2020 IEEE International Conference on Robotics and Automation (ICRA)
31 May - 31 August, 2020. Paris, France

Feedback Linearization for Uncertain Systems via Reinforcement Learning

Tyler Westenbroek*, David Fridovich-Keil*, Eric Mazumdar*, Shreyas Arora, Valmik Prabhu,
S. Shankar Sastry, and Claire J. Tomlin

***Abstract*— We present a novel approach to control design for nonlinear systems which leverages model-free policy optimization techniques to learn a linearizing controller for a physical plant with unknown dynamics. Feedback linearization is a technique from nonlinear control which renders the input-output dynamics of a nonlinear plant *linear* under application of an appropriate feedback controller. Once a linearizing controller has been constructed, desired output trajectories for the nonlinear plant can be tracked using a variety of linear control techniques. However, the calculation of a linearizing controller requires a precise dynamics model for the system. As a result, model-based approaches for learning exact linearizing controllers generally require a simple, highly structured model of the system with easily identifiable parameters. In contrast, the model-free approach presented in this paper is able to approximate the linearizing controller for the plant using general function approximation architectures. Specifically, we formulate a continuous-time optimization problem over the parameters of a learned linearizing controller whose optima are the set of parameters which best linearize the plant. We derive conditions under which the learning problem is (strongly) convex and provide guarantees which ensure the true linearizing controller for the plant is recovered. We then discuss how model-free policy optimization algorithms can be used to solve a discrete-time approximation to the problem using data collected from the real-world plant. The utility of the framework is demonstrated in simulation and on a real-world robotic platform.**

I. Introduction

Geometric nonlinear control theory has developed a powerful set of feedback architectures which exploit the underlying structure of a control system to simplify downstream tasks such as trajectory generation and tracking [1, 2]. However, geometric controllers often require an accurate model for the system or a simple parameterization of the dynamics which can be readily identified. Meanwhile, the model-free reinforcement learning literature [3–5] has sought to automatically compute optimal feedback controllers for unknown systems without relying on structural assumptions about the dynamics. However, despite a recent resurgence of research into these methods [6–8], their poor sample complexity has thus-far limited their applicability to many robotics applications. In this paper we unify these disparate approaches by using model-free reinforcement learning algorithms to optimize the performance of a class of geometric controllers for a plant with unknown dynamics, observing improvement in the performance of the trained controllers with practical amounts of data collects from the plant.

This work was supported by HICON-LEARN (design of HIgh CONfidence LEARNing-enabled systems), Defense Advanced Research Projects Agency award number FA8750-18-C-0101, and Provable High Confidence Human Robot Interactions, Office of Naval Research award number N00014-19-1-2066. EECS, UC Berkeley. westenbroekt@berkeley.edu.
* indicates equal contribution.

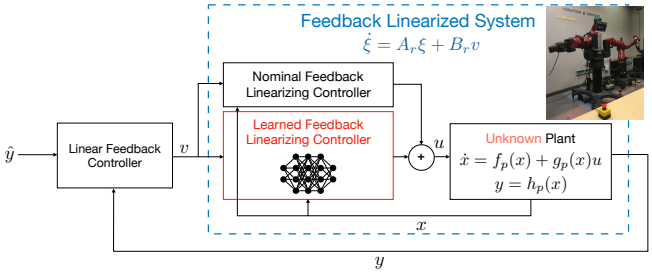

Fig. 1: Schematic diagram of our framework. By learning an appropriate feedback linearizing controller, we render an initially unknown nonlinear system (with state x, output y, and input u) *linear* in an auxiliary input v. The framework can make use of a nominal dynamics model (if available).

Specifically, this paper focuses on a geometric technique called feedback linearization, which renders the input-output dynamics of a nonlinear system *linear* under the application of an appropriately chosen feedback controller. Once a linearizing controller has been constructed, it can be used in conjunction with efficient tools from linear systems theory to generate and track desired output trajectories for the full nonlinear plant [9–11]. Despite its widespread use throughout robotics [1, 2, 12–14] the primary drawback of feedback linearization is that exact cancellation of the system's nonlinearities requires a precise model of the dynamics. Complex phenomena such as friction, aerodynamic drag, and internal actuator dynamics yield nonlinearities which may be challenging to model or identify. While there have been extensive efforts to construct exact linearizing controllers for unknown plants using extensions of adaptive linear control theory [15–22], these methods require a highly structured representation of the system's nonlinearites and employ complicated parameter update schemes to avoid singularities in the learned controller. A more detailed discussion of these methods is provided in Section II-C.

In sharp contrast to the standard methods discussed above, we propose a general framework for learning a linearizing controller for a plant with unknown dynamics using model-free policy optimization techniques. Our approach requires only that the order of the relationship between the inputs and outputs is known, can utilize arbitrary function approximation schemes, and remains singularity free during training. Concretely, our approach begins by constructing a linearizing controller for an approximate dynamics model of the plant (if available), and then augments this fixed model-based controller with a learned affine correction term.

We then define a continuous-time optimization problem over the learned parameters which selects parameters which better linearize the unknown plant. We provide conditions on the structure of the learned component which ensure that the learning problem is (strongly) convex. Our analysis draws connections between the familiar persisency-of-excitation conditions from the adaptive control literature and our formulation. Finally, we discuss how off-the-shelf reinforcement learning algorithms can be used to solve discrete-time approximations to the optimization problem, and demonstrate the utility of our framework by learning linearizing controllers for highly uncertain robotic systems in both simulation and on real-world hardware.

The rest of the paper is structured as follows. Section II introduces feedback linearization and discusses prior approaches for learning linearizing controllers. Section III details our approach, provides our theoretical results, and discusses practical algorithms for solving the learning problem. Our simulated and real-world robotic examples are presented in Section IV, and Section V provides closing remarks and discusses future work. The proofs for Lemma 2 and Theorem 1 can be found in the longer version of this paper [23].

II. Feedback Linearization

This section outlines how to compute an input-output linearizing controller for a known dynamics model and discusses prior data-driven methods for learning a linearizing controller for an unknown plant. Due to space constraints, we refer the interested reader to [1, Chapter 9] for a more complete introduction to feedback linearization. In this paper, we consider square control-affine systems of the form

$$\begin{aligned} \dot{x} &= f(x) + g(x)u \\ y &= h(x), \end{aligned} \tag{1}$$

with sate $x \in \mathbb{R}^n$, input $u \in \mathbb{R}^q$ and output $y \in \mathbb{R}^q$. The mappings $f\colon \mathbb{R}^n \to \mathbb{R}^n$, $g\colon \mathbb{R}^n \to \mathbb{R}^{n\times q}$ and $h\colon \mathbb{R}^n \to \mathbb{R}^q$ are each assumed to be smooth. We restrict our attention to a compact subset $D \subset \mathbb{R}^n$ of the state-space.

A. Single-input single-output systems

We begin by introducing feedback linearization for single-input, single-output (SISO) systems (i.e., $q = 1$). We begin by examining the first time derivative of the output:

$$\begin{aligned} \dot{y} &= \frac{dh}{dx}(x) \cdot \Big(f(x) + g(x)u\Big) \\ &= \underbrace{\frac{dh}{dx}(x) \cdot f(x)}_{L_f h(x)} + \underbrace{\frac{dh}{dx}(x) \cdot g(x)}_{L_g h(x)} u \end{aligned}$$

Here the terms $L_f h(x)$ and $L_g h(x)$ are known as *Lie derivatives* [1], and capture the rate of change of $y = h(x)$ along the vector fields f and g, respectively. In the case that $L_g h(x) \neq 0$ for each $x \in D$, we can apply the control law

$$u(x, v) = \frac{1}{L_g h(x)}(-L_f h(x) + v)\ , \tag{2}$$

which exactly 'cancels out' the nonlinearities of the system and enforces the linear relationship $\dot{y} = v$. After application of this linearizing control law, we can now control the output trajectory $y(t)$ through its first derivative. However if the input does not affect the first time derivative of the output (that is, if $L_g h \equiv 0$) then the control law (2) will be undefined. In general, we can differentiate y multiple times until the input appears. Assuming that the input does not appear the first $\gamma - 1$ times we differentiate the output, the γ-th time derivative of y will be of the form

$$y^{(\gamma)} = L_f^\gamma h(x) + L_g L_f^{\gamma-1} h(x) u.$$

Here, $L_f^\gamma h(x)$ and $L_g L_f^{\gamma-1} h(x)$ are higher order Lie derivatives (see [1, Chapter 9] for more details). If $L_g L_f^{\gamma-1} h(x) \neq 0$ for each $x \in D$ then the control law

$$u(x, v) = \frac{1}{L_g L_f^{\gamma-1} h(x)}\big(-L_f^\gamma h(x) + v\big)$$

enforces the trivial linear relationship $y^{(\gamma)} = v$. We refer to γ as the *relative degree* of the nonlinear system, which is simply the order of its input-output relationship.

B. Multiple-input multiple-output systems

Next, we consider (square) multiple-input, multiple-output (MIMO) systems where $q > 1$. As in the SISO case, we differentiate each of the output channels until at least one input appears. Let γ_j be the number of times we need to differentiate y_j (the j-th entry of y) for at least one input to appear. Combining the resulting expressions for each of the outputs yields an input-output relationship of the form

$$[y_1^{(\gamma_1)}, \ldots, y_q^{(\gamma_q)}]^T = b(x) + A(x)u. \tag{3}$$

Here, the matrix $A(x) \in \mathbb{R}^{q\times q}$ is known as the *decoupling matrix* and the vector $b(x) \in \mathbb{R}^q$ is known as the *drift term*. If $A(x)$ is bounded away from singularity for each $x \in D$ then we observe that the control law

$$u(x, v) = A^{-1}(x)(-b(x) + v) \tag{4}$$

where $v \in \mathbb{R}^q$ yields the decoupled linear system

$$[y_1^{(\gamma_1)}, y_2^{(\gamma_2)}, \ldots, y_q^{(\gamma_q)}]^T = [v_1, v_2, \ldots, v_q]^T, \tag{5}$$

where v_j is the j-th entry of v and $y_j^{(\gamma_j)}$ is the γ_j-th derivative of the j-th output. We refer to $\gamma = (\gamma_1, \gamma_2, \ldots, \gamma_q)$ as the *vector relative degree* of the system. The decoupled dynamics (5) are LTI and can be compactly represented with the *reference model*

$$\dot{\xi}_r = A\xi_r + Bv_r, \tag{6}$$

where $\xi_r = (y_1, \dot{y}_1, \ldots, \ldots, y_1^{\gamma_1-1}, \ldots, y_q, \ldots, y_q^{\gamma_q-1})$ and $A \in \mathbb{R}^{|\gamma|\times|\gamma|}$ and $B \in \mathbb{R}^{|\gamma|\times q}$ are dynamics matrices with appropriate entries. The reference model can then be used to construct a desired trajectory $\xi_r(\cdot)$ for the output of the nonlinear system (and its derivatives), and to construct linear feeback controllers which can be used in conjunction with (4) to track the reference (see [1, Theorem 9.14] for details).

C. Constructing linearizing controllers with data

The fundamental challenge model-based methods face when constructing a linearizing controller for an unknown plant is that it is very difficult to identify an estimate for the system's decoupling matrix (or possibly its inverse) which is guaranteed to be invertible. The predominant approaches for learning linearizing controllers are founded on the linear model reference adaptive control (MRAC) literature [15], and seek to recursively improve an estimate for the true linearizing controller using data collected from the plant [15–19, 24–26]. These methods update the parameters for estimates of the decoupling matrix (or its inverse) online, but require that the estimated matrix remain invertible at all times. Typically, projection-based update rules are used to keep the parameters of the estimated matrix in a region which is singularity-free. However, the construction of these bounds requires a highly accurate yet simple parameterization of the system, and assumes that the true parameters of the system lie within some nominal set.

One alternative approach [27–29] is to perturb the estimated linearizing control law to avoid singularities. These methods enable the use of more general function approximation schemes but sacrifice some tracking performance. Recently, non-parametric function approximators have been been used to learn a linearizing controller [20, 30], but these methods still require structural assumptions to avoid singularities. In the following section we use model-free policy optimization algorithms to update the paramters of a learned linearizing controller while avoiding singularities.

III. Directly Learning a Linearizing Controller

Our goal is to learn a linearizing controller for the plant

$$\begin{aligned} \dot{x}_p &= f_p(x_p) + g_p(x_p)u_p \\ y_p &= h_p(x_p) \end{aligned} \tag{7}$$

which is unknown. Our approach can incorporate nominal model

$$\begin{aligned} \dot{x}_m &= f_m(x_m) + g_m(x_m)u_m \\ y_m &= h_m(x_m) \end{aligned} \tag{8}$$

which represents our "best guess" for the true dynamics of the plant. We will assume that the plant and model have well-defined relative degrees $(\gamma_1^p, \dots, \gamma_q^p)$ and $(\gamma_1^m, \dots, \gamma_q^m)$, respectively, on some chosen set $D \subset \mathbb{R}^n$. We make the following assumption about our plant and model:

Assumption 1: The model system (8) and plant (7) have the same relative degree on D in the sense that $(\gamma_1^p, \dots, \gamma_q^p) = (\gamma_1^m, \dots, \gamma_q^m)$.

This is a rather mild assumption, as the order of the relationship between the inputs and outputs of the plant can usually be inferred from first principles, even without a perfect model for the dynamics of the system. For example, for Lagrangian systems, such as the manipulator arms we consider in Section IV, we know the torques applied by an actuator produce an acceleration in the joints, even if we don't know the exact relationship between the two quantities. Moreover, with this assumption in place, we know there are linearizing controllers of the form

$$\begin{aligned} u_m(x, v) &= \beta_m(x) + \alpha_m(x)v \\ u_p(x, v) &= \beta_p(x) + \alpha_p(x)v \end{aligned}$$

for the model and plant, respectively, which match the input-output dynamics of both systems to a common reference model (6). Here, $\beta_p(x), \beta_m(x) \in \mathbb{R}^q$ and $\alpha_p(x), \alpha_m(x) \in \mathbb{R}^{q\times q}$. While we do not know u_p *a priori*, we do know that

$$\begin{aligned} \beta_p(x) &= \beta_m(x) + \Delta\beta(x) \\ \alpha_p(x) &= \alpha_m(x) + \Delta\alpha(x) \end{aligned}$$

for some continuous functions $\Delta\beta$ and $\Delta\alpha$. We construct the following parameterized estimates for these functions:

$$\Delta\beta(x) \approx \beta_{\theta_1}(x) \qquad \Delta\alpha(x) \approx \alpha_{\theta_2}(x)$$

Here, $\theta_1 \in \Theta_1 \subset \mathbb{R}_1^{K_1}$ and $\theta_2 \in \Theta_2 \subset \mathbb{R}^{K_2}$ are parameters to be trained by running experiments on the plant. We will assume that Θ_1 and Θ_2 are convex sets, and we will frequently abbreviate $\theta = (\theta_1, \theta_2) \in \Theta_1 \times \Theta_2 := \Theta$. We assume that β_{θ_1} and α_{θ_2} are continuous in x and continuously differentiable in θ_1 and θ_2, respectively.

Altogether, for a given $\theta = (\theta_1, \theta_2) \in \Theta$ our estimate for the controller which exactly linearizes the plant is given by

$$\hat{u}_\theta(x, v) = [\beta_m(x) + \beta_{\theta_1}(x)] + [\alpha_m(x) + \alpha_{\theta_2}(x)]v \tag{9}$$

When no prior information about the dynamics of the plant is available (other than its vector relative degree), we simply set $\beta_m \equiv 0$ and $\alpha_m \equiv 0$ in the above expression. Next we define an optimization problem which selects the parameters for the learned controller which best linearize the plant.

A. Continuous-time optimization problem

From Section II we know that the input-output dynamics of the plant are of the form

$$y^{(\gamma)} = b_p(x) + A_p(x)u \tag{10}$$

where the terms b_p and A_p are unknown to us, and we have written the highest order derivatives of the outputs as $y^{(\gamma)} = (y_1^{(\gamma_1)} \dots, y_q^{(\gamma_q)})^T$ to simplify notation. Under application of $\hat{u}_\theta$ the dynamics are given by:

$$y^{(\gamma)} = \underbrace{b_p(x) + A_p(x)\hat{u}_\theta(x, v)}_{W_\theta(x,v)} \tag{11}$$

Since our goal is to find parameters which linearize the plant, we want to find $\theta^* \in \Theta$ such that $W_{\theta^*}(x, v) \approx v$ for each $x \in D$ and $v \in \mathbb{R}^q$. Thus, we define the point-wise loss $\ell\colon \mathbb{R}^n \times \mathbb{R}^q \times \mathbb{R}^{K_1+K_2} \to \mathbb{R}$ by

$$\ell(x, v, \theta) = \|v - W_\theta(x, v)\|_2^2, \tag{12}$$

which provides a measure of how well the learned controller $\hat{u}_\theta$ linearizes the plant at the state x when the virtual input v is applied to the linear reference model. Next, we specify a probability distribution X over $\mathbb{R}^n$ with support

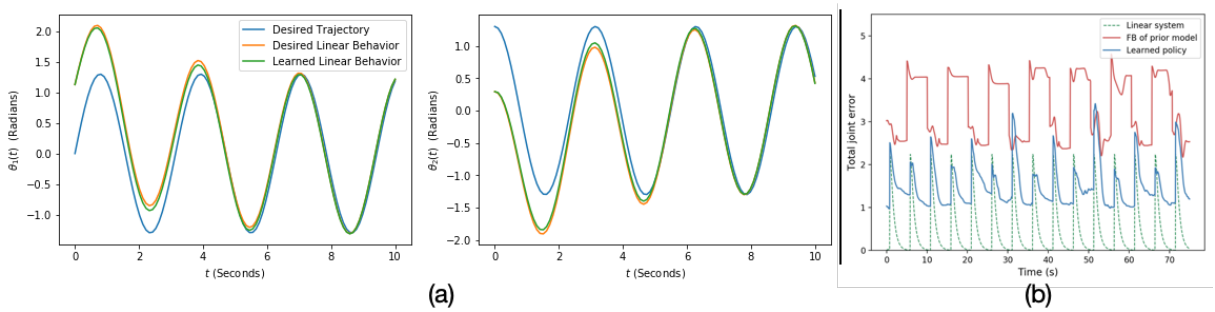

Fig. 2: (a) Depiction of the tracking task for the double pendulum. The blue curves represent the desired joint angles over time. The orange and green curves represent trajectories generated by the ideal feedback linearizing controller (perfect model information) and our learned policy. Both trajectories begin with an initial tracking error, but quickly converge to the desired motion. (b) Total ℓ_2 error on a set-point tracking task for the Baxter robot after 104 minutes of training for the desired linear system is (dotted, green), the nominal feedback linearizing controller (red), and our learned controller (blue)

$D \subset \mathbb{R}^n$ and let V be the uniform distribution over the set $\{v \in \mathbb{R}^q \colon \|v\| \leq 1\}$. We then define the weighted loss

$$L(\theta) = \mathbb{E}_{x \sim X, v \sim V}\, \ell(x, v, \theta) \tag{13}$$

and select our optimal choice for the parameters of the learned controller via the following optimization:

$$(\mathbf{P}) \colon \quad \min_{\theta \in \Theta} L(\theta) \tag{14}$$

Here, the distribution X models our preference for having an accurate linearizing controller at different points in the state-space, and the uniformity of V ensures that optimal solutions of **P** accurately linearize the plant for all possible choices of the virtual input to the reference model. The primary challenge in solving **P** is that we do not know the terms in $W_\theta(x, v)$, since we do not know $A_p(x)$ and $b_p(x)$. However, using the relationship (11), we can query $W_\theta(x, v)$ by measuring $y^{(\gamma)}$ when different inputs are applied to the plant. Thus, **P** can be solved by running experiments on the plant and using any stochastic optimization method which only requires access to the point-wise loss (13). While we focus on the use of policy-gradient reinforcement learning algorithms below, there are many possible model-free approaches for solving **P** which merit further investigation. Importantly, since we have abstracted away the need for a model when solving **P**, we can iteratively improve the performance of our learned controller without requiring that it remain invertible at each stage of the optimization. Next, we discuss when we can recover the true linearizing controller for the plant by solving **P**:

Lemma 1: Suppose that there exists $\theta^* \in \Theta$ such that $\hat{u}_{\theta^*}(x, v) = u_p(x, v)$ for each $x \in D$ and $v \in V$. Then θ^* is a globally optimal solution of **P**.

Proof: Note that if $u_{\theta^*}(x, v) = u_p(x, v)$ for each $x \in X$ and $v \in V$ then $L(\theta) = 0$. Moreover, we clearly have $L(\theta) \geq 0$ for each $\theta \in \Theta$. Thus, θ^* must be a global minimizer of the optimization (14). ∎

However, **P** is generally non-convex meaning that in practice we can only hope to find locally optimal solutions to the problem. Thus, we seek conditions which simplify the structure of the optimization. The standard convergence proofs in the adaptive control literature assume the learned controller is of the form

$$\beta_{\theta_1}(x) = \sum_{k=1}^{K_1} \theta_k^1 \beta_k(x) \qquad \alpha_{\theta_2}(x) = \sum_{k=1}^{K_2} \theta_k^2 \alpha_k(x) \tag{15}$$

where $\{\beta_k\}_{k=1}^{K_1}$ and $\{\alpha_k\}_{k=1}^{K_2}$ are nonlinear continuous functions. When we adopt this structure our optimization becomes convex:

Lemma 2: Assume that β_{θ_1} and α_{θ_2} are of the form (15). Then **P** is convex. Moreover, if $\{\beta_k\}_{k=1}^{K_1}$ and $\{\alpha_k\}_{k=1}^{K_2}$ are each linearly independent then **P** is strongly convex.

Thus, when our learned controller takes the form (15) we can reliably use iterative techniques to find its globally optimal solution. The proof of Lemma 2, which is given in the accompanying technical report, shows that L is actually quadratic in the parameters when the learned controller is of the form (15). The key property being exploited here is that $W_\theta(x, v)$ becomes affines in θ when the controller is linear in the parameters. Our conditions on X and V are analogous to the persistency of excitation results from the adaptive control literature [17, Chapter 2], and ensure that each component of the learned controller is excited while solving **P**. This is the underlying reason why the optimization becomes strongly convex when the components of our controller are linearly independent. Taken together, the preceding lemmas imply our main theoretical result:

Theorem 1: Suppose that for some $\theta^* \in \Theta$ we have $\hat{u}_{\theta^*}(x, v) = u_p(x, v)$ for each $x \in D$ and $v \in \mathbb{R}^q$. Further assume that the learned controller is of the form (15) and that the sets $\{\beta_k\}_{k=1}^{K_1}$ and $\{\alpha_k\}_{k=1}^{K_2}$ are linearly independent. Then θ^* is the unique global (and local) minimizer of **P**.

There are a number of well-studied bases, such as polynomials or radial basis functions, which can be used to approximate any continuous function to a desired degree of accuracy. Thus, by including enough terms in $\{\beta_k\}_{k=1}^{K_1}$ and $\{\alpha_k\}_{k=1}^{K_2}$ we can theoretically recover u_p to a pre-specified precision by solving **P**. However, for high dimensional systems, the number of terms required in such expansions can become prohibitively large. Other architectures, such as multi-layer feed-forward neural networks, yield more compactly represented function approximation schemes but complicate the analysis of **P**. However, since our approach does not suffer from singularities during the optimization process, in practice

we can still use these powerful function approximators to find an improved linearizing controller for high-dimensional systems with unknown dynamics. We next demonstrate this point by discussing how policy optimization algorithms can be used to solve a discrete-time approximation to **P**.

B. Discrete-time approximations and reinforcment learning

While the theory and optimization problem we developed in the previous section are in continuous time, many real world plants have actuators which can only be updated at a fixed sampling frequency. Thus in this section we formulate a discrete time approximation to **P** which we cast as a canonical reinforcement learning problem. Unfortunately, the analysis of linearizable systems in the sampled-data setting becomes significantly more complex [31], which is why the majority of the literature on adaptive control for linearizable systems is formulated in continuous time.

Letting $\Delta t > 0$ denote the sampling rate, we set $t_k = k\Delta t$ for each $k \in \mathbb{N}$. Next, letting $x(\cdot)$ denote the state trajectory of the plant, we then denote $x_k = x(t_k)$ and let u_k denote the control applied on the interval $[t_k, t_{k+1})$. This yields the following difference equation for the dynamics of the plant:

$$x_{k+1} = x_k + \underbrace{\int_{t_k}^{t_{k+1}} f_p(x(t)) + g_p(x(t))u_k d\tau}_{F(x_k,u_k)}$$

The map $F_p\colon \mathbb{R}^n \times \mathbb{R}^q \to \mathbb{R}^n$ will generally no longer be affine in the control. Similarly, we let $\xi(\cdot) = (y_1(\cdot), \dots, y_1^{(\gamma_1-1)}(\cdot), \dots, y_q(\cdot), \dots y_q^{(\gamma_q-1)}(\cdot))$ denote the solution to the linearized portion of the state, and set $\xi_k = \xi(t_k)$ for each iterate. Now, if we integrate (10) over the interval $[t_k, t_{k+1})$ we obtain a difference equation for the outputs of the form

$$\xi_{k+1} = e^{A\Delta t}\xi_k + H(x_k, u_k) \tag{16}$$

where again the mapping $H\colon \mathbb{R}^n \times \mathbb{R}^q \to \mathbb{R}^\gamma$ will generally no longer be affine in u_k. This is the primary hurdle to applying the theory developed in the previous section to the current setting. Nevertheless, we can still attempt to enforce an approximate linear relationship between a virtual input v_k and successive iterates of the outputs. Letting $\bar{A} = e^{A\Delta t}$ and $\bar{B} = \int_0^{\Delta t} e^{At}B dt$, the sampled-data version of the reference model (6) becomes

$$\xi_{k+1} = \bar{A}\xi_k + \bar{B}v_k,$$

which is the ideal linear behavior we would like to enforce by applying the control $u_k = \hat{u}_\theta(x_k, v_k)$. To encourage actions which better track the discrete-time reference model, we apply the point-wise loss $\bar{\ell}\colon \mathbb{R}^n \times \mathbb{R}^q \times \mathbb{R}^q \to \mathbb{R}$ where

$$\bar{\ell}(x_k, u_k, v_k) = \|\bar{B}v_k - H(x_k, u_k)\|_2^2.$$

We can calculate this quantity using (16) and by observing ξ_k and ξ_{k+1}, which can each be calculated by numerically differentiating the outputs from the plant. This loss provides a measure of how well the control u_k enforces the desired change in the state of the linear system (as specified by v_k) at the state x_k. We then define the following reinforcement learning problem over the parameters of $\hat{u}_\theta$:

$$\min_{\theta\in\Theta} \mathbb{E}_{x_0\sim X, v_k\sim V, w_k\sim\mathcal{N}(0,\sigma_w^2)}\left[\sum_{k=1}^{N} \bar{\ell}(x_k, u_k, v_k)\right]$$
$$\text{subject to:} \quad x_{k+1} = x_k + F(x_k, u_k), \quad x_0 = x_0$$
$$u_k = \hat{u}_\theta(x_k, v_k) + w_k$$

Here, we sample initial conditions for the problem using our desired state distribution X and sample inputs to the linear system according to V at each time step. The zero mean added noise w_k is used to encourage exploration, and to make the effects of the policy random. Finally, N is the length of the training episodes. This problem can be solved to local optimality using standard reinforcement learning algorithms [32–35] and by running experiments on the real-world hardware to evaluate $\bar{l}(x_k, u_k, v_k)$. While it is not immediately clear how to extend the theoretical results from Section III-A to the present case, we intend to address this issue in a forthcoming article.

IV. Examples

We now use our approach to learn feedback linearizing policies for three different systems and use the learned policies to construct tracking controllers as in [1, Theorem 9.14]. In all cases, the input to the parameterized policy replaces all angles with their sine and cosine.

A. Simulations

1) Double pendulum with polynomial policies: We first test our approach on a fully actuated double pendulum with state $x = [\theta_1, \theta_2, \omega_1, \omega_2]^T$, output $y = [\theta_1, \theta_2]^T$, where θ_1 and θ_2 represent the angles of the two joints, with angular rates ω_1 and ω_2 respectively. The system has two inputs u_1 and u_2 that control the torque at both joints. Although the system is relatively low dimensional and fully actuated, it is highly nonlinear [36].

For the learning problem, we give the nominal model inaccurate estimates for both the length and mass of each of the arms. Specifically, we scale each of the parameters to $1/2$ their true values. The learned controller is comprised of 150 radial basis functions, which are centered randomly throughout the state-space. The learned controller is linear in its parameters, so as to agree with Theorem 1. We use the REINFORCE algorithm [32], and baseline state-value estimates with the average reward over all states. At each iteration (or epoch) we collect 50 rollouts of 0.25 seconds each, and we train for 500 epochs. Figure 2 (a) presents the result of using our learned controller to track a desired sinusoidal trajectory, and compares it to the performance of the exact linearizing controller for the system. The difference between the performance our learned controller and the theoretical ideal observed to be very small. We do not plot the trajectory generated by the nominal model-based controller, since it immediately diverges from the desired behavior. Similar results were observed using the same learned controller to track other desired trajectories for the system. The linear

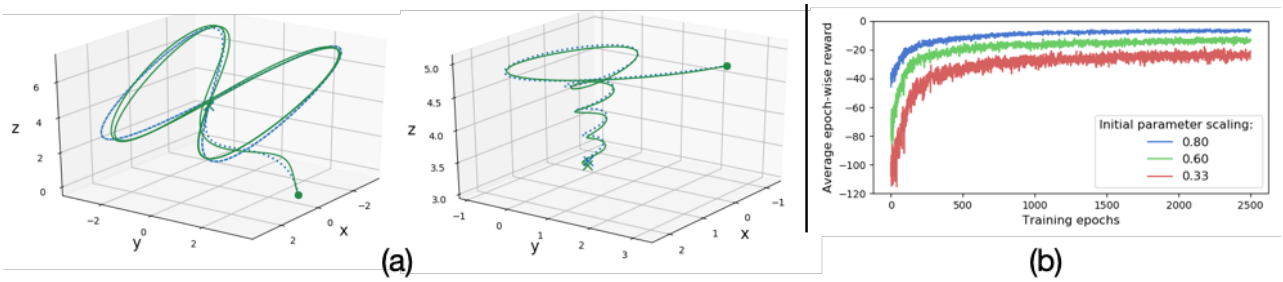

Fig. 3: (a) The performance of our learned linearizing controller on two high-performance quadrotor tracking tasks when the parameters of the inital dynamics model are scaled by a factor of 0.6. The first task is a figure-eight, and the second is a corkscrew maneuver. In both maneuvers the quadrotor also tracks an oscillating reference in the yaw angle. In both figures, the green trajectory is the one taken by the learned controller and the blue trajectory is that of the true linearizing controller for the system. In both cases, the learned controller closely matches the desired behavior. (b) Learning curves for learned linearizing controllers which are instantiated with nominal dynamics models with three levels of scaling of the true dynamics parameters.

feedback matrix used in the tracking controller was obtained by solving an infinite horizon LQR problem where the state deviation was penalized 10 times more than the magnitude of the input.

2) 14D quadrotor with neural network policies: Our second simulation environment uses the quadrotor model and feedback linearization controller proposed in [37], which makes use of dynamic extension [1]. In particular, the states for the model are $(x, y, z, \psi, \theta, \varphi, \dot{x}, \dot{y}, \dot{z}, p, q, r, \xi, \zeta)$ where x, y and z are the Cartesian coordinates of the quadrotor, and ψ, θ and φ represent the roll, pitch and yaw of the quadrotor, respectively. The next six states represent the time derivatives of these state: $\frac{d}{dt}(x, y, z, \psi, \theta, \varphi) = (\dot{x}, \dot{y}, \dot{z}, p, q, r)$. Finally, ξ and ζ are the extra states obtained from the dynamic extension procedure. The outputs for the model are the x, y, z and ψ coordinates.

The inaccurate nominal dynamics model was constructed by multiplying the mass and inertial constants of the true system by factors of 0.33, 0.6 and 0.8 for different experiments. The trained policies were feed-forward neural networks with $\tanh$ activations with 2 hidden layers of width 64. For each training epoch, 50 rollouts of length 25 were collected and the parameters were updated using PPO. We trained both policies for 2500 epochs. Figure 3 (b) illustrates how a better initial dynamics model leads to faster learning of an accurate linearizing controller. In particular, the reward-per-epoch is plotted for policies trained with the three scaling factors indicated above. We observe that worse initial models result in worse policy performance, given the same network architecture and training time. Figure 3 (a) demonstrates the ability of the learned controller to overcome significant model mismatch (scaling factor of 0.6) to match the desired linear tracking behavior.

B. Robotic experiment: 7-DOF manipulator arm

We also evaluate our approach in hardware, on a 7-DOF Baxter robot arm. The dynamics of this 14-dimensional system are extremely coupled and nonlinear. Taking the 7 joint angles as output y, however, the system is input-output linearizable with relative degree two. We use the system measurements (i.e., masses, link lengths, etc.) provided with Baxter's pre-calibrated URDF [38] and the OROCOS Kinematics and Dynamics Library (KDL) [39] to compute a nominal feedback linearizing control law.

This nominal controller suffers from several inaccuracies. First, Baxter's actuators are series-elastic, meaning that each joint contains a torsion spring [40] which is unmodeled, and the URDF itself may not be perfectly accurate. Second, the OROCOS solver is numerical, which can lead to errors in computing the decoupling matrix and drift term. Finally, our control architecture is implemented in the Robot Operating System [41], which can lead to minor timing inconsistency.

We use the PPO algorithm to tune the parameters of a 128×2 neural network with $\tanh$ activations for the learned component of the controller. For each training epoch, 1250 rollouts of one timestep (0.05 s) each were collected. We trained for 100 epochs, which took 104 minutes. Figure 2 (b) summarizes typical results on tracking a square wave reference trajectory for each joint angle with period 5s. The nominal feedback linearized model from OROCOS has significant steady-state error. Our learned approach significantly reduces, but does not eliminate, this error. We conjecture the remaining error is a sign that the (relatively small) neural network may not be sufficiently expressive.

V. Discussion

While the methods we have presented avoid some of the challenges model-based methods face when trying to construct a linearizing controller for an unknown plant, we feel the techniques presented here should be used to complement model-based design approaches rather than replace them. As we observed empirically, a reasonable (though ultimately inaccurate) model can be used to provide a better starting point for the learning process. Thus, we feel that in practice our approach should be used primarily to overcome difficult to model non-linearities, and should be used in combination with a simple nominal dynamics models with parameters which are easily identified. Investigating the performance of different learning algorithms in real-world settings merits significant future research effort. On the theoretical side, we feel the general way in which we constructed the optimization problem in III-A provides a foundation for combining model-free policy optimization techniques with geometric control architectures. Future work will investigate how to optimize over different classes of controllers by extending the approach presented here.

REFERENCES

[1] S. Sastry. *Nonlinear systems: analysis, stability, and control*. Vol. 10. Springer Science & Business Media, 1999.

[2] A. Isidori. *Nonlinear control systems*. Springer Science & Business Media, 2013.

[3] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-dynamic programming*. Athena Scientific, 1996.

[4] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. 2018.

[5] R. S. Sutton et al. "Policy gradient methods for reinforcement learning with function approximation". *Advances in neural information processing systems*. 2000.

[6] J. Schulman et al. "Trust region policy optimization". *International conference on machine learning*. 2015.

[7] T. P. Lillicrap et al. "Continuous control with deep reinforcement learning". *arXiv preprint arXiv:1509.02971* (2015).

[8] J. Schulman et al. "Proximal policy optimization algorithms". *arXiv preprint arXiv:1707.06347* (2017).

[9] P. Martin, R. M. Murray, and P. Rouchon. "Flat systems, equivalence and trajectory generation" (2003).

[10] R. E. Kalman et al. "Contributions to the theory of optimal control". *Bol. soc. mat. mexicana* 5.2 (1960).

[11] F. Borrelli, A. Bemporad, and M. Morari. *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.

[12] J. W. Grizzle, G. Abba, and F. Plestan. "Asymptotically stable walking for biped robots: Analysis via systems with impulse effects". *IEEE Transactions on automatic control* 46.1 (2001).

[13] A. D. Ames et al. "Rapidly exponentially stabilizing control lyapunov functions and hybrid zero dynamics". *IEEE Transactions on Automatic Control* 59.4 (2014).

[14] D. Mellinger and V. Kumar. "Minimum snap trajectory generation and control for quadrotors". *2011 IEEE International Conference on Robotics and Automation*. IEEE. 2011.

[15] S. Sastry and M. Bodson. *Adaptive control: stability, convergence and robustness*. Courier Corporation, 1989.

[16] J. J. Craig, P. Hsu, and S. S. Sastry. "Adaptive control of mechanical manipulators". *The International Journal of Robotics Research* 6.2 (1987).

[17] S. S. Sastry and A. Isidori. "Adaptive control of linearizable systems". *IEEE Transactions on Automatic Control* 34.11 (1989).

[18] K. Nam and A. Araposthathis. "A model reference adaptive control scheme for pure-feedback nonlinear systems". *IEEE Transactions on Automatic Control* 33.9 (1988).

[19] I. Kanellakopoulos, P. V. Kokotovic, and A. S. Morse. "Systematic design of adaptive controllers for feedback linearizable systems". *1991 American Control Conference*. IEEE. 1991.

[20] J. Umlauft et al. "Feedback linearization using Gaussian processes". *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE. 2017.

[21] G. Chowdhary et al. "Bayesian nonparametric adaptive control using gaussian processes". *IEEE Transactions on Neural Networks and Learning Systems* 26.3 (2014).

[22] G. Chowdhary et al. "Bayesian nonparametric adaptive control of time-varying systems using Gaussian processes". *2013 American Control Conference*. IEEE. 2013.

[23] T. Westenbroek et al. "Feedback Linearization for Unknown Systems via Reinforcement Learning". *arXiv preprint arXiv:1910.13272* (2019).

[24] J. T. Spooner and K. M. Passino. "Stable adaptive control using fuzzy systems and neural networks". *IEEE Transactions on Fuzzy Systems* 4.3 (1996).

[25] F.-C. Chen and H. K. Khalil. "Adaptive control of a class of nonlinear discrete-time systems using neural networks". *IEEE Transactions on Automatic Control* 40.5 (1995).

[26] A. Yesildirek and F. L. Lewis. "Feedback linearization using neural networks". *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*. Vol. 4. IEEE. 1994.

[27] E. B. Kosmatopoulos and P. A. Ioannou. "A switching adaptive controller for feedback linearizable systems". *IEEE Transactions on automatic control* 44.4 (1999).

[28] E. B. Kosmatopoulos and P. A. Ioannou. "Robust switching adaptive control of multi-input nonlinear systems". *IEEE transactions on automatic control* 47.4 (2002).

[29] C. P. Bechlioulis and G. A. Rovithakis. "Robust adaptive control of feedback linearizable MIMO nonlinear systems with prescribed performance". *IEEE Transactions on Automatic Control* 53.9 (2008).

[30] J. Umlauft and S. Hirche. "Feedback Linearization based on Gaussian Processes with event-triggered Online Learning". *IEEE Transactions on Automatic Control* (2019).

[31] J. Grizzle and P. Kokotovic. "Feedback linearization of sampled-data systems". *IEEE Transactions on Automatic Control* 33.9 (1988).

[32] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. 1st. Cambridge, MA, USA: MIT Press, 1998.

[33] J. Schulman et al. "Trust region policy optimization". *International Conference on Machine Learning*. 2015.

[34] J. Schulman et al. "Proximal Policy Optimization Algorithms". *CoRR* ().

[35] D. Silver et al. "Deterministic Policy Gradient Algorithms". *Proceedings of the 31st International Conference on Machine Learning*. Proceedings of Machine Learning Research. 2014.

[36] T. Shinbrot et al. “Chaos in a double pendulum”. *American Journal of Physics* 60.6 (1992).

[37] S. A. Al-Hiddabi. “Quadrotor control using feedback linearization with dynamic extension”. *2009 6th International Symposium on Mechatronics and its Applications*. IEEE. 2009.

[38] R. Robotics. “Baxter”. *Retrieved Jan* 10 (2013).

[39] H. Bruyninckx. “Open robot control software: the OROCOS project”. *Proceedings 2001 ICRA. IEEE international conference on robotics and automation (Cat. No. 01CH37164)*. Vol. 3. IEEE. 2001.

[40] R. L. Williams II. “Baxter Humanoid Robot Kinematics© 2017 Dr. Bob Productions Robert L. Williams II, Ph. D., williar4@ ohio. edu Mechanical Engineering, Ohio University, April 2017” (2017).

[41] M. Quigley et al. “ROS: an Open-Source Robot Operating System”. *ICRA Workshop on Open Source Software*. 2009.