1 inch weight plates. The rig allows for incremental, discrete changes in load while the quadruped is in operation. The rig itself weighs 0.9 kg.

The experiments were designed to compare the performance of the baseline and proposed controllers under varying load conditions during operation. Two tests for each controller were performed: a step-in-place test and a 0.1 m/s forward motion test. The load conditions for the tests are shown in Table 1. Due to the manual loading process, the duration of each load varies by a small amount of transition time, typically less than 1 s. To protect the hardware from possible damage, we do not load beyond 10 kg, and limit operation at this load to 5 s.

In the transition from simulation to hardware, we had to address the problem of acceleration estimation from noisy measurements. [43] uses a Kalman filter to fuse IMU and

TABLE I
LOAD CONDITIONS FOR HARDWARE EXPERIMENTS

Load (kg)	0.9	5.4	7.7	10	0.9
Duration (s)	5	10	10	5	5

joint encoder measurements and produce a CoM velocity measurement. From this, first order difference is then used to compute a CoM acceleration estimate for the learning algorithm. Two parameters for the Kalman filter, namely the window size and IMU variance value, are tuned to give a final acceleration estimate with suitable trade-off between lag and noise. The window size is modified from 120 to 60 samples, and the IMU variance is modified from equal to the encoder variance to 5 times the encoder variance. Ultimately, after tuning, the estimator produces acceptable linear acceleration estimates, but the angular terms proved too noisy to be useful. As such, we proceeded with hardware experiments with learning enabled for only the linear terms.

The hardware experiments were performed on the A1 on flat, grassy terrain. Both the baseline and proposed methods perform nominally with low load, but as the weight increases, the baseline controller sags in body height and is unable to maintain forward velocity. The proposed controller does not suffer this degradation and is able to maintain desired body height and forward velocity for the range of load conditions. Results for the forward walking test are summarized in Figure 5. Video comparison of both trot-in-place and forward walking is available in [40].

V. DISCUSSION

We have introduced a method to update model parameters through online learning, while a controller is running with past, inaccurate versions of these parameters. Unlike methods in adaptive control, our method uses learned parameters that are general functions of the state, thus inherently time-varying. To the best of our knowledge, this is the first method that applies machine learning to make real-time updates (500 to 1000 Hz) in hardware walking experiments.

While the nominal models in our applications are derived from classical mechanics, our method can be applied to any black box nominal model e.g. a simulator. While our baselines are derived from classical control principles, our method can also be applied to any controller using the black box model, even a policy trained in simulation. We hope to explore these potentials in future work, under broader definitions of unknown dynamics, such as sim-to-real transfer.

APPENDIX

A. Proofs

1) Proof of Lemma 1:

$$\begin{aligned}\left\|[\mathbf{A},\mathbf{b}]\right\| &= \max_{\mathbf{x}\in\mathbb{R}^n, y\in\mathbb{R}} \left\|[\mathbf{A},\mathbf{b}]\begin{bmatrix}\mathbf{x}\\ y\end{bmatrix}\right\| \;:\; \left\|\begin{bmatrix}\mathbf{x}\\ y\end{bmatrix}\right\| \le 1\\ &= \max_{\mathbf{x}\in\mathbb{R}^n, y\in\mathbb{R}} \left\|\mathbf{A}\mathbf{x}+y\mathbf{b}\right\| \;:\; \left\|\begin{bmatrix}\mathbf{x}\\ y\end{bmatrix}\right\| \le 1\\ &\le \max_{\|\mathbf{x}\|\le 1} \|\mathbf{A}\mathbf{x}\| + \max_{y\in[-1,1]} \|y\mathbf{b}\|\\ &= \delta_A + \delta_b.\end{aligned}$$

2) Proof of Lemma 2: We first prove the vector version in a claim, which is used in the proof of the lemma.

Claim 1: Consider $y_t = \langle \mathbf{u}_t, \mathbf{a}_t\rangle \in \mathbb{R}$ for $t = 1,\dots,k$, and $\mathbf{u}_t, \mathbf{a}_t \in \mathbb{R}^m$. Suppose $\|\mathbf{u}_t\| < B$ and $\|\mathbf{a}_{t+1} - \mathbf{a}_t\| < \epsilon$. Let $\tilde{\mathbf{a}}$ be the OLS estimator on this dataset, then

$$\|\mathbf{a}_k - \tilde{\mathbf{a}}\| < \frac{B}{\sigma_{\min}(\mathbf{U})} k\sqrt{k}\epsilon$$

.

Proof: Define the *feasible set of weights* for a dataset as

$$\mathcal{A} = \{(\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_k) : y_t = \langle \mathbf{u}_t, \mathbf{a}_t\rangle, \|\mathbf{a}_t - \mathbf{a}_{t+1}\| \le \epsilon, \forall t\}.$$

Then $\mathbf{a}_k$ can only exist in the kth component of $\mathcal{A}$, denoted

$$\mathcal{A}_k = \{\mathbf{a}_k : \exists(\mathbf{a}_1, \cdots, \mathbf{a}_{k-1}) \text{ s.t. } (\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_k) \in \mathcal{A}\}.$$

Our goal is to bound $\max_{\mathbf{a}_k\in\mathcal{A}_k} \|\tilde{\mathbf{a}} - \mathbf{a}_k\|$.

Define $\mathbf{e}_t = \mathbf{a}_t - \mathbf{a}_k$, where $\mathbf{e}_k = 0$ and $\|\mathbf{e}_{t+1} - \mathbf{e}_t\| < \epsilon$. We can rewrite $\mathcal{A}$ using these conditions as

$$\begin{aligned}\mathcal{A} = \{(\mathbf{a}_1, \cdots, \mathbf{a}_k) :\; & y_t = \langle \mathbf{u}_t, \mathbf{a}_t\rangle, \mathbf{e}_t = \mathbf{a}_t - \mathbf{a}_k,\\ & \mathbf{e}_k = 0, \|\mathbf{e}_{t+1} - \mathbf{e}_t\| < \epsilon, \forall t\}.\end{aligned}$$

Define $\mathbf{E} = [\mathbf{e}_1^T; ...; \mathbf{e}_k^T]$ and $\mathbf{A} = [\mathbf{a}_1^T; ...; \mathbf{a}_k^T]$. Also, by definition of OLS, $\tilde{\mathbf{a}} = (\mathbf{U}^T\mathbf{U})^{-1}\mathbf{U}^T\mathbf{y}$. Therefore

$$\begin{aligned}\|\mathbf{a}_k - \tilde{\mathbf{a}}\| &= \max_{\mathbf{a}_k\in\mathcal{A}_k} \|\tilde{\mathbf{a}} - \mathbf{a}_k\|\\ &= \max_{\mathbf{a}_k\in\mathcal{A}_k} \|(\mathbf{U}^T\mathbf{U})^{-1}\mathbf{U}^T\mathbf{y} - \mathbf{a}_k\|\\ &= \max_{\mathbf{a}_k\in\mathcal{A}_k} \|(\mathbf{U}^T\mathbf{U})^{-1}\mathbf{U}^T(\mathbf{U}\circ\mathbf{A}\cdot\mathbf{1}) - \mathbf{a}_k\|\\ &= \max_{\mathbf{a}_k,\mathbf{E}} \|(\mathbf{U}^T\mathbf{U})^{-1}\mathbf{U}^T(\mathbf{U}\mathbf{a}_k + \mathbf{U}\circ\mathbf{E}\cdot\mathbf{1}) - \mathbf{a}_K\|\\ &= \max_{\mathbf{E}} \|(\mathbf{U}^T\mathbf{U})^{-1}\mathbf{U}^T(\mathbf{U}\circ\mathbf{E}\cdot\mathbf{1})\|\\ &\le \|(\mathbf{U}^T\mathbf{U})^{-1}\mathbf{U}^T\| \max_{\mathbf{E}} \|\mathbf{U}\circ\mathbf{E}\cdot\mathbf{1}\|\\ &= \frac{\max_{\mathbf{E}} \|\mathbf{U}\circ\mathbf{E}\cdot\mathbf{1}\|_2}{\sigma_{\min}(\mathbf{U})},\end{aligned}$$

where $\circ$ denotes the Hadamard operator, $\sigma_{\min}(\mathbf{U})$ is the minimum non-zero singular value of $\mathbf{U}$; the last equality follows from singular value decomposition of $\mathbf{U}$. Note that

$$\begin{aligned}\max_{\mathbf{E}} \|\mathbf{U}\circ\mathbf{E}\cdot\mathbf{1}\| &< \max_{t=1,\dots,k} \|\mathbf{u}_t\| \|[0;\cdots;k-1]\|\epsilon\\ &< \frac{\sqrt{3}B}{3} k\sqrt{k}\epsilon < Bk\sqrt{k}\epsilon,\end{aligned}$$

which finishes the proof of Claim 1.

Now we extend the result of Claim 1 to prove Lemma 2, Note that in the context of Lemma 2, $\mathbf{A}_t \in \mathbb{R}^{d\times m}$, and is different from the definition of $\mathbf{A}$ in the proof of Claim 1. We use the standard matrix norm relationship ([44])

$$\|\mathbf{X}\|/\sqrt{d} \le \|\mathbf{X}\|_{2\to\infty} \le \|\mathbf{X}\|, \tag{25}$$

for any matrix $\mathbf{X} \in \mathbb{R}^{d\times m}$. Combining the second half of equation (25) with the lemma's assumption, we have

$$\|\mathbf{A}_{t+1} - \mathbf{A}_t\|_{2\to\infty} < \epsilon. \tag{26}$$

As explained in Subsection II-B, the matrix-vector least squares problem is solved by reducing to d independent vector-scalar sub-problems, for each dimension of $\mathbf{y}_t$. Each sub-problem solves for one row of $\tilde{\mathbf{A}}$. From equation (26), we

already know that rows of the ground truth weight matrices satisfy the smoothness assumption in Claim 1. Therefore we can apply Claim 1 to each row of $\tilde{\mathbf{A}}$, yielding

$$\|\mathbf{A}_k - \tilde{\mathbf{A}}\|_{2\to\infty} < \frac{B}{\sigma_{\min}(\mathbf{U})} k\sqrt{k}\epsilon.$$

Combining this with the first half of equation (25) finishes the proof of Lemma 2.

ACKNOWLEDGMENT

We thank Fernando Castañeda for providing his code of the method in [33] to compare with our method. Yu Sun would like to thank his advisors, Alexei A. Efros and Moritz Hardt, for their unwavering support, and Armin Askari, Zihao Chen, Ashish Kumar, John Miller, and Haozhi Qi, for their help.

REFERENCES

[1] K. J. Åström and P. Eykhoff, "System identification—a survey," *Automatica*, vol. 7, no. 2, pp. 123–162, 1971.

[2] K. Ayusawa, G. Venture, and Y. Nakamura, "Identification of humanoid robots dynamics using floating-base motion dynamics," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2008, pp. 2854–2859.

[3] M. Mistry, S. Schaal, and K. Yamane, "Inertial parameter estimation of floating base humanoid systems using partial force sensing," in *2009 9th IEEE-RAS International Conference on Humanoid Robots*. IEEE, 2009, pp. 492–497.

[4] Z. Li and D. Hoiem, "Learning without forgetting," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 12, pp. 2935–2947, 2017.

[5] P. Agrawal, A. Nair, P. Abbeel, J. Malik, and S. Levine, "Learning to poke by poking: Experiential learning of intuitive physics," *NIPS*, 2016.

[6] P. W. Battaglia, R. Pascanu, M. Lai, D. Rezende, and K. Kavukcuoglu, "Interaction networks for learning about objects, relations and physics," *NIPS*, 2016.

[7] J. Wu, I. Yildirim, J. J. Lim, B. Freeman, and J. Tenenbaum, "Galileo: Perceiving physical object properties by integrating a physics engine with deep learning," *Advances in neural information processing systems*, vol. 28, pp. 127–135, 2015.

[8] M. Janner, S. Levine, W. T. Freeman, J. B. Tenenbaum, C. Finn, and J. Wu, "Reasoning about physical interactions with object-oriented prediction and planning," *arXiv preprint arXiv:1812.10972*, 2018.

[9] B. Amos, I. D. J. Rodriguez, J. Sacks, B. Boots, and J. Z. Kolter, "Differentiable mpc for end-to-end planning and control," *arXiv preprint arXiv:1810.13400*, 2018.

[10] H. Qi, X. Wang, D. Pathak, Y. Ma, and J. Malik, "Learning long-term visual dynamics with region proposal interaction networks," *ICLR*, 2021.

[11] A. Nagabandi, K. Konolige, S. Levine, and V. Kumar, "Deep dynamics models for learning dexterous manipulation," in *Conference on Robot Learning*. PMLR, 2020, pp. 1101–1112.

[12] K. J. Åström and B. Wittenmark, *Adaptive control*. Courier Corporation, 2013.

[13] J.-J. E. Slotine, W. Li *et al.*, *Applied nonlinear control*. Prentice hall Englewood Cliffs, NJ, 1991, vol. 199, no. 1.

[14] J.-J. E. Slotine and W. Li, "On the adaptive control of robot manipulators," *The international journal of robotics research*, vol. 6, no. 3, pp. 49–59, 1987.

[15] R. Ortega and M. W. Spong, "Adaptive motion control of rigid robots: A tutorial," *Automatica*, vol. 25, no. 6, pp. 877–888, 1989.

[16] H. Berghuis, H. Roebbers, and H. Nijmeijer, "Experimental comparison of parameter estimation methods in adaptive robot control," *Automatica*, vol. 31, no. 9, pp. 1275–1285, 1995.

[17] S. Jin, C. Wang, and M. Tomizuka, "Robust deformation model approximation for robotic cable manipulation," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 6586–6593.

[18] Z. Kuang, X. Zhang, L. Sun, H. Gao, and M. Tomizuka, "Feedback-based digital higher-order terminal sliding mode for 6-dof industrial manipulators," *American Control Conference*, 2021.

[19] G. Tournois, M. Focchi, A. Del Prete, R. Orsolino, D. G. Caldwell, and C. Semini, "Online payload identification for quadruped robots," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 4889–4896.

[20] Q. Nguyen and K. Sreenath, "L1 adaptive control for bipedal robots with control lyapunov function based quadratic programs," in *2015 American Control Conference (ACC)*. IEEE, 2015, pp. 862–867.

[21] N. Hovakimyan and C. Cao, *L1 adaptive control theory: Guaranteed robustness with fast adaptation*. SIAM, 2010.

[22] S. Shalev-Shwartz *et al.*, "Online learning and online convex optimization," *Foundations and trends in Machine Learning*, vol. 4, no. 2, pp. 107–194, 2011.

[23] L. Bottou and Y. LeCun, "Large scale online learning," *Advances in neural information processing systems*, vol. 16, pp. 217–224, 2004.

[24] A. Nagabandi, C. Finn, and S. Levine, "Deep online learning via meta-learning: Continual adaptation for model-based rl," *ICLR 2019*, 2019.

[25] E. Hazan, "Introduction to online convex optimization," *Foundations and Trends in Optimization: Vol. 2: No. 3-4, pp 157-325*, 2019.

[26] Y. Sun, E. Tzeng, T. Darrell, and A. A. Efros, "Unsupervised domain adaptation through self-supervision," *arXiv preprint arXiv:1909.11825*, 2019.

[27] Y. Sun, X. Wang, Z. Liu, J. Miller, A. A. Efros, and M. Hardt, "Test-time training for out-of-distribution generalization," 2019.

[28] Y. Sun, X. Wang, L. Zhuang, J. Miller, M. Hardt, and A. A. Efros, "Test-time training with self-supervision for generalization under distribution shifts," in *ICML*, 2020.

[29] N. Hansen, R. Jangir, Y. Sun, G. Alenyà, P. Abbeel, A. A. Efros, L. Pinto, and X. Wang, "Self-supervised policy adaptation during deployment," ICLR, 2021.

[30] X. Li, S. Liu, S. De Mello, K. Kim, X. Wang, M.-H. Yang, and J. Kautz, "Online adaptation for consistent mesh reconstruction in the wild," *Neural Information Processing Systems (NeurIPS)*, 2020.

[31] V. Arnold, *Mathematical Methods of Classical Mechanics*, 2nd ed., ser. 60. Springer-Verlag New York, 1989.

[32] R. M. Murray, Z. Li, S. S. Sastry, and S. S. Sastry, *A mathematical introduction to robotic manipulation*. CRC press, 1994.

[33] T. Westenbroek, F. Castañeda, A. Agrawal, S. S. Sastry, and K. Sreenath, "Learning min-norm stabilizing control laws for systems with unknown dynamics," in *2020 59th IEEE Conference on Decision and Control (CDC)*. IEEE, 2020, pp. 737–744.

[34] D. Precup, R. S. Sutton, and S. Dasgupta, "Off-policy temporal-difference learning with function approximation," in *ICML*, 2001, pp. 417–424.

[35] "Models, feedback control, and open problems of 3d bipedal robotic walking," *Automatica*, vol. 50, no. 8, pp. 1955 – 1988, 2014.

[36] W.-L. Ma, N. Csomay-Shanklin, and A. D. Ames, "Coupled control systems: Periodic orbit generation with application to quadrupedal locomotion," *IEEE Control Systems Letters*, vol. 5, no. 3, pp. 935–940, 2021. [Online]. Available: http://ames.caltech.edu/ma2021coupled.pdf

[37] A. D. Ames, K. Galloway, K. Sreenath, and J. W. Grizzle, "Rapidly exponentially stabilizing control lyapunov functions and hybrid zero dynamics," *IEEE Transactions on Automatic Control*, vol. 59, no. 4, pp. 876–891, 2014.

[38] J. Di Carlo, P. M. Wensing, B. Katz, G. Bledt, and S. 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)*. IEEE, 2018, pp. 1–9.

[39] X. Da, Z. Xie, D. Hoeller, B. Boots, A. Anandkumar, Y. Zhu, B. Babich, and A. Garg, "Learning a contact-adaptive controller for robust, efficient legged locomotion," *4th Conference on Robot Learning (CoRL 2020), Cambridge MA, USA.*, 2020.

[40] Supplementary video. https://youtu.be/Je_2Y-FQpKw.

[41] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2019.

[42] C. Chevallereau, G. Abba, Y. Aoustin, F. Plestan, E. Westervelt, C. C. De Wit, and J. Grizzle, "Rabbit: A testbed for advanced control theory," *IEEE Control Systems Magazine*, vol. 23, no. 5, pp. 57–79, 2003.

[43] X. B. Peng, E. Coumans, T. Zhang, T.-W. E. Lee, J. Tan, and S. Levine, "Learning agile robotic locomotion skills by imitating animals," in *Robotics: Science and Systems*, 07 2020.

[44] J. Cape, M. Tang, C. E. Priebe *et al.*, "The two-to-infinity norm and singular subspace geometry with applications to high-dimensional statistics," *Annals of Statistics*, vol. 47, no. 5, pp. 2405–2439, 2019.