Big Data
Volume 4 Number 4, 2016

DOI: 10.1089/big.2016.0041

ORIGINAL ARTICLE

DOOMED: Direct Online Optimization of Modeling Errors in Dynamics

Nathan Ratliff[1], Franziska Meier[1–3,*] Daniel Kappler[1,2] and Stefan Schaal[2,3]

Abstract

It has long been hoped that model-based control will improve tracking performance while maintaining or increasing compliance. This hope hinges on having or being able to estimate an accurate inverse dynamics model. As a result, substantial effort has gone into modeling and estimating dynamics (error) models. Most recent research has focused on learning the true inverse dynamics using data points mapping observed accelerations to the torques used to generate them. Unfortunately, if the initial tracking error is bad, such learning processes may train substantially off-distribution to predict well on actual *observed* acceleration rather than the desired accelerations. This work takes a different approach. We define a class of gradient-based online learning algorithms we term Direct Online Optimization of Modeling Errors in Dynamics (DOOMED) that directly minimize an objective measuring the divergence between actual and desired accelerations. Our objective is defined in terms of the true system's unknown dynamics and is therefore impossible to evaluate. However, we show that its gradient is observable online from system data. We develop a novel adaptive control approach based on running online learning to directly correct (inverse) dynamics errors in real time using the data stream from the robot to accurately achieve desired accelerations during execution.

Keywords: learning control; adaptive control; inverse dynamics; online learning; manipulation; feedback control

Introduction and Motivation

Model-based control is considered essential toward the goal of designing compliant controllers that ideally enable safe interaction with humans, a topic that has received much attention lately. For instance, a typical scenario in which model-based control is considered is the control of robotic manipulators in collaborative settings. In these situations, motion generation is often realized through dynamic systems approaches, allowing to generate a bundle of trajectories (for various starting positions) instead of just one. Dynamic systems approaches are often used to derive kinematic policies, which, given the current state of the system, encode the desired action in the form of accelerations. For instance, one popular realization of such acceleration policies is Dynamic Movement Primitives[1] often used for their ease of training from imitation or reinforcement learning signals; and recently, kinematic linear quadratic regulators (LQRs)[2] are starting to receive increasing attention as flexible and online adaptable acceleration polices for use in dynamically changing environments.

Converting these desired actions—desired accelerations—into motor commands, however, requires an accurate model of the system's inverse dynamics, a key difficulty of this approach. Even if analytical models exist, they typically are not accurate enough throughout the entire state space necessitating substantial feedback control to circumvent, especially since there are often effects that cannot be modeled realistically such as system changes resulting from heat, wear, and cable stretch.

There is a fundamental separation between kinematics, the description of motion, and dynamics, a prescription for how those motions are physically generated in Classical Mechanics.[3] Since *a priori* dynamics models are often necessarily inaccurate because of uncertainties

[1]*Lula Robotics, Inc., Seattle, Washington.*
[2]*Autonomous Motion Department, MPI for Intelligent Systems, Tübingen, Germany.*
[3]*CLMC Lab, University of Southern California, Los Angeles, California.*

*Address correspondence to: *Franziska Meier, CLMC Lab, University of Southern California, 3710 McClintock Avenue, Los Angeles, CA 90089-0001*, E-mail: fmeier@usc.edu

like the ones listed above, planning and control methodologies commonly use a similar decomposition to take or generate first a control signal describing the desired *kinematic* motion, and then utilize feedback control around that signal to aid in prescribing the right control inputs to realize that motion. To do so precisely, the controller needs to correct any dynamics modeling errors in real time. To do that, though, state-of-the-art methodologies for executing acceleration policies[4,5] often start with a trajectory generation step, a process of integrating these policies forward incrementally into integral curve trajectories.

However, this reduces the quite expressive acceleration policy to one kinematic trajectory. The full policy defines how the kinematic state should change anywhere within a relatively large region of the state space and can equivalently be viewed as an entire collection, a continuum, of trajectories, one for each possible integral curve. Selecting just one of them for control means that much of how the policy describing how the system should react in regions around that chosen trajectory is ignored.

For a natural way of making use of the full acceleration policies, we have to address the following two difficulties. First, we need to be able to correct for dynamic modeling errors on the fly, such that we can realize the desired accelerations as accurately as possible. Second, we need to be able to create an error feedback term, without explicitly integrating the policy into desired trajectories.

Note, there has been an effort in bringing the promise of machine learning to the world of model-based control to address the issue of estimating better inverse dynamics (error) models.[6–9] Learning inverse dynamics models from scratch is extraordinarily difficult requiring substantial amounts of data to learn a globally valid model. While acquiring huge amounts of data is simple in this scenario, capturing data throughout the entire state space is not. And even if it were possible, dynamics errors due to wear and tear or task-related changes would not be captured in an offline training data set. Thus, algorithms that can perform online learning, in real time, and constantly adapt existing models have been a focus of recent work in this area.[7,10,11] However, the controllers making use of any of these learning approaches still require a reference trajectory for error feedback control.

In this work, we take a different approach to inverse dynamics error learning, which addresses both the error learning on the fly and the creation of an error feedback term. Our approach leverages, to the greatest extent possible, existing structured mathematical models of the dynamics, especially those built around rigid body assumptions, to build a strong, well-informed, prior model for our learning process. Given that model, we add a data-driven technique on top of it to estimate in real time the error between that prior inverse dynamics model and the torque needed to track the desired signal well. What is different to the previously mentioned methods is the loss function that our online learner optimizes.

The above-cited learning approaches exclusively consider minimizing the loss between the applied torque—the model prediction to achieve desired accelerations in the current state—and the model prediction for the actual measured acceleration. We term these methods indirect loss minimization methods in this work. While, intuitively, they learn the true inverse dynamics model and should thus be exactly what we want to estimate, there are some issues when using this indirect loss. In particular, they train slightly off-distribution in the sense that they train to predict well on actual observed accelerations rather than the desired accelerations.

In this study, we propose instead to explicitly minimize the error between the desired and actual accelerations using online learning. The objective we define is an inertia weighted acceleration error. Because we do not know the systems true inertia matrix, it is impossible to evaluate this objective. However, we show that its gradient is *observable** online, enabling the application of numerous gradient-based online learning approaches. In contrast to the above *indirect loss* approach, this *direct loss* minimization approach is both well posed and able to leverage a long lineage of well-studied online learning methods to adapt quickly and achieve good tracking in a real-time setting. Our proposed algorithm leverages the data streaming from the robot to correct the inverse dynamics model on the fly by updating a correction model until it achieves the right *acceleration*.

Note, it is this switch in loss function that enables accurate and direct execution of raw acceleration policies acting on state feedback without requiring an additional feedback controller. In essence, our online learner becomes the adaptive error feedback term, where the error

*We use "observable" in the sense of "observability" in control, to mean variables that can be estimated from sensor measurements. There may be multiple ways in which these variables can be estimated. For instance, accelerations can be observed directly through accelerometer readings, or through a statistical estimator on state readings (accelerations via finite-differencing). Note that depending on the estimator used, there may be differences in statistical variance of the estimate. Additionally, through a telescoping argument of summed finite-differences, it can be shown that common finite-differencing estimators are often usable in practice despite the noise in the raw estimates.

is calculated in the space of accelerations. Thus, with the change in loss function, we both enable control techniques to operate on the full kinematic description given by the acceleration policy and we open these control techniques to a new domain of inquiry by expressing it in the language of machine learning. This enables new communities of researchers to start tackling this problem from different perspectives using unique tools.

We start out by reviewing the various backgrounds of inverse dynamics control, adaptive control, and online gradient descent in the Background section. Then, in the Direct Online Optimization of Modeling Errors in Dynamics section, we derive our proposed approach *Direct Online Optimization of Modeling Errors in Dynamics*. In the Experiments section, we extensively evaluate our work both in simulation and on a real robotic system. Finally, we conclude in the Conclusion and Future Work section.

Background

Our work has connections to a variety of research areas such as inverse dynamics control, adaptive control, and the use of modern machine learning techniques such as online gradient-based optimization commonly used in large-scale deep learning. In the following, we provide a (nonexhaustive) review and introduction into these topics to cater to readers from various backgrounds.

Inverse dynamics

The dynamics of any classical dynamical system[3] can be derived from the Principle of Least Action and expressed as

$$\boldsymbol{\tau} = \boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{h}(\boldsymbol{q}, \dot{\boldsymbol{q}}), \tag{1}$$

where $\boldsymbol{q}, \dot{\boldsymbol{q}}$ and $\ddot{\boldsymbol{q}}$ denote the current state and accelerations of the system, respectively. Furthermore, $\boldsymbol{M}(\boldsymbol{q})$ represents the generalized inertia matrix and $\boldsymbol{h}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ collects the modeled forces, including gravitational, Coriolis, and centrifugal forces and viscous and Coulomb friction.

Model-based control[12,13] constructs a model of these dynamics to predict the torques $\boldsymbol{\tau}$ required to realize desired accelerations $\ddot{\boldsymbol{q}}$ in the current state $\boldsymbol{q}, \dot{\boldsymbol{q}}$ by estimating the true dynamical functions $\boldsymbol{M}(\boldsymbol{q})$ and $\boldsymbol{h}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ using data. We generally denote these estimated parameters as $\widehat{\boldsymbol{M}}$ and $\widehat{\boldsymbol{h}}$. In particular, for manipulators, rigid body assumptions commonly substantially simplify the mathematics of the estimation problem. These rigid body dynamics (RBD) models are linear in the unknown model parameters and permit the use of standard linear regression techniques to estimate $\widehat{\boldsymbol{M}}$ and $\widehat{\boldsymbol{h}}$:

$$\boldsymbol{\tau} = \widehat{\boldsymbol{M}}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \widehat{\boldsymbol{h}}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \mathbf{Y}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}})\boldsymbol{a}, \tag{2}$$

where the inverse dynamics torques are a linear combination of the inertial parameters $\boldsymbol{a}$ and a nonlinear function $\mathbf{Y}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}})$ of positions, velocities, and accelerations, as has been shown in An et al.[14]

Equation (1) is written in *inverse dynamics* form. Given an acceleration $\ddot{\boldsymbol{q}}$, it tells us what torque $\boldsymbol{\tau}$ would produce that acceleration. Inverting the expression gives the generic equation for forward dynamics:

$$\ddot{\boldsymbol{q}} = \boldsymbol{M}^{-1}(\boldsymbol{\tau} - \boldsymbol{h}). \tag{3}$$

This equation expresses the kinematic effect of applied torques $\boldsymbol{\tau}$ in terms of the accelerations $\ddot{\boldsymbol{q}}$ they generate.

Online learning of inverse dynamics

While inverse dynamics control with an estimated RBD model is successfully deployed on modern manipulation platforms,[5] the inaccuracies of the estimated dynamics model remain an open issue. The better we can model and predict the dynamics, the less we rely on error feedback control to account for modeling errors. Thus, the learning of inverse dynamics (error) models is an active research area.

The problem of inverse dynamics learning has many facets and is tackled from many different fronts. A focus of recent research progress has been scaling up modern function approximators so that real-time (online) model learning becomes feasible.[6–10,15] These methods attempt to learn a *global* inverse dynamics model, which can be updated online and used for real-time prediction. These approaches differ in whether the hyperparameters of the function approximator are adapted online[7,9–11] or offline on a fixed training data set.[6,8] Furthermore, they can be divided into methods that update the hyperparameters to fit the currently observed data, meaning the parameters adapt to the current data distribution,[10,11,15] or whether they attempt to learn a globally valid model and thus design algorithms that minimize the forgetting of parameters learned in the past.[7,9] The latter type of approaches retains a memory and theoretically improves with repeated execution of similar tasks, and as such is often categorized as *learning control* methods. Computational efficiency and robustness have been the focus of this research path.

Our proposed approach falls into the category of *adaptive control*.[16] Adaptive control approaches update

either controller or system model parameters online. As opposed to *learning control* approaches, there is no notion of improving over multiple task trials. Within this field, a popular approach has been to utilize the RBD model to derive updates for adaptive control laws.[17,18] For this, the linear relationship of the RBD parameters and the torques Equation (2) is used also for modeling the RBD model error. Then, Lyapunov-based update rules are derived to continuously (in real time) update the error model. In contrast, in our work, we derive gradient-based online learning algorithms for tuning dynamics model function approximators to directly minimize the discrepancy between desired and actual accelerations, and demonstrate their real-world efficacy for the case of adapting the torque offset needed to achieve a desired acceleration.

Adaptive control on direct versus indirect loss

In the context of adaptive control, we make the distinction between *direct* and *indirect* loss minimization approaches. To explain the difference, we first take a more detailed look at the error made when using an approximate RBD model.

For any given desired acceleration $\ddot{\boldsymbol{q}}_d$, we calculate torques $\boldsymbol{\tau} = \widehat{\boldsymbol{M}}\ddot{\boldsymbol{q}}_d + \widehat{\boldsymbol{h}}$ using our approximate inverse dynamics model, but when we apply those torques they are pushed through the *true* system that may differ substantially from the estimated model:

$$\ddot{\boldsymbol{q}}_a = \boldsymbol{M}^{-1}\left((\widehat{\boldsymbol{M}}\ddot{\boldsymbol{q}}_d + \widehat{\boldsymbol{h}}) - \boldsymbol{h}\right), \tag{4}$$

where here $\ddot{\boldsymbol{q}}_a$ denotes the actual observed accelerations of the true system. We emphasize that this true system model is generic and holds for any Lagrangian mechanical system, without requiring rigid body assumptions. The true dynamics is typically unknown, so this expression, thus far, is of only theoretical interest. We will see below that expressing the structure in this way enables the derivation of a practical algorithm that we can use in practice.

Rigid body assumptions are often restrictive, introducing a substantial offset between the true model and the estimated model. We propose, therefore, to learn an offset function $\boldsymbol{f}_{\text{offset}}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}_d, \boldsymbol{w})$ that models the error made by this approximate RBD model.

$$\ddot{\boldsymbol{q}}_a = \boldsymbol{M}^{-1}\left(\left(\widehat{\boldsymbol{M}}\ddot{\boldsymbol{q}}_d + \widehat{\boldsymbol{h}} + \boldsymbol{f}_{\text{offset}}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}_d, \boldsymbol{w})\right) - \boldsymbol{h}\right), \tag{5}$$

In an extreme case, when the approximate dynamics model is the zero function, this $\boldsymbol{f}_{\text{offset}}$ function represents the full inverse dynamics model, which must be trained from data. Often, though, we can take it to be a model representing only the difference between the true inverse dynamics and the modeled inverse dynamics (e.g., calculated under standard rigid body assumptions).

Depending on what type of loss function is used to adapt parameters $\boldsymbol{w}$, we distinguish between adaptive control as *direct* and *indirect* loss minimization approaches. More concretely, consider an objective of the form as follows:

$$l_{\text{indirect}}(\boldsymbol{w}) = \|\boldsymbol{\tau} - \boldsymbol{f}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}, w)\|^2, \tag{6}$$

where f is any class of function approximators parameterized by $\boldsymbol{w}$ predicting the inverse dynamics. In this case, $\boldsymbol{\tau}$ is the actual applied torques, and $\ddot{\boldsymbol{q}}$ is the corresponding actual accelerations that were observed. Adapting parameters based on this loss function attempts to make the model's predicted torque on the *actual* observed acceleration more accurate. However, in reality, we wanted to achieve the desired accelerations. This discrepancy is especially problematic when stictions are involved: all actual observed accelerations are zero when we are not applying enough torque, and thus, the error model never receives data to accurately predict an offset for desired torques.[†] In these cases, without a state error feedback control term, pushing through stictions is problematic.

In this work, we instead develop a new adaptive control methodology that *directly minimizes* the acceleration error:

$$l_{\text{direct}}(\boldsymbol{w}) = \|\ddot{\boldsymbol{q}}_d - \ddot{\boldsymbol{q}}_a(\boldsymbol{w})\|_{\boldsymbol{M}}^2. \tag{7}$$

This objective measures the inertia weighted difference between observed and desired accelerations. The inertia weighting is a common metric on acceleration differences used first in Gauss's Principle of Least Constraint[3] (which can be used to develop all of analytical dynamics[19]) and, more recently, in many modern control techniques building on those principles as given in Ref.[20] This metric is intuitively nice for dynamics formulations since the first derivative of the error itself (before accounting for chain rule terms) is in units of

[†]Depending on the rigidity of the hypothesis class representing the offset function, the system may go through an online iterative improvement process that tangentially coaxes the predicted torque at the desired acceleration to increase over time until it is large enough to push through stiction. Basically, predicting a torque τ at $\ddot{q}_a = \boldsymbol{0}$ may induce a slightly larger torque prediction $\tau + \Delta\tau$ at the desired acceleration $\ddot{q}_d$. If that is the case, the next training step will update the model to predict the slightly larger $\tau + \Delta\tau$ at $\ddot{q}_a$, which will in turn induce an even larger prediction $\tau + 2\Delta\tau$ at $\ddot{q}_d$ and so on. The process may ultimately converge toward large enough torques to break through the stiction, but it is hard to analyze and the property need not hold in general.

force, since $\boldsymbol{M\ddot{q}}$ has units of force, and as we will see below, it is especially convenient for our problem. In the Direct Online Optimization of Modeling Errors in Dynamics section, we derive our approach as a gradient-based online learning method on this direct objective, enabling us to leverage a broad collection of practical and theoretically sound tools developed by the machine learning community.

Online and stochastic gradient descent

Machine learning often frames the learning problem as one where, given data, the task is to estimate a model that generalizes that data well (where these terms are made rigorous in various theoretical settings). However, it is often useful to analyze learning algorithms instead within a setting where data are presented only incrementally, and in the extreme, only one data point at a time. This is the subject of *online learning* (see Cesa-Bianchi and Lugosi[21]).

Online learning, over the past decade, has become a general theoretical framework, in which both online learning processes and batch learning processes can be analyzed building from a framework called *regret analysis*. Regret bounds were first studied in the context of online gradient descent in Zinkevich,[22] where regret is defined in terms of how well the algorithm does on the stream of objectives relative do the best it could have done if it had seen all of the objective functions in advance. Characteristic of this approach is the lack of assumptions made on the sequence of objective functions seen. Rather than assuming the data points are independent and identically distributed (iid) as is frequently the case in statistics and machine learning,[23,24] these approaches allow the data stream to be anything. The theoretical performance of an algorithm is rated purely relative to the best it could do. If the sequence is inherently bad, that is ok—the algorithm does as well as possible given the difficult problem. If the sequence is good, then the regret bounds show that the algorithms perform well.

In particular, regret bounds in the online setting can be specialized to give generalization guarantees if additional assumptions are made on the data generation process. For instance, if we additionally assume that the data are iid, then it is possible to produce novel and out-of-sample generalization bounds that are competitive with (or superior to) the best known guarantees.[25,26]

This framework is closely related to another widely studied class of algorithms known as stochastic gradient descent,[27] which is perhaps the most commonly used underlying optimization technique in modern large-scale machine learning.[28] Characteristic of stochastic gradient methods is the assumption of noise in the gradient estimates due to both noise in the underlying data and seeing only part of the problem (a statistical "mini-batch" sample) at any given time.

These techniques are extremely important to our settings, since in online control we see only streams of data as they are generated by the robot, and, as we will see below, our gradient estimates are noisy. Deriving these algorithms as gradient-based online learning enables us to inherit all of the analytical techniques and stability properties of these optimizers, as well as a collection of strongly experimentally verified algorithmic variants designed to address the real-world problems that arise from large-scale machine learning in practice.[29,30]

Direct Online Optimization of Modeling Errors in Dynamics

This section derives the most basic variant of our Direct Online Optimization of Modeling Errors in Dynamics (DOOMED) algorithm for tuning an offset to the dynamics model to minimize acceleration errors. We first derive the objective function and then show how its gradient can be estimated from data.

Equation (5) expresses the true observed acceleration achieved on the physical system as a function of the offset function's parameter setting $\boldsymbol{w}$. In full, this expression takes the following form:

$$\ddot{\boldsymbol{q}}_a(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}}_d,\boldsymbol{w}) = \boldsymbol{M}(\boldsymbol{q})^{-1}\left[\left(\widehat{\boldsymbol{M}}(\boldsymbol{q})\ddot{\boldsymbol{q}}_d + \widehat{\boldsymbol{h}}(q,\dot{\boldsymbol{q}}) + \boldsymbol{f}_{\text{offset}}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}}_d,\boldsymbol{w})\right) - \boldsymbol{h}(\boldsymbol{q},\dot{\boldsymbol{q}})\right], \quad (8)$$

but we often use the shorthand $\ddot{\boldsymbol{q}}_a(\boldsymbol{w}) = \boldsymbol{M}^{-1}\ [(\boldsymbol{f}_{\text{id}} + \boldsymbol{f}_{\text{offset}}(\boldsymbol{w})) - \boldsymbol{h}]$ for brevity to suppress the dependence on $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$, and $\ddot{\boldsymbol{q}}_d$ and emphasize the dependencies on $\boldsymbol{w}$. $\boldsymbol{f}_{\text{id}}$ here denotes the modeled approximate inverse dynamics function. The observed accelerations $\ddot{\boldsymbol{q}}_a$ are then a function of $\boldsymbol{w}$. Now that we have an expression for $\ddot{\boldsymbol{q}}_a(\boldsymbol{w})$ for the true accelerations as a function of the offset function's parameters $\boldsymbol{w}$, we can write out an explicit loss function measuring the

error between the desired accelerations and actual accelerations:

$$l(\boldsymbol{w}) = \frac{1}{2} \| \ddot{\boldsymbol{q}}_d - \ddot{\boldsymbol{q}}_a(\boldsymbol{w}) \|_{\boldsymbol{M}}^2 . \tag{9}$$

This error is a common metric most famously used in Gauss's Principle of Least Constraint.

Note that $\boldsymbol{M}$ in this expression is the *true* mass matrix, which we do not know, as is the implicit $\boldsymbol{h}$ in $\ddot{\boldsymbol{q}}_a(\boldsymbol{w})$, so we cannot evaluate the objective directly. However, the Jacobian of $\ddot{\boldsymbol{q}}_a(\boldsymbol{w})$ is

$$\frac{\partial}{\partial \boldsymbol{w}} \ddot{\boldsymbol{q}}_a(\boldsymbol{w}) = \boldsymbol{M}^{-1} \boldsymbol{J}_f, \tag{10}$$

where $\boldsymbol{J}_f = \frac{\partial \boldsymbol{f}_{\text{offset}}}{\partial \boldsymbol{w}}$ is the Jacobian of the offset function approximator. So, if we evaluate the gradient of the expression, we see that the unknown elements all vanish from the expression:

$$\nabla_w l(\boldsymbol{w}) = \nabla_{\boldsymbol{w}} \frac{1}{2} \| \ddot{\boldsymbol{q}}_d - \ddot{\boldsymbol{q}}_a(\boldsymbol{w}) \|_{\boldsymbol{M}}^2 \tag{11}$$

$$= - \left[\frac{\partial \ddot{\boldsymbol{q}}_a(\boldsymbol{w})}{\partial \boldsymbol{w}} \right]^T \boldsymbol{M} (\ddot{\boldsymbol{q}}_d - \ddot{\boldsymbol{q}}_a(\boldsymbol{w})) \tag{12}$$

$$= - \boldsymbol{J}_f^T \boldsymbol{M}^{-1} \boldsymbol{M} (\ddot{\boldsymbol{q}}_d - \ddot{\boldsymbol{q}}_a(\boldsymbol{w})) \tag{13}$$

$$= - \boldsymbol{J}_f^T (\ddot{\boldsymbol{q}}_d - \ddot{\boldsymbol{q}}_a(\boldsymbol{w})), \tag{14}$$

leaving us with a combination of quantities that we can evaluate, measure, or otherwise observe in practice.[‡] Interestingly, this expression is intuitive. The gradient is simply the acceleration error pushed through the Jacobian of the offset function. We know the desired acceleration $\ddot{\boldsymbol{q}}_d$, we can easily obtain the true acceleration $\ddot{\boldsymbol{q}}_a$, and we assume we can evaluate the Jacobian of the offset function approximator $\boldsymbol{f}_{\text{offset}}$. So despite being unable to evaluate the objective or fully evaluate the gradient, we can still obtain the gradient from the running system (online) in practice. For the experiments in this article, we use an extremely simple offset function of the form $\boldsymbol{f}_{\text{offset}}(\boldsymbol{w}) = \boldsymbol{w}$ representing the excess instantaneous torque needed to accelerate as desired.[§] For this simple offset expression, the Jacobian is just the identity matrix $\frac{\partial}{\partial \boldsymbol{w}} \boldsymbol{f}_{\text{offset}} = \boldsymbol{I}$, so the gradient is simply the acceleration error.

Note also that if our parameters are hypothesized forces in any task space, or multiple task spaces, the resulting gradient expression is again intuitive. Let $\boldsymbol{f}_{\text{offset}} = \sum_{i=1}^{k} \boldsymbol{J}_i^T \lambda_i$, where $\boldsymbol{J}_i$ is the Jacobian of the task map (e.g., the Jacobian of the end-effector when the task space is the end-effector space), and λ_i is a hypothesized force applied in the task space (e.g., at the end-effector). Then, $\boldsymbol{J} = [\boldsymbol{J}_1^T, \ldots, \boldsymbol{J}_k^T]$, and the gradient of the loss becomes

$$\nabla_w l = - \begin{bmatrix} J_1 \\ \vdots \\ J_k \end{bmatrix} (\ddot{\boldsymbol{q}}_d - \ddot{\boldsymbol{q}}_a) = - \begin{bmatrix} \ddot{\boldsymbol{x}}_1^d - \ddot{\boldsymbol{x}}_1^a \\ \vdots \\ \ddot{\boldsymbol{x}}_k^d - \ddot{\boldsymbol{x}}_k^a \end{bmatrix}, \tag{15}$$

where $\ddot{\boldsymbol{x}}_i^d = J_i \ddot{\boldsymbol{q}}_d$ is the desired acceleration through the ith task space, and $\ddot{\boldsymbol{x}}_i^a = \boldsymbol{J}_i \ddot{\boldsymbol{q}}_a$ is the actual acceleration through the ith task space. In other words, the same intuitive rule for estimating the gradient holds within any task space: the gradient is simply the acceleration error as observed in the task space.

Using this loss function as the risk term in an online regularized risk objective of the form

$$\mathcal{L}(\boldsymbol{w}) = \frac{1}{2} \| \ddot{\boldsymbol{q}}_d - \ddot{\boldsymbol{q}}_a(\boldsymbol{w}) \|_{\boldsymbol{M}}^2 + \frac{\lambda}{2} \| \boldsymbol{w} \|^2, \tag{16}$$

where λ determines the importance of the regularization term, we can write out a simple gradient descent online learning algorithm as

$$\boldsymbol{w}^{t+1} = (1 - \eta^t \lambda) \boldsymbol{w}^t + \eta^t \boldsymbol{J}_f^T (\ddot{\boldsymbol{q}}_d^t - \ddot{\boldsymbol{q}}_a^t). \tag{17}$$

Where $\boldsymbol{J}_f = \boldsymbol{I}$ (or is the Jacobian of a mapping to some task space), this expression is essentially an integral term on the acceleration error with a forgetting factor. However, vanilla gradient descent is the simplest online gradient-based algorithm we could use. By deriving the method as online learning, we now understand theoretically how to apply an entire arsenal of new, more powerful, and adaptive online gradient-based algorithms to this same problem to improve performance.

Our primary concern in this work is quick response time to perturbations such as unmodeled objects in the end-effector, unmodeled friction, and physical pushes. Hence, we focus our experiments on learning simply the instantaneous constant offset between the predicted inverse dynamics torques and the actual torques needed to produce a given desired acceleration.

[‡]In practice, we observe accelerations in our experiments using finite-differencing on state measurements. See Real-World Experiments on a KUKA Lightweight Arm section for real-world experiments using noisy acceleration estimates, and the Handling Noisy Acceleration Estimates section for discussions of how estimator variance affects these algorithms.

[§]Note that this function is constant, but in practice, the online learning algorithm is able to *track* the needed offset as it changes across a single movement as a function of the state-dependent inaccuracies.

This model is very simple and quick to learn, so it can adjust in real time as the system shifts between regions of the state space with differing inverse dynamics errors. The aforementioned task space force model has a similar property. We do not empirically analyze it in this work, but since it remains linear while leveraging prior knowledge and mathematical modeling to define well-informed and intuitive nonlinear features for the problem, it may actually converge faster in practice. However, highly flexible large-scale general purpose function approximators, such as deep networks, often have higher data complexities and would, alone, react more slowly to changes in the dynamics. We are actively exploring how to utilize online learning processes, such as the methodology described here, alongside slower offline processes operating at a different time scale on more sophisticated models to best leverage the generalization capacity of large-scale function approximators while maintaining the real-time reactiveness characteristic of DOOMED.[31]

Next, we review two tricks pulled from the combined online learning literature (or more general stochastic gradient descent machine learning literature) and the adaptive control literature to remove the potential for parameter oscillations and track changes in modeling errors while simultaneously enabling high accuracy for precise meticulous movements.

Parameter oscillations in online learning and their physical manifestation

Parameter oscillations in neural networks are a problem resulting from ill-conditioning of the objective. The objective, as seen from the parameter space (under Euclidean geometry, which is a common easy choice), is quite elongated, meaning that it is extremely stretched with a highly diverse Hessian eigenspectrum. Gradient descent alone in those settings undergoes severe oscillations making its progress slow. More importantly, in our case, oscillations resulting from the ill-conditioning of the problem manifest physically as oscillations in the controller when the step size is too large. Fortunately, the learning community has a number of tricks to prevent these oscillations and promote fast convergence.

The most commonly used method for preventing oscillations is the use of momentum. Denoting $\boldsymbol{g}^t$ as the gradient at time t, the momentum update is

$$\boldsymbol{u}^{t+1} = \gamma \boldsymbol{u}^t + (1-\gamma)(-\boldsymbol{g}^t) \tag{18}$$

$$\boldsymbol{w}^{t+1} = \boldsymbol{w}^t + \eta \boldsymbol{u}^{t+1}. \tag{19}$$

Effectively, we treat the parameter $\boldsymbol{w}$ as the location of a particle with mass and treat the objective as a potential field generating force on the particle. The amount of mass affects its perturbation response to the force field, so larger mass results in smoother motion.

It can be shown that this update can be written equivalently as an exponential smoother:

$$\boldsymbol{u}^{t+1} = \boldsymbol{u}^t - \eta \boldsymbol{g}^t, \tag{20}$$

$$\boldsymbol{w}^{t+1} = \gamma \boldsymbol{w}^t + (1-\gamma)\boldsymbol{u}^{t+1}. \tag{21}$$

Note that $\boldsymbol{g}^t$ is still being evaluated at $\boldsymbol{w}^t$, so it is not exactly equivalent to running simply a smoother on gradient descent (gradients in our case come from evaluations at smoothed points), but it is similar.

This latter interpretation is nice because it shows that we are taking the gradients and (1) literally smoothing them over time, and (2) effectively operating on a slower time scale. That second point is important: this technique works because the time scale of the changing system across motions generated by the acceleration policies is fundamentally slower than that of the controller. This enables the controller (online learning) to use hundreds or even thousands of examples to adjust to new changes as it moves between different areas of the configuration space.

Another common trick found in the machine learning literature (especially recently due to its utility in deep learning training[29,30]) is to scale the space by the observed variance of the error signal. We take a related approach motivated by these ideas. When the error signal has high variance in a given dimension, the length scale of variation is smaller (small perturbations result in large changes). In that case, the step size should decrease. Similarly, when the observed variance is small, we can increase the step size to some maximal value. In our case, we care primarily about variance in the actual accelerations $\ddot{\boldsymbol{q}}_a$ (which measures the baseline noise, too, in the estimates) since we can assume the desired acceleration $\ddot{\boldsymbol{q}}_d$ signal is changing only slowly relative to the 1 ms control loop. Denoting this $\ddot{\boldsymbol{q}}_a$ variance estimate as $\boldsymbol{v}^t$, we scale the update as

$$\boldsymbol{u}^{t+1} = \boldsymbol{u}^t - (\boldsymbol{I} + \alpha\ \mathbf{diag}(\boldsymbol{v}^t))^{-1}\boldsymbol{g}^t. \tag{22}$$

Note that this is equivalent to using an estimated metric or Hessian approximation of the form $\boldsymbol{A} = \boldsymbol{I} + \alpha\ \mathbf{diag}(\boldsymbol{v}^t)$. See Ratliff et al.[32] for more details.

This combination of exponential smoothing (momentum) and a space metric built on an estimate of

variance results in a smoothly changing $\boldsymbol{w}$ that is still able to track changes in the dynamics model errors.

Adaptive tuning of forgetting factors

Indirect approaches to adaptive control (essentially online regressions of linear dynamics models) often tune their forgetting factors based on the magnitude of the error they are seeing.[33] Larger errors mean that the previous model is bad and we should forget it fast to best leverage the latest data. Smaller errors mean that we are doing pretty well, and we should use as much data as possible to fully converge to zero tracking error. Adaptively tuning of forgetting factors, which manifests as adaptive tuning of the regularizers in our case, enables fast response to new modeling errors while simultaneously promoting accurate convergence to meticulous manipulation targets.

As described in detail in Ratliff et al.,[32] we use one forgetting factor per dimension to form a vector of regularization constants λ. In our experiments, we utilize algorithmic variants that adapt each regularization constant based on the acceleration error for that dimension. In particular, for controlling the end-effector to a fixed Cartesian point, the forgetting factor converges to 1 (no forgetting; zero regularization) and within a couple of seconds (including approach slowdown) we achieve accuracies of around 10^{-5} m.

Handling noisy acceleration estimates

The Parameter Oscillations in Online Learning and Their Physical Manifestation section described the tools we use from the machine learning (especially stochastic gradient descent) literature to reduce oscillations. However, in addition, since handling noisy data is a fundamental problem to machine learning in general, these same tools enable us to handle noisy acceleration estimates.

First, the basic algorithms themselves are robust to noise. These gradient-based algorithms are most commonly applied in stochastic contexts, where it is assumed that gradient estimates are noisy. In addition, the article[32] discusses how sequential finite-differenced accelerations actually telescope to an extent allowing the noise to inherently cancel over time.

However, second, momentum acts as a damper to the forces generated by the objective. Its interpretation as an exponential smoother shows that white noise in the estimates cancels over time through the momentum as well, averaging to a more clear acceleration signal.

And finally, we get 1000 training points per second. Physically the robot does not move very far in half a second, and we can assume that the errors in the dynamics model will be changing at a time scale of tenth of a second, 0.1, 0.5, or even 1 second. That is anywhere between 100 and 1000 training examples available to track how errors in the dynamics model change as the robot executes its policy, which is plenty of data to average out noise and run a sufficient number of gradient descent iterations.

A note on step size gains and an analogy to PD gains

The larger the step size, the quicker the adaptive control strategy adjusts to errors between desired and actual accelerations. That means it will fight physical perturbations of the system stronger with a larger step size. To accurately track desired accelerations, we either need large step sizes for fast adaptation or a good underlying dynamics model. If we have a really good dynamics model, we can get away with smaller step sizes. That means the better the dynamics model, the easier physical interaction with the robot becomes. Bad models require larger step sizes, and this manifests as a feeling of "tightness" in the robot's joints. We can always push the robot around, and it will always follow its underlying acceleration policy from wherever it ends up, but the more we rely on the adaptive control techniques, the tighter the robot becomes and the more force we need to apply to physically push it around.

This behavior parallels the trade-offs we see in the choice of Proportional-Derivative (PD) gains for trajectory following. Bad (or no) dynamics models require hefty PD gains, which means it can be near impossible to perturb the robot off its path. However, the better the dynamics model, the better the feedforward term is, and the looser we can make the PD gains while still achieving good tracking. We are able to push the robot around more easily (and it is safer). The difference is whether or not we need that trajectory. The proposed direct adaptive control method attempts to follow the desired acceleration policy well, which means when we perturb it, it always continues from where it finds itself rather than attempting in any sense to return to a predefined trajectory or integral curve.

Experiments

We evaluate our proposed approach in three stages. First, we illustrate our approach on a simple 2D simulation experiment, to give the reader some intuition. Then, we use simulation software for two robotic systems to show the benefit of combining model-based inverse dynamics control with our data-driven approach. Finally, we move to

a real robotic platform—a KUKA lightweight arm—to illustrate the effectiveness of our approach on a real system. In all our experiments presented here, we model the offset as a simple constant $\boldsymbol{f}_{\text{offset}}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}_d, \boldsymbol{w}) = \boldsymbol{w}$, and an error offset is learned per degree of freedom.

A simple simulated experiment

This experiment shows a simple 2D example of the behavior of the adaptive control system, as illustrated in Figure 1. This illustration shows a scenario where the dynamics model used by the robot differs drastically from the true dynamics and where unmodeled nonlinear frictions are significant.

The true mass matrix of the system is defined as $\boldsymbol{M}(\boldsymbol{q}) = 5(\boldsymbol{v}(\boldsymbol{q})\boldsymbol{v}(\boldsymbol{q})' + 0.05\boldsymbol{I})$, with $\boldsymbol{v}(\boldsymbol{q}) = (\sin(5q_1); \cos(2q_2))$. The robot uses a diagonal constant approximation of the form $\widehat{\boldsymbol{M}} = 0.5\boldsymbol{I}$, which assumes masses that are an order of magnitude smaller. In addition, the true system experiences sinusoidal frictional forces of the form

$$\mu(\boldsymbol{q}) = \begin{bmatrix} 100\sin(50\ q_1) \\ 5\sin(50\ q_2) \end{bmatrix}, \tag{23}$$

for which the robot has no knowledge of.

The system is using a simple PD controller (outputting accelerations) to move to a desired fixed point to a target velocity of zero.

$$\ddot{\boldsymbol{q}} = K(\boldsymbol{q}_d - \boldsymbol{q}) - D\dot{\boldsymbol{q}}, \tag{24}$$

where $K = 100$ and $D = 10$. The desired target is $\boldsymbol{q}_d = (1; 1)$, and the robot starts from $\boldsymbol{q}_0 = (0; 0)$ with a random initial velocity.

The acceleration controller itself is tuned incorrectly and therefore overshoots. However, that is the desired behavior we want to track (shown in blue). If we simply pipe the desired accelerations through the very approximate inverse dynamics model, the behavior we get is abysmal (shown in red). On the one hand, the dynamics model is a severe approximation, and on the other hand, we have no advanced knowledge of the strange frictional pattern, so the robot drifts upward and then oscillates during the final approach. However, when we turn on adaptation (shown in green), the system is able to compensate for all of that and we get very good tracking behavior.

This implementation did not simulate noise. However, the real-world implementation discussed in the next section shows the efficacy of the above-described learning tricks under noisy acceleration estimates sent back by the physical robot.

Robotic simulation experiments

The next series of experiments demonstrates the effectiveness of our approach through experiments performed in simulators of existing robotic platforms. Using simulation allows us to safely evaluate an inverse dynamics controller without a feedback control term (such as Proportional-Integral-Derivative [PID] terms). This enables us to directly compare the acceleration error of the pure model-based inverse dynamics with the hybrid approach—which adds online learning—without the interference of additional feedback terms.

Experiments on a simulated Baxter platform. We ran a simulation experiment on the Baxter platform to

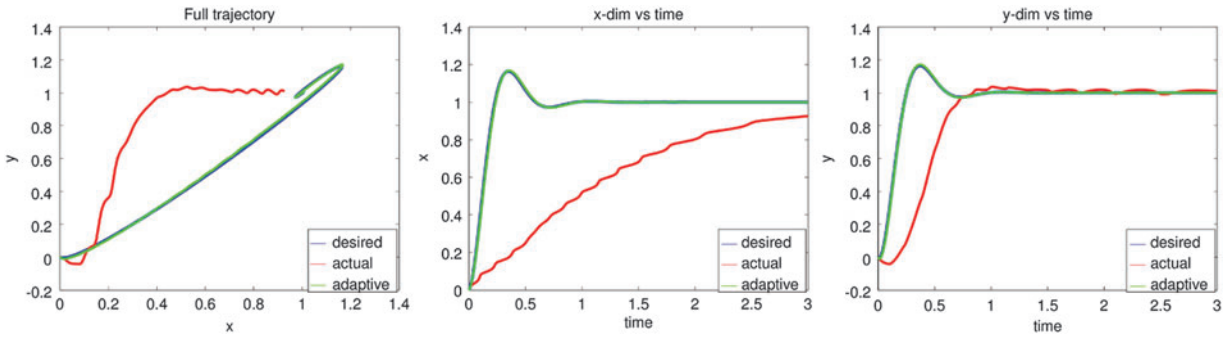

FIG. 1. Two-dimensional simulation of online adaptation to severe model inaccuracies and underlying friction (which varies sinusoidally with each dimension in this case). The time axes (lower axis of the second and third plots) are in units of seconds, and all spacial axes are in units of meters. The controller updated at a rate of 1 kHz.

analyze the performance of the online torque offset learner under severe modeling mismatches. The controller used inverse dynamics, but the simulator added substantial positionally dependent nonlinear friction and torque biases of the form

$$\tau_{\text{friction}} = -7\sin^2(5\boldsymbol{q})(2\sigma(\dot{\boldsymbol{q}}) - 1) \quad \tau_{\text{bias}} = -5\sin(5\boldsymbol{q}),$$

where σ is the typical 0–1 sigmoidal squashing function. In addition, the robot's internal model did not account for joint damping coefficients, while the simulator did. The controller ran with a control cycle of 0.005 seconds, and the simulator ran with an update cycle time of 0.001 seconds. We ran the controller through a 30-second sequence of 10 chained 3-second LQRs and turned on the online learning only halfway through. Figure 2 shows severe acceleration errors resulting from the model mismatch during the first half, but halfway through, the online learning turns on and is able to track the torque offsets quite well, effectively zeroing the acceleration errors. The simulator added noise of the form 0.001uniform$(-1, 1)$ to both the positions and offsets when reporting them to the controller, so the raw acceleration estimates were quite noisy. The acceleration errors in the middle plot of the figure were, therefore, exponentially smoothed with a smoothing factor of 0.975. The online learner used momentum updates, which is effectively a form of smoothing, so the final plot shows the raw unsmoothed learned torque offsets used by the controller.

Experiments on a simulated KUKA lightweight arm. In addition to the Baxter experiments, we also performed a series of evaluations in simulations for the KUKA lightweight arm. For these experiments, we specifically misspecify the mass and friction parameters of the assumed approximate RBD model. We ran a sequence of two chained LQRs—mimicing a pick-and-place task. This sequence was executed with the following parameters: $\alpha = 0.5$ (variance gain), $\gamma = 0.9$ (exponential smoothing parameter), and $\eta = 0.0, 0.01, \ldots, 0.1$ (learning rate of the gradient descent step). A learning rate of $\eta = 0$ means that no online learning was performed and only the approximate RBD model was considered to achieve desired accelerations.

The LQR sequence is executed for each learning rate setting and three metrics are evaluated:

Error metric 1: $\sum_t |\ddot{\boldsymbol{q}}_d^t - \ddot{\boldsymbol{q}}_a^t|/T$, the error between the desired and actual accelerations.
Error metric 2: $\sum_t (\ddot{\boldsymbol{q}}_d^t - \ddot{\boldsymbol{q}}_a^t)/T$, the mean of the acceleration error. A value of 0 here would mean on average the actual accelerations are neither under nor over the desired acceleration.
Error metric 3: $\sum_t \boldsymbol{f}_{\text{offset}}^t/T$, the average magnitude of the adaptive torque command.

We present a summary of these statistics in Figure 3. These results show that the error in terms of acceleration tracking is much higher without the online learning ($\eta = 0$). As soon as the online learner is activated, the error on accelerations is much smaller. Thus, the hybrid approach of model-based inverse dynamics and data-driven error modeling clearly outperforms the purely model-based approach for a range of learning rates. As soon as the learning rate passes some threshold, the

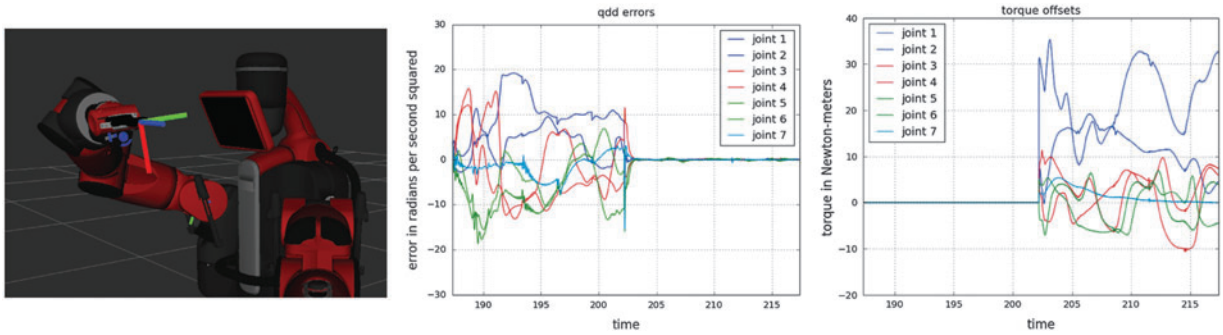

FIG. 2. Left: The Baxter robot. The middle and right plots depict the acceleration errors and learned torque offsets of a 30-second run under severe nonlinear torque biasing, in which online learning was switched on halfway through.

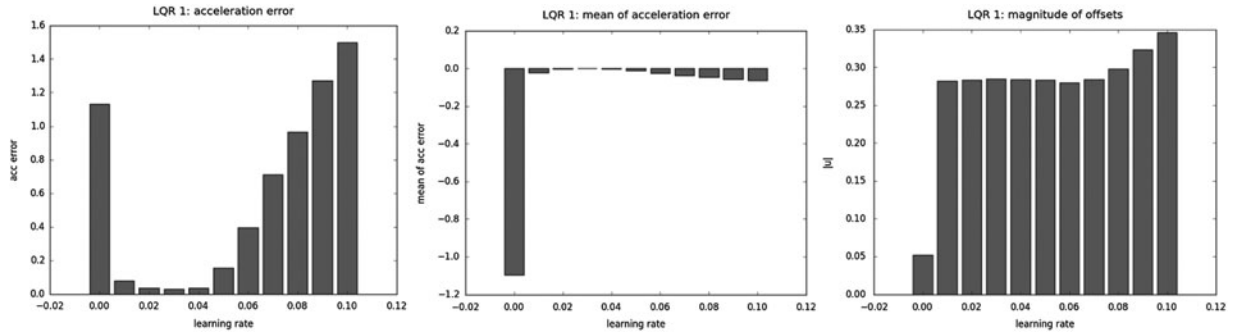

FIG. 3. Errors and torque offset magnitudes as a function of the learning rate (for the first LQR). (Left) Average absolute acceleration error, (middle) mean of the acceleration error, and (right) magnitude of the offset torques computed by the online learner. All statistics are averaged across the seven joints of the KUKA arm. LQR, linear quadratic regulator.

acceleration error can slightly increase. Since the mean of the acceleration error remains small though, this is indicative of oscillations of the online learner. This phenomenon is easily understood as analogy to high-gain PID control, where oscillations can occur when gains are too large.

Real-world experiments on a KUKA lightweight arm

Finally, we have evaluated our hybrid approach on a real system. We have a full implementation of this algorithm working for the Apollo platform (Fig. 4, left) for both simple Cartesian space controllers and a full continuous optimization (MPC-type) motion generation system. On the real system, the low-level controller consists of an inverse dynamics controller, which evaluates the (approximate) RBD model for desired accelerations. This inverse dynamics torque command is combined with the offset estimated by the adaptive controller, see Figure 4. The KUKA lightweight arm has 7 degrees of freedom, and thus, the offset torque is a seven-dimensional vector.

To be robust against noise, the adaptive controller transforms the gradient $\ddot{\boldsymbol{q}}_d - \ddot{\boldsymbol{q}}_a$ as discussed in the Direct Online Optimization of Modeling Errors in Dynamics section. The key parameters that are involved for the offset update are

- the learning rate η [Eq. (17)],
- the variance gain α [Eq. (22)], and
- the smoothing parameter γ [Eq. (21)].

These parameters are shared across all joints. In the following, besides showing the effectiveness of our approach on a real system, we also provide an analysis of

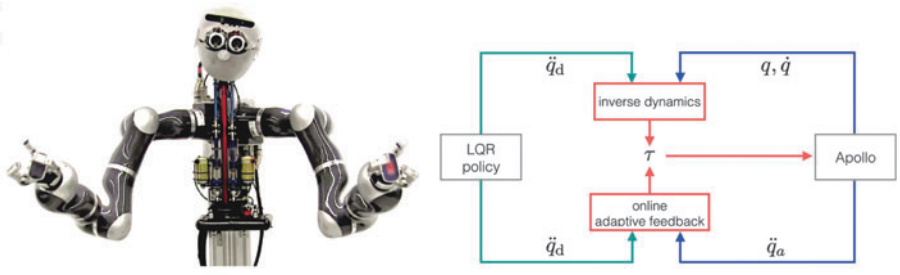

FIG. 4. (Left) Apollo: our experimental platform with two KUKA lightweight arms. (Right) The controller used for our experiments at a rate of 1 kHz.

the sensitivity of our proposed approach on the real Apollo platform.

Parameter sensitivity analysis. For the parameter sensitivity analysis, we attempt to execute a sequence of two preplanned LQRs. This sequence has been executed three times for each parameter combination. The considered values are as follows:

- $\eta = 0.01, 0.02, \ldots, 0.1$
- $\alpha = 0.0, 0.1, \ldots, 0.5$
- $\gamma = 0.9, 0.95$

We have tested all combinations of these parameter settings and report the average acceleration error per joint for both LQRs (averaged over the three trials). If one LQR execution resulted in an unsuccessful execution (within any of the three trials), that parameter setting was classified as unsuccessful. We evaluate the performance and sensitivity of our approach with the help of three quantities defined in the previous Experiments on a Simulated KUKA Lightweight Arm section. The error metrics are visualized as follows:

Error metric 1: We plot this error as a function of the learning rate η and the variance gain γ. This plot is color coded dark green to dark red. Dark green indicates a very low error, dark red indicates higher errors.
Error metric 2: This plot is color coded from red to blue. Red means, on average, we observe actual accelerations over the desired, blue means we observe actual accelerations below the desired. Darker colors indicate a higher bias. Yellow colors represent a bias close to 0.
Error metric 3: Darker color indicates larger magnitudes.

In Figure 5, we plot the average absolute acceleration error and the bias of the acceleration error, for all combinations of η and α while keeping $\gamma = 0.9$ fixed. The first result to notice is that we were able to run the controller for a large portion of parameter combinations. While the error varies across the parameter settings, we can deduce that even without an additional error feedback term, we do not have to perform excessive tuning of the parameters to obtain an empirically well-behaved controller. Note that in all failure cases (indicated by white squares in the plots), it was typically during the execution of the second LQR that the movement was deemed unsafe.

The second result is that there seems to exist a relatively large band across the two parameter settings that seem to achieve low acceleration errors (top two plots of Fig. 5). This promises that tuning the adaptive controller is not too difficult.

When looking at the mean acceleration error bias, we notice that we seem to typically overestimate the required torques for joints 1 and 2 (the shoulder joints) and underestimate the required torque for joints 3–6. This is best understood by keeping in mind that we are fixing the approximate RBD model. Depending on the movement, this approximate model may over or underestimate the required torque to achieve desired accelerations. If the adaptive controller can fix the model, the bias should become smaller (which is represented by lighter colors). For low learning rates, we tend to observe larger bias values. This indicates that we may not be adapting to the error fast enough, and we consistently over/underestimate the required torque offsets, as can be seen in Figure 5, for joints 2–4, for instance.

However, simply increasing the learning rate is not necessarily the right thing to do. Notice, although for joint 1 we observe a small bias of the acceleration error for higher learning rates, the average absolute error increases as we increase the learning rate. This tells us that there may be oscillations, which on average have a small bias. This can mean that we are too aggressive in trying to fix the model incurring larger acceleration errors that we try to account for at the next time step. The variance gain also seems to contribute to this effect as well. With smaller values, this effect seems to be pronounced (for instance, see joint 1, LQR #2 the bottom left corner for both error measures).

Finally, to make sure that our adaptive torque offset estimates are sensible, we plot the offset torque magnitude per joint in Figure 6. We observe that for the shoulder joints, we estimate higher torque offsets than for the elbow and wrist joints. In general, the offset torques range between 0 and 2 Nm, which is a reasonable range.

We have repeated the same set of experiments with a smoothing parameter of $\gamma = 0.95$. The figures for this set of experiments can be found in the appendix of Ratliff et al.[32] In general, the relationship between the variance gain and learning rate is similar to the above-presented results. However, we notice that the effect of higher learning rates is more extreme for some joints. For instance, for joint 5, the acceleration error seems to increase faster (as the learning rate is increased), indicating that oscillations are a function of both the learning rate and smoothing parameter. Intuitively, this makes sense, as the smoothing slows down the response to the errors as well, and thus, a high learning rate with very little smoothing seems to not be an optimal pairing.

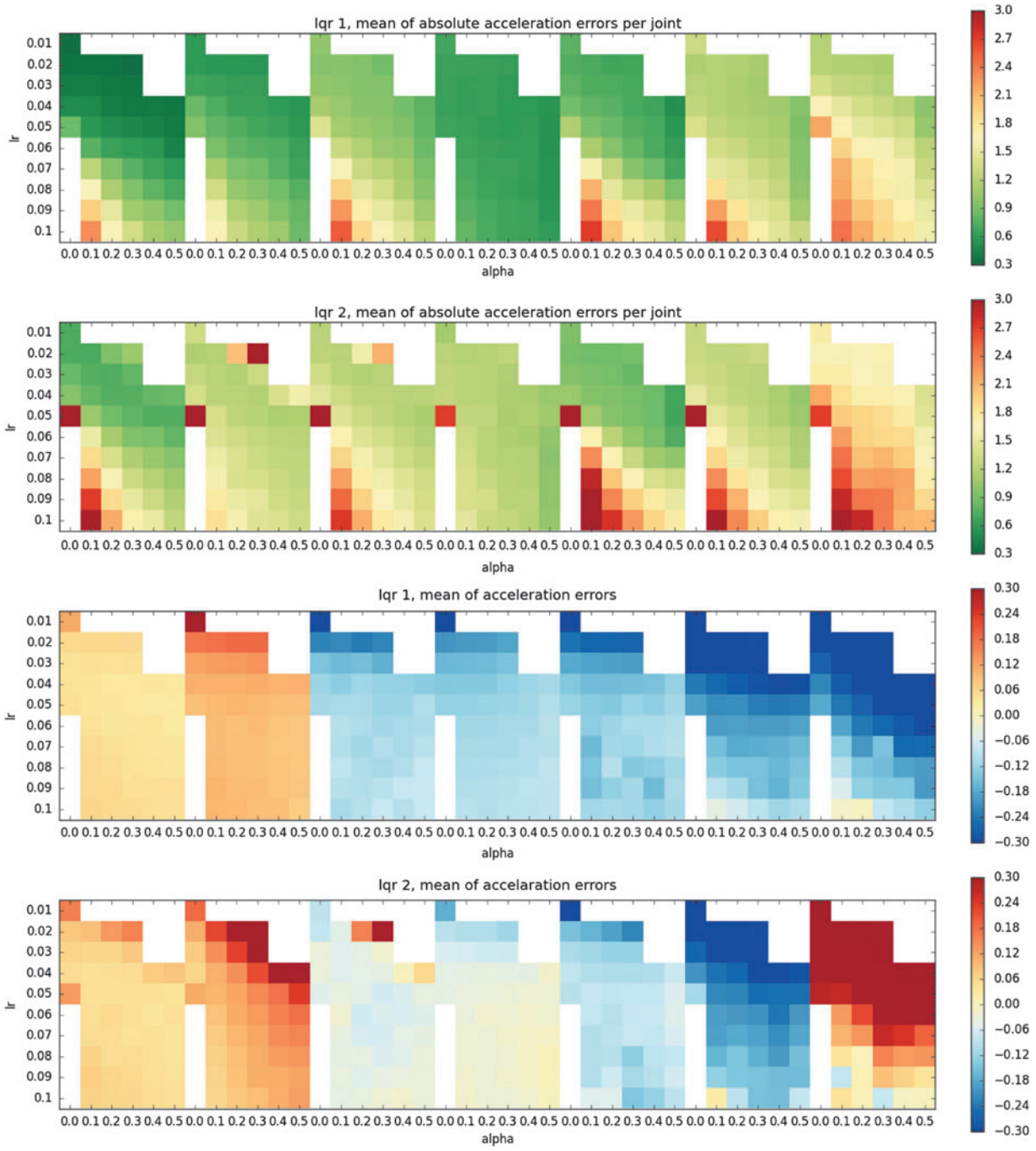

FIG. 5. The average acceleration error and the error bias for both LQRs with $\gamma = 0.9$. Results are displayed per joint, from left to right joint 1 to 7. From top to bottom: the average absolute acceleration for LQR #1 and #2, the mean of the acceleration error bias for both LQRs.

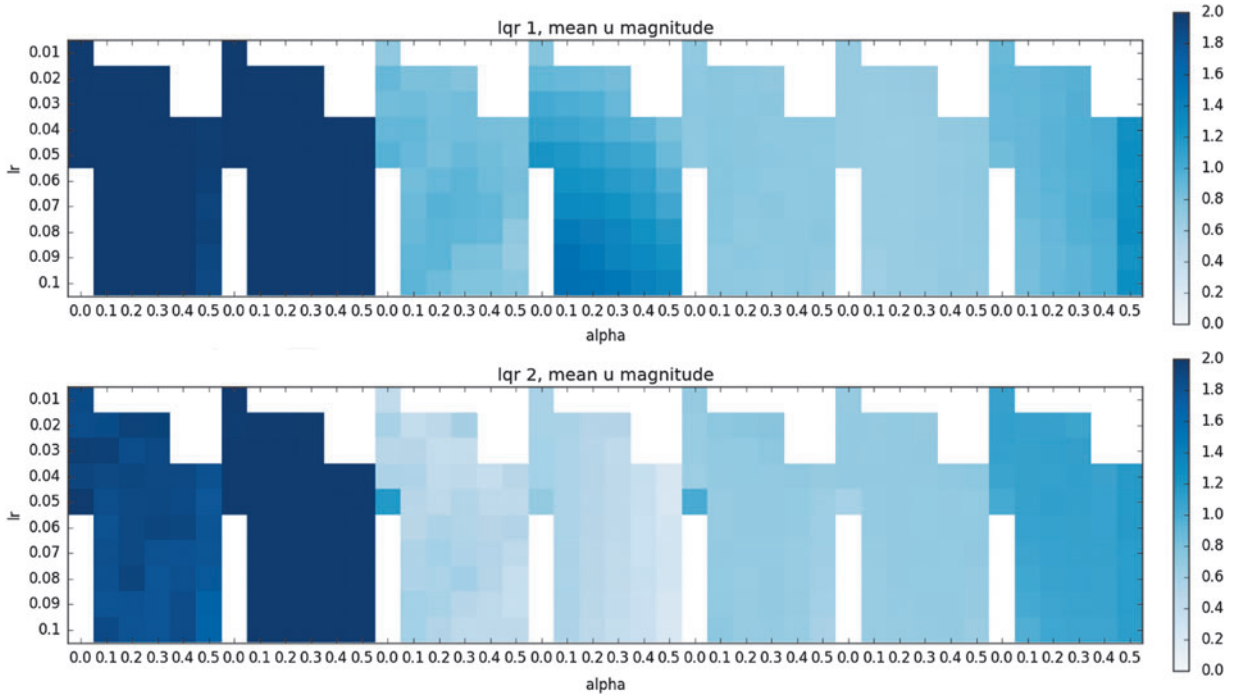

FIG. 6. Torque magnitudes estimated by the adaptive controller $\gamma = 0.9$. (Top) LQR #1 and (bottom) LQR #2. Results are shown per joint, from left to right joint 1 to 7.

All in all we can conclude that the adaptive controller can effectively fix errors in the dynamics model on the fly. Tuning the parameters, while unavoidable, is not too difficult.

Video demonstrations. The following three videos show examples of the dynamics adaptation algorithm running on the Apollo manipulation platform. In the first two videos, Apollo is directly executing an acceleration policy designed to generate Cartesian motion moving his finger to a fixed point in space. In the final video, Apollo is executing optimized grasping behaviors.

1. *Robustness to perturbation:* https://youtu.be/clldz75ToVI

   In this video, Apollo is repeatedly perturbed away from the fixed point and allowed to return under the control of the acceleration policy. The behavior is shaped by the desired accelerations, which Apollo is able to accurately reproduce by running the online learning adaptive control method outlined above to track how the dynamics model error shifts throughout the execution.

2. *Bounce tests:* https://youtu.be/m0i5oHQeqA8

   In this video, Apollo is put through a series of more aggressive bounce tests in an attempt to incite oscillation modes. The robustness measures outlined above successfully combat the maneuvers and Apollo's behavior remains natural throughout the attempt.

3. *Grasp tasks:* https://youtu.be/LQABeK2IO80

   In this video, Apollo is executing grasp task motions optimized on the fly. Each motion is sent down to control as a sequence of affine acceleration policies, which are directly executed using the online learning adaptive control methodology described in this document. No vision is used; the sequence of object locations is planned out in advance. The system, however, uses force control in the grasps (reading from the fingertip's strain gauges) to be robust to variations in size and specific positioning of the objects.

The underlying adaptive control parameterization is the same in all cases—we tune once and that enables the execution of any number of acceleration policies.

Conclusion and Future Work

In this work, we presented a novel approach to online learning of inverse dynamics modeling errors. Our algorithm minimizes a loss function that directly penalizes the error between desired and actual accelerations, enabling the direct execution of raw acceleration policies such as operational space controllers and LQR controllers outputting desired accelerations as a function of the robot's state. Using a direct loss overcomes the off-distribution learning issue present in indirect loss approaches, which prevail in existing state-of-the art inverse dynamics learning. We have shown how we can perform online gradient descent on parameters of general nonlinear function approximators to learn an error model of the dynamics. In our extensive evaluations, even with a simple constant error model, by updating and adapting it online, we show that our approach can correct inverse dynamics errors on the fly for real-world motion generation.

So far we have presented a formal reduction to machine learning along with an experimental exploration of the technique for fixed-based manipulation. However, a rigorous analysis of controller stability is still slated for future work. There is a clear connection between the convergence rate and degree of oscillation along objective troughs during optimization and the stability of the controller that we aim to exploit analytically. Similarly, we plan to explore the use of more modern optimization tools from the machine learning literature, especially the deep learning literature such as Refs.[29,30,34] in the context of the DOOMED framework for the online adaptation of the dynamics model. And, in particular, we plan to leverage their associated theoretical analysis to obtain similar convergence and stability results for our problem.

Future work will also investigate the use of more complex function approximators, especially friction models and body point forces and torques to compensate for tools, sensors, and other types of loads, as well as interactions between this memory-less adaptive control style learning and learning control style iteration model improvement leveraging these updates.

Acknowledgments

This research was supported, in part, by the National Science Foundation grants IIS-1205249, IIS-1017134, EECS-0926052, the Office of Naval Research, the Okawa Foundation, and the Max-Planck-Society. This work was performed, in part, while Nathan Ratliff was affiliated with the Max Planck Institute for Intelligent Systems.

Author Disclosure Statement

No competing financial interests exist.

References

1. Ijspeert AJ, Nakanishi J, Hoffmann H, et al. Dynamical movement primitives: Learning attractor models for motor behaviors. Neural Comput. 2013;25:328–373.
2. Stengel R. Optimal control and estimation. New York: Dover. 1994.
3. Taylor JR. Classical mechanics. University Science Books. 2005.
4. Pastor P, Righetti L, Kalakrishnan M, Schaal S. Online movement adaptation based on previous sensor experiences. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2011. pp. 365–371.
5. Righetti L, Kalakrishnan M, Pastor P, et al. An autonomous manipulation system based on force control and optimization. Auton Robots. 2013;36:11–30.
6. Gijsberts A, Metta G. Real-time model learning using Incremental Sparse Spectrum Gaussian Process Regression. Neural Netw. 2013;41.
7. Meier F, Hennig P, Schaal S. Incremental local Gaussian regression. In: Advances in Neural Information Processing Systems, 27, 2014.
8. Nguyen-Tuong D, Seeger M, Peters J. Local Gaussian process regression for real time online model learning and control. Adv Neural Inf Process Syst. 2008;22.
9. Vijayakumar S, Schaal S. Locally weighted projection regression: An o(n) algorithm for incremental real time learning in high dimensional space. In: Proceedings of the Seventeenth International Conference on Machine Learning (ICML), June 29–July 2, 2000, Stanford University, Stanford, CA. 2000, pp. 1079–1086.
10. Camoriano R, Traversaro S, Rosasco L, et al. Incremental semiparametric inverse dynamics learning. In: 2016 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2016. pp. 544–550.
11. Meier M, Schaal S. Drifting Gaussian processes with varying neighborhood sizes for online model learning. In: 2016 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2016. pp. 264–269.
12. An CH, Atkeson CG, Hollerbach JM. Model-based control of a robot manipulator. Cambridge, MA: MIT Press. 1988.
13. Craig JJ. Introduction to robotics: Mechanics and control, vol. 3. Upper Saddle River: Pearson Prentice Hall. 2005.
14. An CH, Atkeson CG, Hollerbach JM. Estimation of inertial parameters of rigid body links of manipulators. In: 24th IEEE Conference on Decision and Control, 1985. pp. 990–995.
15. Polydoros AS, Nalpantidis L, Krüger V. Real-time deep learning of robotic manipulator inverse dynamics. In: 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2015. pp. 3442–3448.
16. Astrom KJ, Wittenmark B. Adaptive control, In: Antoulas AC (ed) Mathematical System Theory: The Influence of R. E. Kalman. Berlin, Heidelberg: Springer, 1991, pp. 437–450.
17. Craig JJ, Hsu P, Sastry SS. Adaptive control of mechanical manipulators. Boston: Addison-Wesley Longman Publishing Co., Inc. 1988.
18. Slotine J-JE, Li W. On the adaptive control of robot manipulators. Int J Robot Res. 1987;6:49–59.
19. Udwadia FE, Kalaba RE. Analytical dynamics: A new approach, 1st ed. Cambridge University Press. 2007.
20. Bruyninckx H, Khatib O. Gauss principle and the dynamics of redundant and constrained manipulators. In: International Conference on Robotics and Automation, IEEE, 2000, pp. 2563–2568.
21. Cesa-Bianchi N, Lugosi G. Prediction, learning, and games. New York, NY: Cambridge University Press. 2006.
22. Zinkevich M. Online convex programming and generalized infinitesimal gradient ascent. In: Proceedings of the 20th International Conference on Machine Learning (ICML-03), 2003. pp. 928–936.
23. Bishop CM. Pattern recognition and machine learning. New York: Springer. 2007.
24. Wasserman L. All of statistics: A concise course in statistical inference. New York: Springer-Verlag, 2004.
25. Cesa-Bianchi N, Conconi A, Gentile C. On the generalization ability of on-line learning algorithms. IEEE Trans Inf Theory. 2004;50: 20502057.

26. Ratliff N, Bagnell JA, Zinkevich M. (Online) Subgradient methods for structured prediction. In: 11th International Conference on Artificial Intelligence and Statistics (AIStats), March 2007.
27. Bottou L. Stochastic learning. Berlin, Heidelberg: Springer Berlin Heidelberg. 2004.
28. Bottou L, Curtis FE, Nocedal J. Optimization methods for large-scale machine learning. arXic.org 2016. Available at: https://arxiv.org/abs/1606.04838.
29. Duchi J, Hazan E, Singer Y. Adaptive subgradient methods for online learning and stochastic optimization. J Mach Learn Res. 2011;12:2121–2159.
30. Kingma D, Ba J. Adam: A method for stochastic optimization. In: International Conference on Learning Representations, 2014.
31. Meier F, Kappler D, Ratliff N, Schaal S. Towards robust online inverse dynamics learning. In: Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems, IEEE, 2016.
32. Ratliff N, Meier F, Kappler D, Schaal S. DOOMED: Direct Online Optimization of Modeling Errors in Dynamics. Technical report, arXiv:1608.00309 [cs.RO], Cornell University Library, 2016.
33. Ioannou P, Sun J. Robust adaptive control, 1st ed. Dover Publications. 2012.
34. Fraccaroli F, Peruffo A, Zorzi M. A new recursive least squares method with multiple forgetting schemes. In: 2015 54th IEEE Conference on Decision and Control (CDC), IEEE, 2015. pp. 3367–3372.

Cite this article as: Ratliff N, Meier F, Kappler D, Schaal S (2016) DOOMED: Direct Online Optimization of Modeling Errors in Dynamics. *Big Data* 4:4, 253–268, DOI: 10.1089/big.2016.0041.

Abbreviations Used

DOOMED = Direct Online Optimization of Modeling Errors in Dynamics
iid = identically distributed
LQRs = linear quadratic regulators
PD = Proportional-Derivative
PID = Proportional-Integral-Derivative
RBD = rigid body dynamics