A General Framework for Structured Learning of Mechanical Systems

Jayesh K. Gupta[1,*], Kunal Menda[1,*], Zachary Manchester[1], Mykel J. Kochenderfer[1]

***Abstract*— Learning accurate dynamics models is necessary for optimal, compliant control of robotic systems. Current approaches to white-box modeling using analytic parameterizations, or black-box modeling using neural networks, can suffer from high bias or high variance. We address the need for a flexible, gray-box model of mechanical systems that can seamlessly incorporate prior knowledge where it is available, and train expressive function approximators where it is not. We propose to parameterize a mechanical system using neural networks to model its Lagrangian and the generalized forces that act on it. We test our method on a simulated, actuated double pendulum. We show that our method outperforms a naive, black-box model in terms of data-efficiency, as well as performance in model-based reinforcement learning. We also conduct a systematic study of our method's ability to incorporate available prior knowledge about the system to improve data efficiency.**

I. Introduction

When engineering a controller for a robotic domain, we often rely on accurate models of the system we aim to control [1], [2]. One faces the perennial question of whether to seek out domain expertise, or to take a data-driven, black-box approach to constructing such a model. The former approach would make assumptions about the system, such as its kinematic structure, inertia properties, and assumptions regarding the forces acting on the system, leaving only a few parameters for data-driven calibration [3], [4]. The latter approach [5], [6], [7], on the other hand, would treat the system's equations of motion as any other function that the tools of machine learning are capable of fitting. That is, this approach would reduce the problem of learning the system dynamics to that of optimizing the parameters of an expressive function class, such as a neural network, in order to minimize some form of a prediction loss.

Both of the aforementioned approaches have limitations, summarized in Figure 1. The assumptions made by the domain expert may not capture hard-to-model effects, leading to inaccuracies via *model bias*. On the other hand, while the black-box approach of training an overparameterized function class may be capable of capturing the phenomena present in the training data, it often requires infeasibly large amounts of training data to achieve generalization, due to *model variance*.

Ideally, we would want to take a *gray-box* approach that models the parts of the system for which we believe that models are accurate, while capturing the difficult-to-model system dynamics by training a highly expressive function class such as a neural network from data. However, typical approaches to fitting system dynamics with neural networks do not allow us to easily incorporate prior knowledge.

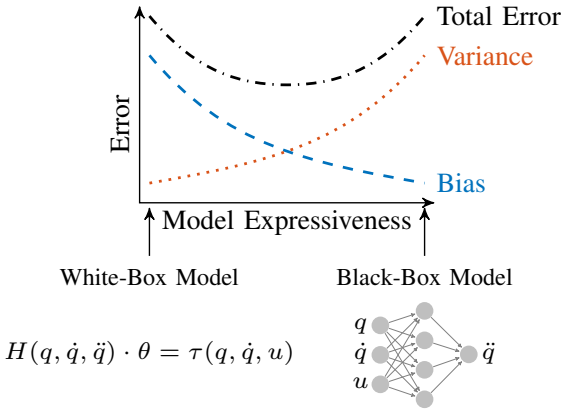

Fig. 1: The bias-variance tradeoff as a function of model expressiveness.

Although there have been proposals to learn *offset functions* that correct for model bias, they are still restricted to specific use cases [8], [9].

In this work, we propose a structured approach to gray-box modeling of mechanical systems. Instead of treating the equations of motion governing the system as an arbitrary functional mapping, we make the single assumption that the system conforms to *Lagrangian dynamics*. Consequently, we propose to learn the *Lagrangian* of the system, as well as a structured representation of the forces that act on the system. By doing so, we can parameterize the space of all mechanical systems in a structured and modular manner. From these two functions, we can evaluate the accelerations acting on the system by using the *Euler-Lagrange equation*.

The method we present has the flexibility to incorporate as much prior knowledge as in a white-box approach, and, in the event of having no prior knowledge whatsoever, the method remains as expressive as a black-box approach. However, we will show that even in this scenario, our method achieves lower model variance than naive approaches to black-box modeling owing to the constraints of physical compliance.

By modularizing the hypothesis class into functions that represent the system's Lagrangian and functions that represent the forces acting on it, we gain the principal benefit of being able to incorporate prior knowledge where it is available. For example, say we only know *a priori* that the system is *control affine* (i.e., there exists a linear mapping between actuator inputs and the torques applied to the system) and that no forces other than gravity act on the system. In such a situation, we can use an expressive function class to model the system's Lagrangian, while using a much more restricted function class to model the control-affine torques, thereby reducing model variance.

*Authors contributed equally.

[1]Jayesh K. Gupta, Kunal Menda, Zachary Manchester and Mykel J. Kochenderfer are at Stanford University, Stanford, CA 94305, USA {jayeshkg,kmenda,zacmanchester,mykel}@stanford.edu

This paper is organized as follows: Section II, overviews related approaches to modeling the dynamics of mechanical systems. Section III, describes the foundations of Lagrangian dynamics, the theoretical background to gradient-based model fitting, as well as approaches to model-based control and reinforcement learning. Next, Section IV, describes our modular parameterization of mechanical system dynamics, as well as our methodology for fitting such models to data. Finally, Section V, uses an actuated double-pendulum as a test-bed to demonstrate that our method can seamlessly incorporate prior knowledge in a flexible and expressive hypothesis class. Additionally, we demonstrate that using our approach over naive approaches to black-box modeling leads to improved model-based reinforcement learning. Source code demonstrating our work can be found at `https://github.com/sisl/mechamodlearn`.

II. Related Work

System identification has been a field of much interest to the robotics and controls community for decades. More recently, the techniques of *inverse kinematics* and *feedback linearization* have been used to actuate robots to track desired motion trajectories. Such approaches, like many other model-based controllers, rely on high-gain Proportional-Derivative controllers to compensate for inaccurate dynamics models, leading to non-compliant and potentially dangerous behavior [8]. Additionally, model-learning is of interest to the reinforcement learning community because of its data-efficiency compared to model-free methods [10] and potential for transfer during continual learning [11], [12]. Hence, learning dynamics models using data, or simply fine-tuning models crafted from prior knowledge, has been approached in several ways.

White-box approaches to system identification attempt to simply calibrate the kinematic and inertial properties of an otherwise analytically specified dynamics model [3]. Typically starting with an analytic model or the topology of the system, the system dynamics can be cast as follows:

$$H(q, \dot{q}, \ddot{q}) \cdot \theta = \tau(q, \dot{q}, u) \tag{1}$$

Here, $q, \dot{q}$, and $\ddot{q}$ are the generalized coordinates, velocities, and accelerations of the system, u are the actuator inputs, and τ is the resulting generalized torque applied to the system. The functions $H(q, \dot{q}, \ddot{q})$ and $\tau(q, \dot{q}, u)$ are given by the analytic model of the system, which are related by the learnable kinematic and inertial parameters θ. These parameters may be learned from data using least-squares regression.

The problem with white-box approaches is that analytic specifications of many poorly understood phenomena affecting the dynamics are not always accurate, such as interactions involving contact, fluids, friction, or wear-and-tear of robotic joints. Simplified analytic models of such phenomena are likely to introduce *model bias*, in that no set of parameters can be selected for the model that would accurately represent the phenomenon.

At the other end of the spectrum, black-box approaches to system dynamics typically use highly expressive function classes to capture the full range of phenomena in the data [5], [6], [7]. A neural network is commonly chosen to express the function class, taking as inputs $q, \dot{q}$ and u, and outputting the predicted acceleration, $\ddot{q}$, as depicted in Figure 1. However, highly expressive function classes such as neural networks are typically *overparameterized*. This means that, given a finite amount of data, many choices of parameters could perfectly fit the data. As a consequence, had the data used in training been slightly different, one would expect the predictions in poorly sampled parts of the input space to change dramatically. This phenomenon is typically referred to as *overfitting*, leading to prediction error due to *model variance*. Model variance is remedied by supplying large amounts of training data, though this is typically impractical in real-world robotic scenarios. There are black-box approaches that fit limited data using Gaussian processes [13], though such approaches do not scale well with dataset size or state-space dimensionality.

As summarized in Figure 1, the white-box approach of relying solely on models leads to model bias, while taking the black-box approach of relying on a highly expressive function class to learn dynamics leads to model variance. There have been attempts to take *gray-box* approaches to combining prior knowledge with the expressive function class such as a neural network, or the data-efficiency of Gaussian Processes, attempting to make the ideal compromise between bias and variance. One such approach is to take a gradient-based approach to learning an 'offset function' that captures the discrepancy between the white-box dynamics model and the true dynamics [8]. By using a black-box model such as a neural network to represent the offset function, they are able to capture the effects of phenomena not accounted for in the analytic model. Another approach learns the offset function using data-efficient Gaussian Process regression [9]. The approach we present can be considered a generalization of these approaches. In our approach, this offset function is one of many modules available to the practitioner to plug in to the learned dynamics model in lieu of an analytic representation obtained from prior knowledge. Furthermore, while the aforementioned approaches [8], [9], focus on allowing black-box functions to model non-conservative forces, we show how we can use neural networks to also model a system's Lagrangian, which captures passive dynamics.

Another closely related work developed concurrently and independently to this work is that of Lutter, et al. [14]. Akin to our work, the authors encode the physical prior of Lagrangian mechanics into the model architecture. However, there are key distinctions between the approaches that allow our work to serve a greater scope than that presented by Lutter, et al. [14]. Firstly, they propose to directly model the conservative forces acting on the system by using a neural network. In contrast, we choose to model the system's potential energy and derive the conservative forces from it, thereby guaranteeing that the forces are conservative. Further, by explicitly modeling the potential energy, we are able to use variational integrators that are appropriate for contact-rich simulation. Secondly, Lutter, et al. [14] make the limiting assumption that all generalized torques are directly measurable (which requires precise knowledge of the control input Jacobian) as opposed to learned functions

of the system's state and input. We instead explicitly model generalized forces in a manner flexible enough to encode prior knowledge regarding their structure, but general enough to capture complex phenomena such as dissipative friction. These key distinctions make our methodology more generally applicable to a wider range of robotic settings.

Finally, there exists an approach for explicitly learning the topology of a mechanical manipulator [15]. The work ignores generalized forces, assuming that the generalized forces applied to the system are directly measurable. Furthermore, it requires learning in a higher-dimensional *maximal coordinate* space, while our approach directly learns on a lower dimensional constraint manifold by using *minimal coordinates*.

III. Background

Our work draws inspiration from a variety of fields including classical mechanics, system identification, model-based control, and modern machine learning techniques such as deep learning:

A. Lagrangian Dynamics

The purpose of modeling a dynamic system is to be able to predict how the state of the system evolves over time. Formally, the state of a system is described using *generalized coordinates* $q \in \mathbb{R}^N$ and velocities $\dot{q} \in \mathbb{R}^N$, where N is the number of coordinates. For any mechanical system, the *kinetic energy* can be written as:

$$T(q, \dot{q}) = \frac{1}{2} \dot{q}^T \mathbf{M}(q) \dot{q} \tag{2}$$

where $\mathbf{M}(q)$ is positive definite and called the *mass matrix* of the system. Furthermore, the *potential energy* of the system can be defined as a scalar function $V(q)$. Together, these energies specify the *Lagrangian* of a rigid body system as:

$$\mathcal{L}(q, \dot{q}) = T(q, \dot{q}) - V(q) \tag{3}$$

With knowledge of a system's Lagrangian, the dynamics of the system are specified by the Euler-Lagrange (EL) equation:

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}}\right) - \frac{\partial \mathcal{L}}{\partial q} = F(q, \dot{q}, u) \tag{4}$$

where $F(q, \dot{q}, u)$ represents the *generalized forces* that act on the system, and $u \in \mathbb{R}^M$ are the actuator inputs. In the case of a control-affine system, for example, we would set $F(q, \dot{q}, u) = \mathbf{B}(q) \cdot u$, where $\mathbf{B}(q) \in \mathbb{R}^{N \times M}$. While the EL equation implicitly specifies the system's continuous time dynamics, the explicit form of the dynamics can be found by combining Equations (3) and (4). By applying the chain-rule, we get what is commonly referred to as the *manipulator equation* [16]:

$$\mathbf{M}(q)\ddot{q} + \mathbf{C}(q, \dot{q})\dot{q} + G(q) = F(q, \dot{q}, u) \tag{5}$$

Here, $G(q) = -\nabla_q V(q)$ are the conservative forces that act on the system, and

$$\mathbf{C}_{ij}(q, \dot{q}) = \frac{1}{2} \sum_{k=1}^{N} \left(\frac{\partial \mathbf{M}_{ij}}{\partial q_k} + \frac{\partial \mathbf{M}_{ik}}{\partial q_j} - \frac{\partial \mathbf{M}_{jk}}{\partial q_i} \right) \dot{q}_k \tag{6}$$

is the *Coriolis matrix* of the system [16]. Hence, if we have a function predicting the mass matrix $\mathbf{M}(q)$, the potential $V(q)$, and the forces $F(q, \dot{q}, u)$ that act on the system, we have fully specified the continuous time dynamics of the system.[1]

B. Prediction

Suppose we are given $\mathbf{M}(q)$, $V(q)$, and $F(q, \dot{q}, u)$, as well as initial conditions $q_t, \dot{q}_t$ and actuator input u_t, and want to predict $q_{t'}, \dot{q}_{t'}$ for some $t' = t + \Delta t$. To make this prediction, we need to simulate the model forward in time, which requires integrating Equation (5). We first form an explicit equation for the generalized acceleration:

$$\ddot{q} = \mathbf{M}^{-1}(q) \left[F(q, \dot{q}, u) - \mathbf{C}(q, \dot{q})\dot{q} - G(q) \right] \tag{7}$$

Next, we define the generalized state at time t as $x_t = [q_t, \dot{q}_t]$. We then find $x_{t'}$ using the Runge-Kutta fourth order (RK4) integration scheme [17]. The details of how we implement RK4 can be found in Appendix II.[2]

Though RK4 is considered the standard approach to fixed time-step explicit integration, there are other integration schemes. For example, if we have access to the Lagrangian of a system, we can use an implicit integration scheme called *variational integration*, which better handles non-smooth dynamics [18]. Although we do not consider non-smooth dynamics in this work, access to these techniques is a principal benefit of explicitly modeling a system's Lagrangian.

C. System Identification

In classical system identification, Equation (5) is typically re-written in the linear least squares form as in Equation (1):

$$\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \| H_i \theta - \tau_i \|^2 \tag{8}$$

For smaller datasets, this can be solved to global optimality in closed form. For larger datasets, we can take a gradient-based approach to minimize the same loss in Equation (8).

Alternatively, if parameters cannot be factored into the linear representation shown in Equation (8), then it is common to minimize error in generalized accelerations predicted by the model [8], [19] parameterized by θ:

$$\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \| \ddot{q}_i - \hat{\ddot{q}}_i(\theta) \|^2 \tag{9}$$

where $\hat{\ddot{q}}$ is found according to Equation (7). Here, θ can correspond to the parameters of some function approximator such as a neural network.

D. Model-Based Control

Given a model specifying a system's dynamics, many techniques from control theory can force a system to follow a specified trajectory, or to track some set point [20]. Since we limit the scope of experiments in this work to smooth dynamical systems, we use Direct Collocation Trajectory

[1] See Appendix I[2] for a more efficient method for computing $\mathbf{C}(q, \dot{q})\dot{q}$.
[2] See `http://rejuvyesh.com/publications/mechamodlearn.pdf`

Optimization (DIRCOL) [21] in order to generate a 'nominal trajectory' $(\bar{x}_{1:t_H}, \bar{u}_{1:t_H})$ that:

1) Transports the system from some initial condition to some desired set point,
2) Is dynamically feasible according to the model, and
3) Minimizes some cost function defined over the trajectory, which is typically quadratic in the control effort and tracking error.

To track the nominal trajectory in real-time, we use a Time-Varying Linear Quadratic Regulator (TVLQR) [22]. Here, the system dynamics are linearized along the nominal trajectory from DIRCOL, and feedback gains K_t are found using dynamic programming to minimize a quadratic cost function. The actuator input to the system at some time t is:

$$u_t = \pi(x_t) = \bar{u}_t - K_t(x_t - \bar{x}_t) \tag{10}$$

where $\pi : \mathbb{R}^N \to \mathbb{R}^M$ is referred to as the *synthesized policy*.

E. Model-Based Reinforcement Learning

Assume we have a finite-horizon Markov decision process $\mathcal{M} = \{\mathcal{X}, \mathcal{U}, \mathcal{T}, R, H\}$, where $\mathcal{X}$ is the state-space, $\mathcal{U}$ is the action-space, $\mathcal{T} : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$ is the system dynamics-model, $R : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ is the reward function, and H is the horizon-length. In our context, $\mathcal{X} \in \mathbb{R}^{2N}$ is the space of all generalized coordinates and velocities, and $\mathcal{U} \in \mathbb{R}^M$ is the space of all actuator inputs. Additionally, we assume that while $\mathcal{T}(x, u)$ is not known *a priori*, the reward function $R(x, u)$ is known.

As discussed in Section III-D, if $\mathcal{T}(x, u)$ were known, we could use DIRCOL to synthesize a trajectory that optimally solves our task, and a *policy* $\pi : \mathcal{X} \to \mathcal{U}$ that tracks this trajectory using TVLQR. If $\mathcal{T}(x, u)$ is not known, it can be learned from data observed while interacting with the environment.

Figure 2 shows the architecture for model-based reinforcement learning used in this work. Polydoros et al.provide a survey of a variety of other approaches [23]. Initially, an episode of training data is generated by randomly actuating the system and observing its state-transitions. A model, $\hat{\mathcal{T}}(x, u)$ is learned from the dataset. The model is then passed through DIRCOL and TVLQR to generate a policy $\pi(x)$ that, if the model were accurate, would solve the task. Naturally, the model being imperfect, this would not solve the task, but visit novel states.

By repeating this process, novel data is added to the dataset, allowing the learned model to approach the true model in accuracy. Once its prediction accuracy is close enough to the true model, the synthesized policy $\pi(x)$ will be able to solve the task on the real system. However, it is possible for this approach to fail due to a lack of *exploration*. That is, if we attempt to *greedily* solve the task with our model at every episode, the policies being followed may never visit states that are required in the dataset in order to learn a sufficiently accurate model.

A simple approach to exploration in this setup is to add *exploration noise* that adds variance to the policies generated by DIRCOL+TVLQR. Here, we add random perturbations to the nominal trajectory synthesized by DIRCOL before passing it to TVLQR. By doing so, the system is encouraged to visit states in the neighborhood of what is otherwise believed to be an optimal trajectory.

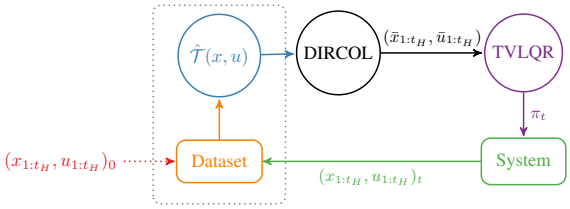

Fig. 2: Framework for Model-Based Reinforcement Learning

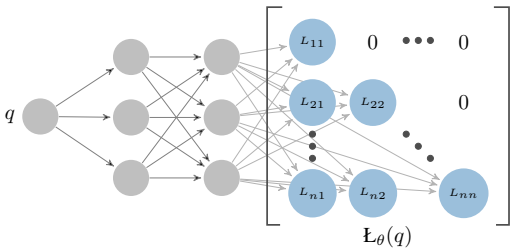

Fig. 3: A neural network parameterization for the Cholesky factor of the model's inertia matrix.

IV. Methodology

In its most general form, we model a physical system by modeling:

1) Its Lagrangian, and
2) The generalized forces that act on it.

We first present a methodology for modeling the positive-definite mass-matrix $\mathbf{M}_\theta(q)$ using a neural network with parameters contained in θ. As shown in Figure 3, we predict $\mathbf{M}_\theta(q) \succ 0$ by predicting the $\frac{N^2+N}{2}$ elements of its Cholesky factor, which is a lower-triangular matrix $\text{Ł}_\theta(q)$, as is done by [14], [24].[3] Here, the neural network first outputs a vector, of which the first N elements are used as the diagonal of $\text{Ł}_\theta(q)$, and the remaining $\frac{N^2-N}{2}$ are used for the off-diagonal elements of the lower half of the matrix. We additionally add a constant offset to the diagonals of $\text{Ł}_\theta(q)$ so that $\mathbf{M}_\theta(q)$ is diagonally dominant and easily invertible given random initializations of θ. We then predict:

$$\mathbf{M}_\theta(q) = \text{Ł}_\theta(q)\text{Ł}_\theta^\top(q) \tag{11}$$

Additionally, we predict the potential energy, $V_\theta(q)$, using a neural network that maps $q \in \mathbb{R}^N \to \mathbb{R}$, and in the most general case, the generalized forces as a function $F_\theta(q, \dot{q}, u)$ that maps $\mathbb{R}^{2N+M} \to \mathbb{R}^N$. Since we are required to take gradients of $\mathbf{M}_\theta(q)$ as well as $V_\theta(q)$, with restpect to q, in order to compute the system's acceleration, and occasionally Hessians of the Lagrangian in order to measure properties of the model such as local generalized stiffness, we require the non-linearities in the neural network to be at least twice-differentiable. In this work, we use the hyperbolic tangent function (tanh) as the non-linearity.

[3] One can optionally enforce that the diagonal elements of $\text{Ł}_\theta(q)$ are positive in order to make the matrix the *unique* Cholesky factor of $\mathbf{M}_\theta(q)$, though we have empirically found this to be unnecessary and only makes model parameters' optimization more difficult.

We have thus far presented the parameterization for the most generic form of a mechanical system. If any prior knowledge is available to a practitioner, they may substitute $\mathbf{M}_\theta(q)$, $V_\theta(q)$, or $F_\theta(q, \dot{q}, u)$ with a more restricted function class than a neural network, so long as as $\mathbf{M}_\theta(q)$ and $V_\theta(q)$ remain twice-differentiable. For example, one may wish to model the system as control-affine with viscous joint-damping. This can be achieved by substituting:

$$F_\theta(q, \dot{q}, u) = \mathbf{B}_\theta(q) \cdot u + \eta_\theta(q) \circ \dot{q} \tag{12}$$

Here, $\mathbf{B}_\theta(q) \in \mathbb{R}^{N \times M}$ and $\eta_\theta(q) \in \mathbb{R}^N$. Though this parameterization is still fairly expressive, it is possible that there are phenomena in the data that it is unable to accurately model.

If $\mathbf{M}_\theta(q)$, $V(q)$ and $F(q, \dot{q}, u)$ are all substituted with analytic models derived from prior knowledge, then this model reduces to a white-box model. On the other hand, if left in its most general form, then the model is still as expressive as a black-box model. This is because $\mathbf{M}_\theta(q)$ is capable of expressing a function always equal to the identity matrix, and $V(q)$ is capable of expressing a function always equal to zero. If this were the case, then it can easily be seen from Equation (7) that $F_\theta(q, \dot{q}, u)$ reduces to the black-box neural network model show in Figure 1. However, we will show empirically that despite having such expressive power, we still gain a reduction in model variance over a naive black-box model.

A. Parameter Optimization

Having described the parameterization for a model of an arbitrary mechanical system, we now describe how we optimize these parameters to learn a model of a system from data. Given some dataset of a system's state-transitions:

$$\mathcal{D} = \{q_k, \dot{q}_k, u_k, q'_k, \dot{q}'_k \mid k \in \{1, \dots N\}\} \tag{13}$$

We optimize the parameters θ by minimizing the *prediction loss*:

$$L(\theta) = \frac{1}{N} \sum_{k=1}^{N} \|q'_k - \hat{q}'_k(\theta)\|^2 + \lambda \|\dot{q}'_k - \hat{\dot{q}}'_k(\theta)\|^2 \tag{14}$$

where $\hat{q}'_k, \hat{\dot{q}}'_k$ are predicted using RK4 applied to the model dynamics with inputs $q_k, \dot{q}_k, u_k$ and parameters θ. We then minimize this loss using gradient descent methods such as Adam [25]. We use the loss presented in Equation (14) as opposed to that presented in Equation (9) for two reasons. Firstly, accelerations $\ddot{q}_k$ are often not directly measured, and rather estimated by finite-difference approximation in practice, which amplifies high-frequency noise present in the data samples. By opting for the loss we presented, we do not need to estimate $\ddot{q}$ as the regression target. Secondly, we have observed that minimizing prediction error is a better metric to optimize a model against than aligning accelerations predicted by the model with those estimated in the data. The down-side of our approach, however, is that differentiating through RK4 in order to compute parameter gradients is a constant-factor more expensive than computing gradients for the loss presented in Equation (9).

V. Experiments

In our experiments, we aim to justify the following properties of our method:

- Ability to learn the dynamics of a complex mechanical system from data,
- Seamless incorporation of prior knowledge to reduce model variance,
- Better data-efficiency for model learning than a naive, black-box approach even in the absence of prior knowledge, and,
- More data-efficient model-based reinforcement learning.

We do so by performing three experiments. In the first, we vary the amount of prior knowledge used when modeling the system and analyze the effect on data required for the learned model to generalize accurately, i.e. analyze the reduction on model variance gained by using prior knowledge. In the second experiment, we compare the ability of a naive approach with our structured black-box approach to make accurate long-horizon predictions. In the third experiment, we perform model-based reinforcement-learning and compare our method with the use of naive-black box model to parameterize the modeled dynamics.

A. Experimental Domain

We first focus on the fully actuated double pendulum (shown in Figure 4) as our system of interest, which has highly non-linear dynamics that are chaotic when unforced. As shown in the figure, the system is specified by the masses m_1 and m_2 and lengths l_1 and l_2 of the rods, the gravitational acceleration g, and coefficients that specify the joint torques τ_1 and τ_2, which will be introduced shortly. The system has two generalized coordinates, $q_1 \in [-\pi, \pi)$ and $q_2 \in [-\pi, \pi)$, which correspond to the relative deflection of each rod in radians, as well as control inputs u_1 and u_2. The dynamics of the system are computed by specifying the mass-matrix, $\mathbf{M}_{sys}(q)$, the potential energy, $V_{sys}(q)$, and the generalized coordinates. The specifications for $\mathbf{M}_{sys}(q)$ and $V_{sys}(q)$ can be found in Appendix III. The generalized forces acting on the system are composed of a control torque, as well as viscous joint-damping:

$$F_{sys}(q, \dot{q}, u) = \begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix} = \begin{bmatrix} b_1 & 0 \\ 0 & b_2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} + \begin{bmatrix} \eta_1 \\ \eta_2 \end{bmatrix} \circ \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \end{bmatrix} \tag{15}$$

Here, b_1 and b_2 are the coefficients of the control-matrix $\mathbf{B}_{sys}(q)$, and η_{sys} specify the linear joint-damping coefficients. Hence, the nine white-box free parameters θ_{wb} of this model are:

$$\theta_{wb} = [m_1, m_2, l_1, l_2, g, b_1, b_2, \eta_1, \eta_2] \tag{16}$$

B. Data-efficiency and Prior Knowledge

In this experiment, we aim to justify the claim that prior knowledge in the form of analytic specifications for $\mathbf{M}(q)$, $V(q)$, $\mathbf{B}(q)$, or η can be easily incorporated into the model parameterization and thereby improve the data-efficiency of learning. We compare with a naive black-box approach in which $\ddot{q} = NN(q, \dot{q}, u; \theta)$, where $NN(\cdot; \theta) : \mathbb{R}^{2N+M} \to \mathbb{R}^N$ is a feedforward neural network.

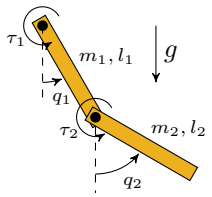

Fig. 4: The actuated double pendulum system.

Name	$\mathbf{M}$ prm	V prm	F prm
MVF	$\mathbf{M}_\theta(q)$	$V_\theta(q)$	$F_\theta(q, \dot{q}, u)$
MVB	$\mathbf{M}_\theta(q)$	$V_\theta(q)$	$\mathbf{B}_\theta(q)u + \eta_{wb} \circ \dot{q}$
MV	$\mathbf{M}_\theta(q)$	$V_\theta(q)$	$\mathbf{B}_{wb}u + \eta_{wb} \circ \dot{q}$
VB	$\mathbf{M}_{wb}(q)$	$V_\theta(q)$	$\mathbf{B}_\theta(q)u + \eta_{wb} \circ \dot{q}$
MB	$\mathbf{M}_\theta(q)$	$V_{wb}(q)$	$\mathbf{B}_\theta(q)u + \eta_{wb} \circ \dot{q}$
M	$\mathbf{M}_\theta(q)$	$V_{wb}(q)$	$\mathbf{B}_{wb}u + \eta_{wb} \circ \dot{q}$
V	$\mathbf{M}_{wb}(q)$	$V_\theta(q)$	$\mathbf{B}_{wb}u + \eta_{wb} \circ \dot{q}$
B	$\mathbf{M}_{wb}(q)$	$V_{wb}(q)$	$\mathbf{B}_\theta(q)u + \eta_{wb} \circ \dot{q}$
F	$\mathbf{M}_{wb}(q)$	$V_{wb}(q)$	$F_\theta(q, \dot{q}, u)$
W-B	$\mathbf{M}_{wb}(q)$	$V_{wb}(q)$	$\mathbf{B}_{wb}u + \eta_{wb} \circ \dot{q}$

TABLE I: Various models incorporating varying degrees of prior knowledge. 'prm' is used as shorthand for 'parameterization'.

The model in which $\mathbf{M}_\theta(q)$, $V_\theta(q)$, $F_\theta(q, \dot{q}, u)$ are all represented by neural networks in the manner introduced in Section IV is as expressive as the naive model. Here, no prior knowledge is encoded aside from the assumption that the system conforms to Lagrangian dynamics. In addition to comparing these two models, we compare models where some components are represented by neural networks, corresponding to a lack of prior knowledge, while others are represented by the same analytic functions as that of the true system, except that the parameters of those functions are learned from data. Table I lists the models compared. Components with the subscript θ are represented in neural networks, and those with the subscript wb are ones represented by analytic functions. Specifically, the trainable parameters in $\mathbf{M}_{wb}(q), V_{wb}(q), \mathbf{B}_{wb}$ and η_{wb} are $\{\hat{m}_1, \hat{m}_2, \hat{l}_1, \hat{l}_2\}$, $\{\hat{m}_1, \hat{m}_2, \hat{l}_1, \hat{l}_2, \hat{g}\}$, $\{\hat{b}_1, \hat{b}_2\}$, and $\{\hat{\eta}_1, \hat{\eta}_2\}$, respectively.

Since the white-box parameterization, referred to as 'W-B', is exactly that of the true system, we would expect it to be able to perfectly represent the dynamics of the true system with only a handful of data points. On the other hand, since the naive approach incorporates no prior knowledge whatsoever, we would expect it to require the largest amount of training data to avoid model variance and accurately generalize learned dynamics to unseen states. Additionally, we would expect all other compared modules to require less data the more prior knowledge they incorporate.

To test this hypothesis, we sample a dataset $\{q, \dot{q}, u, q', \dot{q}'\} \in \mathcal{D}_{\text{train}}$ and $\{q, \dot{q}, u, q', \dot{q}'\} \in \mathcal{D}_{\text{val}}$ where $q \sim \mathcal{U}(-\pi, \pi)$ [rad], $\dot{q} \sim \mathcal{U}(-10, 10)$ [rad s^{-1}], $u \sim \mathcal{N}(0, 120^2)$, and $(q', \dot{q}')$ are found by passing the system dynamics and the sampled state and input through RK4 with a time-step of $\Delta t = 0.01$ s.

We first estimate the amount of training data, i.e. $|\mathcal{D}_{\text{train}}|$, needed in order for the naive model to make accurate predictions on $\mathcal{D}_{\text{val}}$. In order to do so, we begin with $|\mathcal{D}_{\text{train}}| = 8$, doubling $|\mathcal{D}_{\text{train}}|$ until we find the loss on $\mathcal{D}_{\text{train}}$ after training to completion is similar to the loss on $\mathcal{D}_{\text{val}}$, implying that the naive model is generalizing well. This procedure gives a range, $\{|\mathcal{D}_{\text{train}}|_{\text{min}}, |\mathcal{D}_{\text{train}}|_{\text{max}}\} = \{2^{12}, 2^{13}\}$, which contains the minimum $|\mathcal{D}_{\text{train}}|$ required for the naive model to generalize. When given a training dataset of 2^{13} samples, the naive model achieves a validation loss according to Equation (14) of $10^{-2.5}$.

Starting with $|\mathcal{D}_{\text{train}}| = 8$, we then train each of the models in Table I, checking if a training dataset of that size is sufficient for the model to achieve a validation loss of $10^{-2.5}$. If it is not, we double the dataset size and repeat the process.

This procedure ultimately gives us a range, for each model, specifying the minimum amount of data required in order for that model to achieve the stated validation loss. Figure 5 shows the range found for each model in Table I. The experiment was run for 5 different seeds that vary the initializations of trainable parameters, as well as the samples making up the train and validation sets. We find the results to be consistent across seeds.

Examining Figure 5 from left to right, we see that as we reduce the number of components parameterized with prior knowledge, the amount of data required for the model to generalize well, i.e. for the model to avoid suffering from model variance, increases. We note that the white-box parameterization, which has 9 trainable parameters, takes between 8 and 16 data points to train, matching expectations. Further, we note that the MVF parameterization, which incorporates no prior knowledge except that the system conforms to Lagrangian dynamics, requires on the order of half as much data to generalize as well as the naive approach does. The B parameterization requires on the order of 16 times less data to generalize compared to the F parameterization. This is not particularly surprising as the fully generally parameterization of the forcing function used in F is far more expressive than the control-affine parameterization used in B. We also observe that having no prior knowledge regarding the potential energy, i.e. letting $V(q)$ be represented by a neural network, increases the data required to generalize, suggesting that one may want to consider less expressive models for it.

In summary, this experiment shows that the methodology we present allows us to learn dynamics of a complex mechanical system from data, and seamlessly incorporate available prior knowledge by substituting neural network parameterizations for the various components with less expressive parameterizations. Additionally, we show that even in the absence of prior knowledge, i.e. the MVF parameterization, using our approach enables us to generalize effectively from less data than a naive black-box approach.

C. Multi-Step Prediction on the Double Pendulum

In this experiment, we compare the ability of the Naive and MVF models to perform multi-step predictions on the double-pendulum domain. Both models are trained on the same dataset of 4,096 samples. Figure 6 shows that the MVF model is able to accurately replicate the behavior of the true

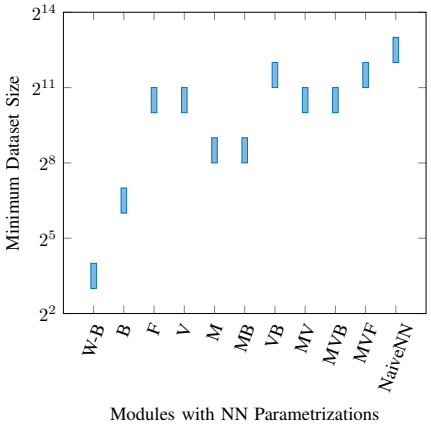

Fig. 5: Minimum amount of training data required for various models to generalize well across the Actuated Double Pendulum state-space.

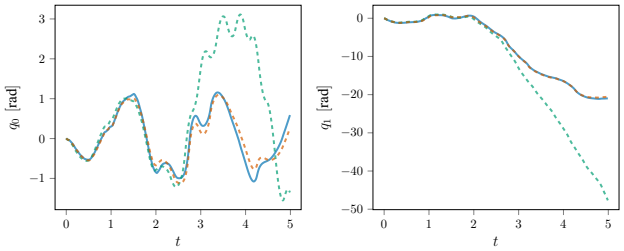

Fig. 6: MVF (orange) vs Naive (green) 5s predictions with the same initial conditions and inputs as the true trajectory (blue).

system for the entirety of the 5 second simulation, while the naive model quickly diverges from ground truth. This result is consistent with the conclusion that the MVF model more accurately learns the system's dynamics from limited data when compared to the naive model. Experiments thus far have trained the models on i.i.d data sampled from an *a priori* specified distribution over the input-space. In the next experiment, we demonstrate the ability to efficiently learn dynamics in a reinforcement learning setting, in which data is presented in the form of trajectories sampled from the system, and thereby not i.i.d.

D. Model-based reinforcement learning

In this experiment, we compare the naive model with the MVF and MVB models from Table I in their abilities to learn the dynamics of the actuated double pendulum for the purpose of solving a swing up task. The task is to actuate the system in a manner that brings it to an state that is upright and stationary, i.e. $(q_1, q_2) = (\pi, 0.0)$ [rad] and $(\dot{q}_1, \dot{q}_2) = (0.0, 0.0)$ [rad s^{-1}], in 2.06 seconds, and then remain stable in this configuration for an additional 0.5 seconds. In order to do so, the model must learn from the available data well enough to accurately generalize predicted dynamics to the relevant parts of the state space. If the model is inaccurate, we expect DIRCOL to produce nominal trajectories that are less dynamically feasible, and TVLQR to produce less accurate linearizations of the dynamics about

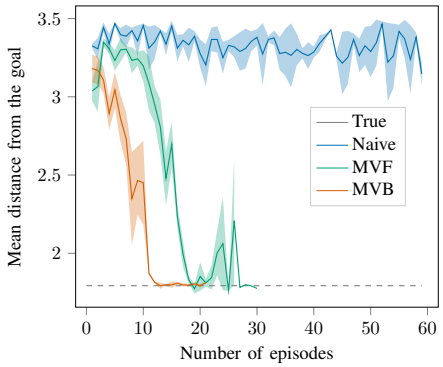

Fig. 7: Model-based reinforcement learning performance of various model parameterizations. Here, performance is the mean Euclidean distance between the end-effector and the target over the trajectory, where a lower score is better.

those trajectories. These two effects will consequently cause the algorithm to require many more interactions with the environment in order to solve the task, when using that model to plan a trajectory. Hence, we expect to see that the naive model requires the largest number of interactions to solve the task, followed by MVF, and then MVB.

We follow the procedure for model-based reinforcement learning described in Section III-E. All trajectories start from the downward equilibrium $(q_1, q_2) = (0.0, 0.0)$ and $(\dot{q}_1, \dot{q}_2) = (0.0, 0.0)$. We provide all models the same initial trajectory found by randomly actuating the system with actions $u_t \sim \mathcal{N}(0, 120^2)$. All trajectories are simulated with $\Delta t = 0.1$ s, and where the action u_t applied is clipped to have a maximum absolute value of 120. We then train all models on this dataset for 5000 epochs, using the Adam optimizer with learning rate of 3×10^{-4}. Every time new data is added to a model's dataset, the model is trained for an additional 1000 epochs.

Next, we perform trajectory optimization using the trained models to get nominal trajectories and a TVLQR tracking controller to synthesize a policy for each model. However, in order to encourage exploration, we add noise sampled from $\mathcal{N}(0, 0.25^2)$ to the nominal trajectories independently to each $q_{1,t}, q_{2,t}, \dot{q}_{1,t}, \dot{q}_{2,t}$ and u_t. Each model's policy is then rolled out on the system, and the trajectory generated is added to each model's dataset. This process is repeated until the trajectories followed by using a given model are reliably solving the task. We repeat this test using three different random seeds.

We measure the performance of the system by measuring the mean Euclidean distance between the end effector and its desired set-point (vertical and stable), over the trajectory. A well-performing model will have a low score using this metric. When evaluating performance, we do not add noise to the nominal trajectories.

Figure 7 shows the results of this experiment. As we can see, the results match our expectations: the MVB parameterization, which incorporates some prior knowledge about the structure of the generalized forces, solves the task with the fewest number of interactions with the environment. Additionally, the MVF parameterization, which incorporates

no prior knowledge, takes more interactions to solve the task, but does so eventually. The naive black-box parameterization does not succeed in solving the task in the allowed time-frame. Observing the nominal trajectories and the policies' abilities to follow the trajectories on the real system, we see the behavior we expect for MVF and MVB.[4] The nominal trajectories eventually start looking more realistic, and the policies start doing a better job of following them. However, when using the naive parameterization, DIRCOL appears to be unable to find a solution that transports the system to the desired goal state. Consequently, the trajectories added to the dataset never explore the relevant parts of state-space, resulting in a model that is never able to generalize well enough to solve the task.

Seeing that even the MVF model, which incorporates no prior knowledge, is able to solve the task in a reasonable amount of time, as well as the fact that prior knowledge improves sample efficiency, validates the hypothesis that using the method we presented enables more efficient model-based reinforcement learning than if a naive black-box parameterization of the dynamics were used.

VI. Conclusions

In this work we presented a method for parameterizing an arbitrary mechanical system using neural networks, as well as a method for training such models from data. Unlike naive black-box approaches that predict accelerations directly, we use neural networks to parameterize the Lagrangian of a system and the generalized forces that act on it. We showed that such a modular parameterization is flexible enough to allow us to seamlessly incorporate prior knowledge where it is available, allowing a practitioner to precisely balance model bias and variance. We showed on a simulated actuated double pendulum that, even in the absence of prior knowledge, our method learns the dynamics of complex mechanical systems more efficiently than a naive, black-box approach.

There are many interesting applications of this work that we intend to explore in the near future. Firstly, by explicitly modeling the system's Lagrangian, we have the ability to use variational integrators to accurately simulate non-smooth dynamics. By interfacing optimal control strategies with such simulators, we can perform model-based reinforcement learning in contact-rich environments. Secondly, by parameterizing the space of mechanical systems, we can take scalable probabilistic approaches to system-identification. This allows us to explore possibilities for robust control and safe, efficient model-based reinforcement learning. Lastly, a limitation of our current framework is that it is more computationally expensive to make predictions with our model than with a black or white-box model. We intend to explore optimizations of our method that can close this performance gap.

Acknowledgments

We are thankful to Jeannette Bohg for advice. This work is supported in part by DARPA under agreement number D17AP00032. The content is solely the responsibility of the authors and does not necessarily represent the official views of DARPA.

[4]Videos of the training episodes can be found at `https://youtu.be/NxHVLlNj6hg`.

References

[1] C. C. de Wit, B. Siciliano, and G. Bastin, *Theory of robot control*. Springer Science & Business Media, 2012.

[2] P. A. Ioannou and J. Sun, *Robust adaptive control*. Courier Corporation, 2012.

[3] C. H. An, C. G. Atkeson, and J. M. Hollerbach, "Estimation of inertial parameters of rigid body links of manipulators," in *IEEE Conference on Decision and Control*, 1985, pp. 990–995.

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

[5] P. J. Werbos, T. McAvoy, and T. Su, "Neural networks, system identification, and control in the chemical process industries," in *Handbook of Intelligent Control Neural, Fuzzy, and Adaptive Approaches*, D. A. White and D. A. Sorge, Eds. New York: Van Nostrand Reinhold, 1992, ch. 10.

[6] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Multistep neural networks for data-driven discovery of nonlinear dynamical systems," *arXiv preprint arXiv:1801.01236*, 2018.

[7] S. Chen, S. Billings, and P. Grant, "Non-linear system identification using neural networks," *International Journal of Control*, vol. 51, no. 6, pp. 1191–1214, 1990.

[8] N. Ratliff, F. Meier, D. Kappler, and S. Schaal, "Doomed: Direct online optimization of modeling errors in dynamics," *Big Data*, vol. 4, no. 4, pp. 253–268, 2016.

[9] D. Nguyen-Tuong and J. Peters, "Using model knowledge for learning inverse dynamics," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2010, pp. 2677–2682.

[10] S. Tu and B. Recht, "The gap between model-based and model-free methods on the linear quadratic regulator: An asymptotic viewpoint," *arXiv preprint arXiv:1812.03565*, 2018.

[11] R. Laroche and M. Barlier, "Transfer reinforcement learning with shared dynamics," in *AAAI Conference on Artificial Intelligence (AAAI)*, 2017.

[12] A. Zhang, H. Satija, and J. Pineau, "Decoupling dynamics and reward for transfer learning," *arXiv preprint arXiv:1804.10689*, 2018.

[13] M. Deisenroth and C. E. Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," in *International Conference on Machine Learning (ICML)*, 2011, pp. 465–472.

[14] M. Lutter, C. Ritter, and J. Peters, "Deep Lagrangian networks: Using physics as model prior for deep learning," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=BklHpjCqKm

[15] F. D. Ledezma and S. Haddadin, "First-order-principles-based constructive network topologies: An application to robot inverse dynamics," in *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*. IEEE, 2017, pp. 438–445.

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

[17] C. Runge, "Ueber die numerische auflösung von differentialgleichungen," *Mathematische Annalen*, vol. 46, no. 2, pp. 167–178, Jun 1895.

[18] Z. Manchester and S. Kuindersma, "Variational contact-implicit trajectory optimization," in *International Symposium on Robotics Research (ISRR), Puerto Varas, Chile*, 2017.

[19] F. Meier, D. Kappler, N. Ratliff, and S. Schaal, "Towards robust online inverse dynamics learning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 4034–4039.

[20] D. Liberzon, *Calculus of Variations and Optimal Control Theory: A Concise Introduction*. Princeton University Press, Jan 2012.

[21] M. Kelly, "An introduction to trajectory optimization: How to do your own direct collocation," *SIAM Review*, vol. 59, no. 4, pp. 849–904, 2017.

[22] B. D. O. Anderson, *Optimal Control: Linear Quadratic Methods (Dover Books on Engineering)*. Dover Publications, 2007.

[23] A. S. Polydoros and L. Nalpantidis, "Survey of model-based reinforcement learning: Applications on robotics," *Journal of Intelligent & Robotic Systems*, vol. 86, no. 2, pp. 153–173, May 2017.

[24] T. Haarnoja, A. Ajay, S. Levine, and P. Abbeel, "Backprop KF: Learning discriminative deterministic state estimators," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2016, pp. 4376–4384.

[25] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

APPENDIX I
EFFICIENTLY COMPUTING CORIOLIS FORCES

In this section, we describe a method for computing $\mathbf{C}(q,\dot{q})\dot{q}$ in $\mathcal{O}(N^2)$. We begin by restating Equation (6):

$$\mathbf{C}_{ij}(q,\dot{q}) = \frac{1}{2}\sum_{k=1}^{N}\left(\frac{\partial \mathbf{M}_{ij}}{\partial q_k} + \frac{\partial \mathbf{M}_{ik}}{\partial q_j} - \frac{\partial \mathbf{M}_{kj}}{\partial q_i}\right)\dot{q}_k \tag{17}$$

We can find the the i^{th} element of $C(q,\dot{q})\dot{q}$ as follows:

$$\begin{aligned}\{\mathbf{C}(q,\dot{q})\dot{q}\}_i &= \sum_{j=1}^{N}\left(\frac{1}{2}\sum_{k=1}^{N}\left(\frac{\partial \mathbf{M}_{ij}}{\partial q_k} + \frac{\partial \mathbf{M}_{ik}}{\partial q_j} - \frac{\partial \mathbf{M}_{kj}}{\partial q_i}\right)\dot{q}_k\right)\dot{q}_j \\ &= \underbrace{\frac{1}{2}\sum_{j=1}^{N}\sum_{k=1}^{N}\frac{\partial \mathbf{M}_{ij}}{\partial q_k}\dot{q}_k\dot{q}_j}_{\text{S1}} + \underbrace{\frac{1}{2}\sum_{j=1}^{N}\sum_{k=1}^{N}\frac{\partial \mathbf{M}_{ik}}{\partial q_j}\dot{q}_k\dot{q}_j}_{\text{S2}} - \frac{1}{2}\sum_{j=1}^{N}\sum_{k=1}^{N}\frac{\partial \mathbf{M}_{kj}}{\partial q_i}\dot{q}_k\dot{q}_j\end{aligned} \tag{18}$$

Note the symmetry between S1 and S2. Combining and moving gradients outside the summation, we get:

$$\{\mathbf{C}(q,\dot{q})\dot{q}\}_i = \sum_{j=1}^{N}\frac{\partial}{\partial q_j}\sum_{k=1}^{N}\mathbf{M}_{ik}\dot{q}_k\dot{q}_j - \frac{\partial}{\partial q_i}\sum_{j=1}^{N}\sum_{k=1}^{N}\frac{1}{2}\mathbf{M}_{kj}\dot{q}_k\dot{q}_j \tag{19}$$

Which can be concisely written as:

$$\mathbf{C}(q,\dot{q})\dot{q} = \nabla_q\left(\mathbf{M}(q)\dot{q}\right)\dot{q} - \nabla_q\left(\frac{1}{2}\dot{q}^\top\mathbf{M}(q)\dot{q}\right) \tag{20}$$

The most expensive operation is the computation of the Jacobian matrix $\nabla_q(\mathbf{M}(q)\dot{q})$, which is computed in $\mathcal{O}(N^2)$.

APPENDIX II
PREDICTION WITH RK4

Given a the system's generalized coordinates q_t and generalized velocity $\dot{q}_t$ at some time t, we wish to predict $q_{t'}$ and $\dot{q}_{t'}$ at some time $t' = t + \Delta t$. First, we define $x_t = [q_t, \dot{q}_t]^\top$, and consequently, $\dot{x}_t = [\dot{q}_t, \ddot{q}_t]^\top$. Here, $\ddot{q}_t$ is computed according to Equation (7). To integrate a function $g(x,u)$, RK4 is a standard integration scheme:

$$\begin{aligned}k_1 &= \Delta t \cdot g(x_t, u_t) \\ k_2 &= \Delta t \cdot g(x_t + k_1/2, u_t) \\ k_3 &= \Delta t \cdot g(x_t + k_2/2, u_t) \\ k_4 &= \Delta t \cdot g(x_t + k_3, u_t) \\ x_{t'} &= x_t + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)\end{aligned} \tag{21}$$

Here, we let $g([q,\dot{q}]^\top, u) = [\dot{q}, \ddot{q}(q,\dot{q},u)]^\top$.

APPENDIX III
EXPERIMENTS

Here we specify the double pendulum dynamics in our experiments, as well as parametrizations of the naive black-box model and structured black box models used in our experiments.

A. Double Pendulum

The mass matrix $\mathbf{M}_{sys}(q)$ and potential energy $V_{sys}(q)$ of the double pendulum are analytically specified below.

$$\mathbf{M}_{sys}(q) = \begin{bmatrix} I_{11} & I_{12} \\ I_{12} & I_2 \end{bmatrix} \tag{22}$$

where

$$I_1 = \frac{1}{3}m_1 l_1^2 \tag{23}$$

$$I_2 = \frac{1}{3}m_2 l_2^2 \tag{24}$$

$$I_{11} = I_1 + I_2 + m_2 {l_1}^2 + m_2 l_1 l_2 \cos q_2 \tag{25}$$

$$I_{12} = I_2 + \frac{1}{2}m_2 l_1 l_2 \cos q_2 \tag{26}$$

$$V_{sys}(q) = -\frac{1}{2} m_1 g l_1 \cos q_1 - m_2 g \left(l_1 \cos q_1 + \frac{l_2}{2} \cos(q_1 + q_2) \right) \tag{27}$$

For the experiments, the parameters specified in Table II are used to define the system dynamics.

Parameter	Value	Unit	Parameter	Value	Unit
m_1	10.0	kg	b_1	1.0	$\mathrm{N\,m}$
m_2	10.0	kg	b_2	1.0	$\mathrm{N\,m}$
l_1	1.0	m	η_1	-0.5	$\mathrm{N\,m\,s\,rad^{-1}}$
l_2	1.0	m	η_2	-0.5	$\mathrm{N\,m\,s\,rad^{-1}}$
g	10.0	$\mathrm{m/s^2}$			

TABLE II: Parameters specifying the system dynamics for both experiments.

B. Model Parameterization

The Naive model consists of a multi-layer perceptron with 3 hidden layers of dimension 64. The MVF model consists of a multi-layer perceptron for $\mathbf{M}(q)$, $V(q)$ and $F(q, \dot{q}, u)$ each, with 3 hidden layers of dimension 32. These network sizes are chosen so that the naive and MVF model have roughly the same number of trainable parameters. All other models listed in Table I that use neural networks to parameterize components use neural networks with with 3 hidden layers of dimension 32 for those components.