Automatic Differentiation and Continuous Sensitivity Analysis of Rigid Body Dynamics

David Millard*[1], Eric Heiden*[1], Shubham Agrawal[2], Gaurav S. Sukhatme[1]

***Abstract*— A key ingredient to achieving intelligent behavior is physical understanding that equips robots with the ability to reason about the effects of their actions in a dynamic environment. Several methods have been proposed to learn dynamics models from data that inform model-based control algorithms. While such learning-based approaches can model locally observed behaviors, they fail to generalize to more complex dynamics and under long time horizons.**

In this work, we introduce a differentiable physics simulator for rigid body dynamics. Leveraging various techniques for differential equation integration and gradient calculation, we compare different methods for parameter estimation that allow us to infer the simulation parameters that are relevant to estimation and control of physical systems. In the context of trajectory optimization, we introduce a closed-loop model-predictive control algorithm that infers the simulation parameters through experience while achieving cost-minimizing performance.

I. INTRODUCTION

Physically-based reasoning is fundamental to successfully performing complex tasks in the physical world. This is particularly relevant to the domain of robot learning. There is a large body of mature work in robot dynamics, which need not be learned from scratch per task. In this work, we introduce a differentiable physics simulator for rigid body dynamics. We leverage this differentiability to estimate parameters that result in simulations that closely match the behavior of observed reference systems. Additionally, through trajectory optimization, we can efficiently generate control inputs that minimize cost functions that are expressed with respect to any quantity that is part of the physics computation.

Differentiable physics provides many advantages when used as part of a learning process. Physically accurate simulation obeys dynamical laws of real systems, including conservation of energy and momentum. Furthermore, joint constraints are enforced with no room for error. The parameters of physics engines, like link geometry and inertia matrices, are well-defined and correspond to properties of real systems. Learning these parameters provides a significantly interpretable parameter space, and can benefit classical control and estimation algorithms. These systems provide a high inductive bias, and model parameters need not be retrained for different tasks or reconfigured environments.

Our contributions are as follows:

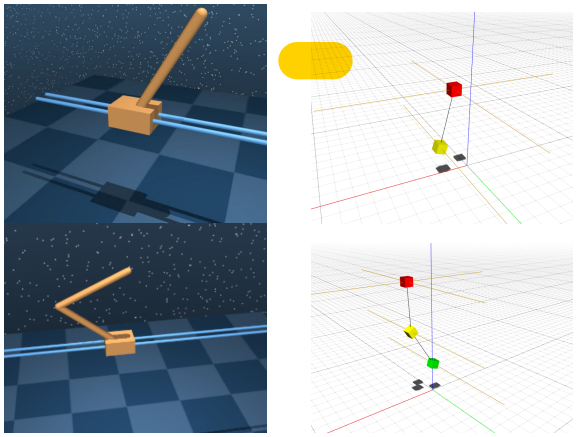

Fig. 1. Visualizations of the simulated cartpole systems. The carts are constrained to the rail, but may move infinitely in either direction. Both systems are actuated by a linear force applied to the cart. *Upper row:* Single cartpole environment from DeepMind Control Suite [1] in the MuJoCo [2] physics simulator (left) and in our simulator (right). *Lower row:* Double cartpole environment from DeepMind Control Suite (left) and in our simulator (right).

1) We present a fully differentiable simulator for rigid body dynamics that supports a rich set of integrators for the accurate simulation of mechanisms over long time horizons.
2) We analyze the performance of gradient calculation methods on the problem of inferring parameters underlying the simulation of rigid-body dynamical systems.
3) We introduce an adaptive model-predictive control algorithm that leverages our differentiable model to perform trajectory optimization while finding the optimal parameters that fit a reference mechanism implemented in another simulator.

II. RELATED WORK

Differentiable physics has recently attracted significant research efforts. Degrave et al. [3] implemented a differentiable physics engine in the automatic differentiation framework Theano. Giftthaler et al. [4] presented a rigid-body-dynamics simulator that allows for the computation of derivatives through code generation via RobCoGen [5]. Similarly, we use Stan Math [6], a C++ library for reverse-mode automatic differentiation to efficiently compute gradients, even in cases where the code branches significantly. Analytical gradients of rigid-body dynamics algorithms have been implemented in the Pinnocchio library [7] to facilitate optimal control and inverse kinematics. While such manually derived gradients can be computed efficiently, they are less general than

* Equal contribution

[1]David Millard, Eric Heiden, Gaurav S. Sukhatme are with the Department of Computer Science, University of Southern California, Los Angeles, USA {dmillard,heiden,gaurav}@usc.edu

[2] Shubham Agrawal is with the Department of Computer Science, Columbia University, New York, USA sa3762@columbia.edu

our approach since they can only be used to optimize for a number of hand-engineered quantities. More recently, Koolen and Deits [8] have implemented rigid-body-dynamics algorithms in the programming language Julia where, among others, libraries for optimization, automatic differentiation, and numerical integration are available. Non-penetrative multi-point contacts between rigid bodies are often simulated by solving a linear complementarity problem (LCP), through which [9] differentiate using the differentiable quadratic program solver OptNet [10]. While our proposed model does not yet incorporate contact dynamics, we are able to demonstrate the scalability of our approach on versatile applications of differentiable physics to common 3D control domains.

Automatic differentiation of the solutions to differential equations is well studied, with applications to pharmacology, meteorology, and many other fields. Recent machine learning work by [11] recasts learning in long short-term memory networks and residual networks as approximations to this problem. Thorough comparisons of methods for computing parameter gradients of ODE solutions are given in [6], [12], [13].

Learning dynamics models has a tradition in the field of robotics and control theory. Early works on forward models [14] and locally weighted regression [15] yielded control algorithms that learn from previous experience. Computing gradients through the solution of differential equations has been further leveraged for system identification [12].

More recently, a variety of novel deep learning architectures have been proposed to learn *intuitive physics* models. Inductive bias has been introduced through graph neural networks [16], [17], [18], particularly interaction networks [19], [20], [21], [22], [23] that are able to learn rigid and soft body dynamics. Vision-based machine learning approaches to predict the future outcomes of the state of the world have been proposed [24], [25], [26], [27], [28]. *Physics-informed learning* imposes a stronger inductive bias on the learning problem to model particular physical processes, such as cosmological structure formation [29] or turbulence models [30]. Deep Lagrangian Networks [31] and Hamiltonian Networks [32] represent functions in the respective classical mechanics frameworks using deep neural networks.

The approach of adapting the simulator to real world dynamics, which we demonstrate through our adaptive MPC algorithm in subsection V-C, has been less explored. While many previous works have shown to adapt simulators to the real world using system identification and state estimation [33], [34], few have shown adaptive model-based control schemes that actively close the feedback loop between the real and the simulated system [35], [36], [37]. Instead of using a simulator, model-based reinforcement learning is a broader field [38], where the system dynamics, and state-action transitions in particular, are learned to achieve higher sample efficiency compared to model-free methods. Within this framework, predominantly Gaussian Processes [39], [40], [41] and neural networks [42], [43] have been proposed to learn dynamics and optimize policies. Bayesian neural networks in particular have been used to learn dynamics in model-based reinforcement learning approaches [44], [45], [46], [47].

III. NOTATION

Throughout this work, we follow the following conventions. $\mathbf{x} \in \mathfrak{X}$ denotes the system's state vector from the state space $\mathfrak{X}$. $\mathbf{u} \in \mathfrak{U}$ denotes the system's control vector from the control space $\mathfrak{U}$. $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ denotes the system's parameter vector from the parameter space $\boldsymbol{\Theta}$. A rigid body system is entirely described by the generalized coordinates[1] $\boldsymbol{\tau}, \mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}$, which denote the generalized forces, positions, velocities, and accelerations, respectively.

IV. APPROACH

A. Rigid Body Dynamics

To simulate the dynamics of a rigid body system, we integrate the Newton-Euler equation

$$\boldsymbol{\tau} = \mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + C(\mathbf{q}, \dot{\mathbf{q}}) + G(\mathbf{q}). \tag{1}$$

$\mathbf{H}(\mathbf{q})$ gives the generalized inertial matrix of the system for configuration $\mathbf{q}$, and $C(\mathbf{q}, \dot{\mathbf{q}})$ describes the centrifugal and Coriolis terms affecting motion. $G(\mathbf{q})$ describes the contribution from gravity.

Given a descriptive model consisting of joints, bodies, and predecessor/successor relationships, we build a kinematic chain that specifies the dynamics of the system. From a mechanism's position vector $\mathbf{q}$, the forward kinematics function $\mathrm{KIN}(\cdot)$ computes the positions and orientation quaternions of the geometries attached to the kinematic chain (such as the end-effector of a robot arm) in world coordinates.

Forward dynamics, computed by $\mathrm{FD}(\cdot)$ is the mapping from positions, velocities and forces to accelerations. We efficiently compute forward dynamics using the Articulated Body Algorithm (ABA) [48], that propagates forces through the bodies while adhering to the motion subspaces defined by the joints that connect them. In our simulator, bodies comprise physical entities with mass, inertia, and attached rendering and collision geometries. Joints describe constraints on the relative motion of bodies in a model. Equipped with such a graph of n bodies connected via joints with forces acting on them, ABA computes the joint accelerations $\ddot{\mathbf{q}}$ in $O(n)$ operations.

B. Integration

Unless specified otherwise, we represent the mechanism's state $\mathbf{x}$ by $[\mathbf{q}, \dot{\mathbf{q}}]$ so that the change in state $\dot{\mathbf{x}}$ corresponds to $[\dot{\mathbf{q}}, \ddot{\mathbf{q}}]$. A mechanism is parameterized by the real vector $\boldsymbol{\theta}$. Such parameters can, depending on the particular system, contain values for the geometries of the links, inertia properties, and other settings that influence the dynamics of the mechanism.

Resulting from the forward dynamics f, a new change in state $\dot{\mathbf{x}}(t_i)$ is computed using ABA at each time step

[1]Generalized coordinates sparsely encode only particular degrees of freedom in the kinematic chain so that connected bodies remain connected.

t_i given the previous state $\mathbf{x}(t_{i-1})$ and parameters $\boldsymbol{\theta}$. Such relationship forms an ordinary differential equation (ODE) $\dot{\mathbf{x}}(t_i) = f(\mathbf{x}(t_{i-1}), t_i, \boldsymbol{\theta})$, which is solved for the next state $\mathbf{x}(t_i)$ through an integrator. We leverage several methods to solve ODEs, from a simple Euler integrator, through explicit stepping schemes like fourth-order Runge-Kutta (RK4), to adaptive stepping algorithms, such as Dormand-Prince (commonly referred to as RK45) method.

In order to simulate a system, the ODE is solved for a sequence of time steps $t_i \in [t_0, \ldots, t_T]$. Throughout this work we consider equidistant time intervals with an integration step size Δt. The smaller the step size, the more accurate the simulation, but the more ODE system evaluations are necessary. Larger time steps improve the execution performance of the simulator but yield decreased accuracy, particularly in chaotic systems.

C. Parameter Estimation

We are interested in the behavior of this ODE system with respect to changes in its parameters, $\boldsymbol{\theta}$, and to its control inputs, $\mathbf{u}$. Parameters of note are the continuous parameters describing the inertia and geometry of links and joint attachments. Discrete parameters, describing the structure of the system, are fundamentally interesting, but are not considered in this method.

Given a cost function $c : \mathfrak{X} \to \mathbb{R}$ evaluated over states evaluated at times t_i, the overall loss $\mathcal{L}$ is defined as follows:

$$\mathcal{L} = \sum_t c(\mathbf{x}(t_i)). \tag{2}$$

Note that the gradient $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}$ only becomes available by integrating over the dynamics f so that the parameters influence the system states. Typically, for parameter estimation, the loss is the distance $||\mathbf{x}(t_i) - \mathbf{x}^*(t_i)||_2^2$ between the simulated states $\mathbf{x}(t_i)$ and the states from a reference trajectory $\mathbf{x}^*(t_i)$, which can be given from a real physical system or another simulation with unknown parameters. This approach is known as an *initial value problem* (IVP) and has an important application in simulation-to-reality (sim2real) transfer learning, where the reality gap between the agent's dynamics model and the real world dynamics needs to be sufficiently small for the agent to operate in the real world (cf. [38]).

D. Analytical Differentiation

In order to estimate parameters, we seek to minimize Equation 2 through gradient-based optimization. Such an approach requires calculating gradients of the parameters $\boldsymbol{\theta}$ with respect to the ODE solution $\mathbf{x}(t_i)$, which is known as Continuous Sensitivity Analysis and has a wide range of applications [12].

Historically, the two primary methods for computing derivatives through complex systems have been numerical and analytical derivation. Analytical (symbolic) derivation gives the user a chance to hand-optimize calculations, but is inflexible and error-prone, as gradients of any new dynamics elements must be determined separately.

E. Numerical Differentiation

A numerical approximation to the analytical gradients can be obtained through finite differences. This approach is an one-at-a-time method (OAT) that, along each parameter dimension $d \in O(|\boldsymbol{\theta}|)$, adapts the parameter vector to approximate the gradient w.r.t. the final system state $\mathbf{x}(t_T)$. A common finite differencing approach is the symmetric difference quotient $\frac{g(y+h)-g(y-h)}{2h}$ that approximates the gradient of function g at point y symmetrically at two nearby points to y using the step size h. Its error is characterized as $O(h^2)$, while higher-order symmetric derivatives can be obtained that achieve higher accuracy at the cost of more function evaluations. In practice, step size h cannot be reduced indefinitely due to floating point errors [49], limiting the overall achievable accuracy of this method.

F. Automatic Differentiation

Additionally, we may compute gradients by using forward or reverse mode automatic differentiation (AD) on the numerical integrator used to solve the ODE. Forward-mode differentiation performs arithmetic on dual numbers to compute functions and their derivatives simultaneously. Reverse-mode differentiation tracks the derivatives of function evaluations, storing them on a tape. Gradients are then computed in reverse by repeatedly applying the chain rule. Tape-based AD software may see high memory usage, especially for solvers with adaptive step sizes or systems with many outputs, while forward-mode differentiation scales poorly with input parameters. We present results for automatically differentiating ODE integrators in Section V.

G. Local Sensitivity Analysis

Local sensitivity analysis, is a method for computing gradients for ODE solutions by augmenting the model dynamics to include the dynamics for the gradient itself (local sensitivities)

$$\frac{d}{dt}\left(\frac{\partial \mathbf{x}}{\partial \boldsymbol{\theta}_i}\right) = \frac{\partial f}{\partial \mathbf{x}}\frac{\partial \mathbf{x}}{\partial \boldsymbol{\theta}_i} + \frac{\partial f}{\partial \boldsymbol{\theta}_i}$$

where $\boldsymbol{\theta}_i$ is the i-th parameter. This approach adds a new equation to the system per parameter, and thus performs poorly for systems with many parameters. Fortunately, for many applications in robotics, we are interested in optimizing a few unknown parameters of a model.

H. Adjoint Sensitivity Analysis

To compute derivatives in our simulator, we use techniques from automatic differentiation and sensitivity analysis. There are multiple ways to compute gradients of functions of the solutions to a system of differential equations. A concise overview is given by [12]. One method, introduced by [50] and popularized recently by [11], is a continuous method called Adjoint Sensitivity Analysis.

We can compute a parameter gradient by backwards solving

$$\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t)^\intercal \frac{\partial f(\mathbf{x}(t), t, \boldsymbol{\theta})}{\partial \mathbf{x}},$$

Algorithm 1 Coupled ODE System

Require:
Mechanism parameters $\boldsymbol{\theta}$
Initial state $\mathbf{x}(t_0)$
Start time t_0, end time t_1

$s_0 \leftarrow [\mathbf{x}(t_0), \mathbf{0}_{|\mathbf{x}|\times|\boldsymbol{\theta}|}]$
function AUGMENT($[\mathbf{x}(t), \frac{\partial \mathbf{x}(t)}{\partial \boldsymbol{\theta}}], t, \boldsymbol{\theta}$)
$A \leftarrow \frac{\partial f}{\partial \mathbf{x}(t)} \frac{\partial \mathbf{x}(t)}{\partial \boldsymbol{\theta}} + \frac{\partial f}{\partial \boldsymbol{\theta}}$
return $[f(\mathbf{x}(t), t, \boldsymbol{\theta}), [A_{11}, \ldots, A_{|\mathbf{x}||\boldsymbol{\theta}|}]]$
end function
$[\mathbf{x}(t_1), \frac{\partial \mathbf{x}(t_1)}{\partial \boldsymbol{\theta}}] \leftarrow$ INTEGRATE(s_0, AUGMENT, $t_0, t_1, \boldsymbol{\theta}$)
return $\frac{\partial \mathbf{x}(t_1)}{\partial \boldsymbol{\theta}}$

which is known as the adjoint problem. At every discrete point t_i where cost is evaluated, the ODE solution is perturbed by $\frac{\partial \mathcal{L}}{\partial \mathbf{x}(t)}$ where $\mathbf{x}(t)$ is solved in the forward pass. Then the loss gradient is

$$\frac{d\mathcal{L}}{d\boldsymbol{\theta}} = \mathbf{a}(t_0)^\intercal \frac{\partial f(\mathbf{x}(t_0), \boldsymbol{\theta}, t_0)}{\partial \mathbf{x}} + \sum_i \int_{t_i}^{t_{i+1}} \mathbf{a}(t)^\intercal \frac{\partial f(\mathbf{x}(t), \boldsymbol{\theta}, t)}{\partial \boldsymbol{\theta}} dt$$

Algorithm 2 Adjoint Sensitivity Method

Require:
Mechanism parameters $\boldsymbol{\theta}$
Final state $\mathbf{x}(t_1)$
Loss gradient of the final state $\mathbf{a}(t_1) = \frac{\partial \mathcal{L}}{\partial \mathbf{x}(t_1)}$
Start time t_0, end time t_1

$s_0 \leftarrow [\mathbf{x}(t_1), \frac{\partial \mathcal{L}}{\partial \mathbf{x}(t_1)}, \mathbf{0}_{|\boldsymbol{\theta}|}]$
function AUGMENT($[\mathbf{x}(t), \mathbf{a}(t), \cdot], t, \boldsymbol{\theta}$)
return $[f(\mathbf{x}(t), t, \boldsymbol{\theta}), -\mathbf{a}(t)^\intercal \frac{\partial f}{\partial \mathbf{x}}, -\mathbf{a}(t)^\intercal \frac{\partial f}{\partial \boldsymbol{\theta}}]$
end function
$[\mathbf{x}(t_0), \frac{\partial \mathcal{L}}{\partial \mathbf{x}(t_0)}, \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}] \leftarrow$ INTEGRATE(s_0, AUGMENT, $t_1, t_0, \boldsymbol{\theta}$)
return $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}$

V. EXPERIMENTS

We evaluate the previously introduced methods for computing gradients through the ODE solutions. To this end, we first benchmark these approaches and subsequently compare them on parameter estimation problems involving a simulated compound pendulum and the automatic design of a robot arm. Next, we present an adaptive control algorithm that leverages the parameter estimation capabilities of our differentiable simulator and combines it with trajectory optimization to control a mechanism in a different simulator. Our simulator is implemented in C++ using the Eigen framework [51] for linear algebra and Stan Math [6] for automatic differentiation (AD). Since the latter only provides an implementation of reverse-mode automatic differentiation, we limit our attention to this algorithm and leave considerations of forward-mode and other AD techniques for future work.

A. Benchmarking Gradient Calculation Approaches

In our first experiment, we consider an n-link compound pendulum that is simulated over a variety of time steps given its link lengths l_k as parameters $\boldsymbol{\theta} = \{l_1, ..., l_n\}$. We focus in our profiling of the approaches introduced in section IV on their computational efficiency, i.e., how many dynamics evaluations $f(\cdot)$ are necessary, how many variables are generated on the automatic differentiation stack, and the total computation time.

We first consider a double pendulum ($n = 2$) and report the performance of the algorithms Numerical Differentiation ("Numerical"), reverse-mode AD ("AutoDiff"), Adjoint Sensitivity Method ("Adjoint") and Coupled ODE System ("Coupled") in Figure 2. Although the number of ODE evaluations (third plot) grows exponentially faster with finite differencing (orange) than the other methods, we note that AutoDiff takes the longest. We conduct the experiment using error-controlled adaptive time stepping methods, such as the Dormand-Price and the Fehlberg methods, and observe the same behavior. Reverse-mode AD records a copy of each participating variable *per operation*, resulting in a large stack of variables (second plot), that needs to be traversed in order to compute gradients. In contrast, Adjoint and Coupled both maintain a constant-size stack of variables while taking approximately the same computation time, while the later requires approximately twice as many dynamics evaluations as the former. Next, we investigate the scalability of the continuous sensitivity methods on a 100-link compound pendulum, requiring a parameter vector of size 100 to be estimated. We exclude AD from this comparison due to its prohibitively high computation time. Adjoint and Coupled both remain close in computation time, although Coupled requires $|\mathbf{x}| \times |\theta|$ sensitivites (subsection IV-G) compared to Adjoint's $|\theta|$ augmented state dimensions (subsection IV-H), which might be offset due to the need of solving two ODE in the case of Adjoint at each time step.

B. Automatic Robot Design

Industrial robotic applications often require a robot to follow a given path in task space with its end effector. In general, robotic arms with six or more degrees of freedom provide large workspaces and redundant configurations to reach any possible point within the workspace. However, motors are expensive to produce, maintain, and calibrate. Designing arms that contain a minimal number of motors required for a task provides economic and reliability benefits, but imposes constraints on the kinematic design of the arm.

One standard for specifying the kinematic configuration of a serial robot arm is the Denavit-Hartenberg (DH) parameterization [52]. For each joint i, the DH parameters are $(d_i, \theta_i, a_i, \alpha_i)$, describing the distance from joint i to motor axis $i-1$, the rotation about axis $i-1$, the distance of joint i along motor axis $i-1$, and the angle between motor axes i and $i-1$, respectively.

We specify a task-space trajectory $[\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_T]$ for $\mathbf{p}_t \in \mathbb{R}^3$ as the position in world coordinates of the robot's end-effector. Given a joint-space trajectory $[\mathbf{q}_0, \mathbf{q}_1, \ldots, \mathbf{q}_T]$,

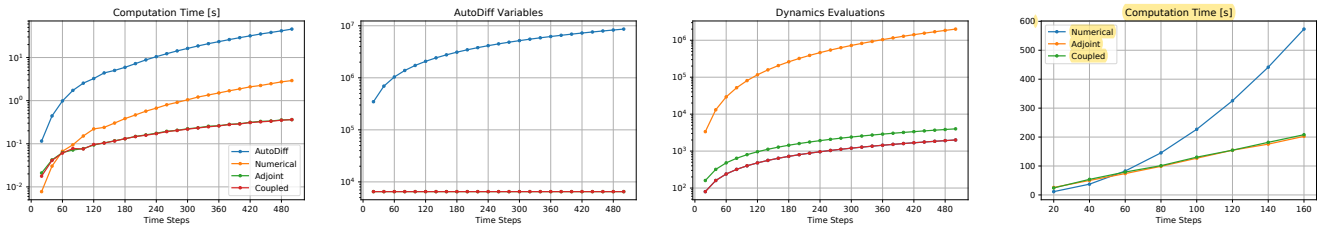

Fig. 2. Benchmarking results of the four Jacobian calculation methods considered in this work on the fourth-order Runge-Kutta integrator. *First three plots:* results for a double pendulum (logarithmic scale). *Last plot:* computation times for a 100-link compound pendulum.

we seek to find the best N-DOF robot arm design, parameterized by DH parameters $\boldsymbol{\theta}^* \in \mathbb{R}^{3N}$, that most closely matches the specified end-effector trajectory:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{t=0}^{T} || \operatorname{KIN}_{\boldsymbol{\theta}}(\mathbf{q}_t) - \mathbf{p}_t ||_2^2,$$

where the forward kinematics function $\operatorname{KIN}(\cdot)$ maps from joint space to the Cartesian coordinates of the end-effector, conditioned on DH parameters $\boldsymbol{\theta}$. Since we compute $\operatorname{KIN}(\cdot)$ using our engine, we may compute derivatives of arbitrary inputs to this function, and use gradient-based optimization through L-BFGS [53] from the Ceres optimization library [54] to converge to arm designs which accurately perform the trajectory-following task, as shown in Figure 3.

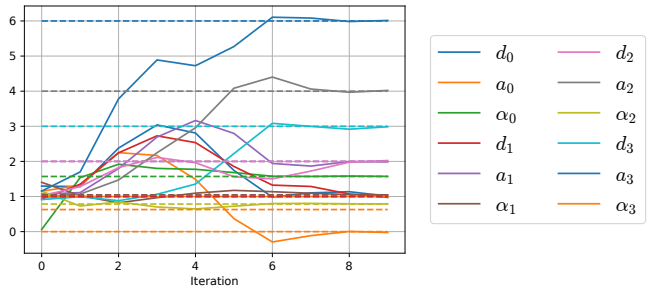

Fig. 3. Optimization of a 4-DOF robot arm design parameterized by the Denavit-Hartenberg (DH) parameters to match the robot arm that generated a given trajectory. (left) Evolution of the DH parameters over the optimization iterations.

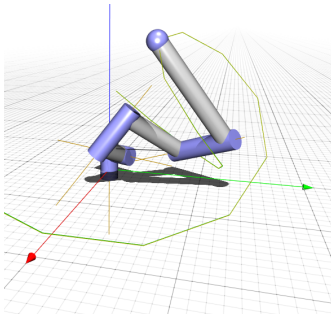

Fig. 4. Visualization of an exemplary 4-DOF robot arm and its trajectory in our simulator.

C. Adaptive MPC

Besides parameter estimation and design, a key benefit of differentiable physics is its applicability to optimal control algorithms. In order to control a system within our simulator, we specify the control space $\mathfrak{U}$, which is typically a subset of the system's generalized forces $\boldsymbol{\tau}$, and the state space $\mathfrak{X}$. Given a quadratic, i.e. twice-differentiable, cost function $c : \mathfrak{X} \times \mathfrak{U} \to \mathbb{R}$, we can linearize the dynamics at every time step, allowing efficient gradient-based optimal control techniques to be employed. Iterative Linear Quadratic Control [55] (iLQR) is a direct trajectory optimization algorithm that uses a dynamic programming scheme on the linearized dynamics to derive the control inputs that successively move the trajectory of states and controls closer to the optimum of the cost function.

Throughout our control experiments, we optimize a trajectory for an n-link cartpole to swing up from an arbitrary initial configuration of the joint angles. In the case of double cartpole, i.e. a double inverted pendulum on a cart, the state $\mathbf{x} \in \mathfrak{X}$ is defined as

$$\mathbf{x} = (p, \dot{p}, \sin q_0, \cos q_0, \sin q_1, \cos q_1, \dot{q}_0, \dot{q}_1, \ddot{q}_0, \ddot{q}_1)\,,$$

where p and $\dot{p}$ refer to the cart's position and velocity, $(q_0, q_1) = \mathbf{q}$ to the joint angles, and $(\dot{q}_0, \dot{q}_1) = \dot{\mathbf{q}}$, $(\ddot{q}_0, \ddot{q}_1) = \ddot{\mathbf{q}}$ to the velocities and accelerations of the revolute joints of the poles, respectively. For a single cartpole the state space is represented analogously, excluding the second revolute joint coordinates $q_1, \dot{q}_1, \ddot{q}_1$. The control input $\mathbf{u} \in \mathfrak{U}$ is a one-dimensional vector describing the force applied to the cart along the x axis. As typical for finite-horizon, discrete-time LQR problems, the cost of a trajectory over H time steps is defined as

$$\boldsymbol{J} = \sum_{k=0}^{H-1} (\tilde{\mathbf{x}}_k^{\mathrm{T}} Q \tilde{\mathbf{x}}_k + \mathbf{u}_k^{\mathrm{T}} R \mathbf{u}_k) + \tilde{\mathbf{x}}_H^{\mathrm{T}} S \tilde{\mathbf{x}}_H, \qquad (3)$$

where $\tilde{\mathbf{x}}_k = \mathbf{x}^* - \mathbf{x}_k$, and the matrices $Q, S \in \mathbb{R}^{|\mathbf{x}| \times |\mathbf{x}|}$ and $R \in \mathbb{R}^{|\mathbf{u}| \times |\mathbf{u}|}$ weight the contributions of each dimension of the state and control input. Throughout this experiment, we set Q, S, R to be diagonal matrices. Minimizing the cost function drives the system to the defined goal state[2] $\mathbf{x}^* = (0, 0, 0, 1, 0, 1, 0, 0, 0, 0)$, at which the pole is upright at zero

[2]The goal state is given for a double cartpole here, it is analogously defined for a single cartpole.

Algorithm 3 Adaptive MPC algorithm using differentiable physics model $f_{\boldsymbol{\theta}}$.

Require: Cost function $\boldsymbol{J}$, episode length M, trajectory length T, horizon length H

for episode = $1..M$ **do**

$R \leftarrow \emptyset$ ▷ Replay buffer to store transition samples from the real environment

Obtain initial state $\mathbf{x}_0^*$ from the real environment

for $t = 1..T$ **do**

$\{\mathbf{u}^*\}_t^{t+H} \leftarrow \underset{\mathbf{u}_{1:H}}{\arg\min}\, \boldsymbol{J}$ ▷ Trajectory optimization using iLQR with cost from Equation 3

$\text{s.t. } \mathbf{x}_1 = \mathbf{x}_t^*, \quad \mathbf{x}_{i+1} = \int f([\mathbf{x}_i, \mathbf{u}_i], i, \boldsymbol{\theta}), \quad \underline{\mathbf{u}} \leq \mathbf{u} \leq \bar{\mathbf{u}}$

Take action $\mathbf{u}_t^*$ in the real environment and obtain next state $\mathbf{x}_{t+1}^*$

Store transition $(\mathbf{x}_t^*, \mathbf{u}_t^*, \mathbf{x}_{t+1}^*)$ in R

end for

Fit dynamics model f to real data R by minimizing the state-action prediction loss (Equation 4)

end for

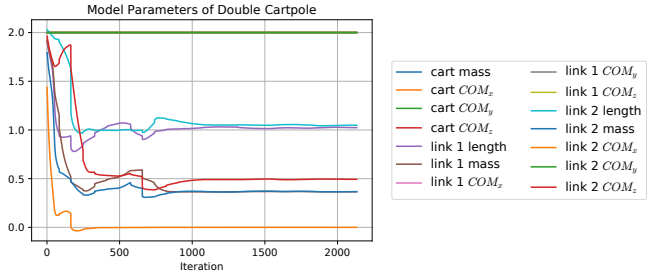

Fig. 5. Convergence of the physical parameters of a double cartpole, over all model fitting iterations combined, using Adaptive MPC (Algorithm 3) in the DeepMind Control Suite environment.

angular velocity and acceleration, and the cart is centered at the origin with zero positional velocity.

Trajectory optimization assumes that the dynamics model is accurate w.r.t the real world and generates sequences of actions that achieve optimal behavior toward a given goal state, leading to open-loop control. Model-predictive control (MPC) leverages trajectory optimization in a feedback loop where the next action is chosen as the first control computed by trajectory optimization over a shorter time horizon with the internal dynamics model. After some actions are executed in the real world and subsequent state samples are observed, *adaptive* MPC (Algorithm 3) fits the dynamics model to these samples to align it closer with the real-world dynamics. In this experiment, we want to investigate how differentiable physics can help overcome the domain shift that poses an essential challenge of model-based control algorithms that are employed in a different environment. To this end, we incorporate our simulator as dynamics model in such receding-horizon control algorithm to achieve swing-up motions of a single and double cartpole in the DeepMind Control Suite [1] environments that are based on the MuJoCo physics simulator.

We fit the parameters $\boldsymbol{\theta}$ of the simulator by minimizing the prediction loss given the state-action transition $(\mathbf{x}_t^*, \mathbf{u}_t^*, \mathbf{x}_{t+1}^*)$ from the real system:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\arg\min} \sum_t || \int f([\mathbf{x}_t^*, \mathbf{u}_t^*], t, \boldsymbol{\theta}) - \mathbf{x}_{t+1}^* ||_2^2 \quad (4)$$

Thanks to the low dimensionality of the model parameter vector $\boldsymbol{\theta}$ (for a double cartpole there are 14 parameters, cf. Figure 5), efficient optimizers such as the quasi-Newton optimizer L-BFGS are applicable, leading to fast convergence of the fitting phase, typically within 10 optimization steps. The length T of one episode is 140 time steps. During the first episode we fit the dynamics model more often, i.e. every 50 time steps, to warm-start the receding-horizon control scheme. Given a horizon size H of 20 and 40 time steps, MPC is able to find the optimal swing-up trajectory for the single and double cartpole, respectively.

Within a handful of training episodes, adaptive MPC infers the correct model parameters involved in the dynamics of a double cartpole (Figure 5). As shown in Figure 1, the models we start from do not match their counterparts from DeepMind Control Suite. For example, the poles are represented by capsules where the mass is distributed across these elongated geometries, whereas initially in our model, the center of mass of the links is at the end of them, such that they have different inertia parameters. We set the masses, lengths of the links, and 3D coordinates of the center of masses to 2, and, using a few steps of the optimizer and less than 100 transition samples, converge to a much more accurate model of the true dynamics in the MuJoCo environment.

VI. CONCLUSION

We introduced a novel differentiable physical simulator, and presented experiments for the inference of physical parameters, optimal control and system design. Since it is constrained to the laws of physics, such as conservation of energy and momentum, our proposed model provides a large, meaningful inductive bias on robot learning problems. Within a handful of trials in out test environment, our gradient-based representation of rigid-body dynamics allows an adaptive MPC scheme to infer the model parameters of the system thereby allowing it to make predictions and plan for actions many time steps ahead. We look forward to exercising this physics engine for learning and control to solve complex tasks on physical robot systems.

REFERENCES

[1] Y. Tassa, Y. Doron, A. Muldal, T. Erez, Y. Li, D. de Las Casas, D. Budden, A. Abdolmaleki, J. Merel, A. Lefrancq, T. P. Lillicrap, and M. A. Riedmiller, "Deepmind control suite," *CoRR*, vol. abs/1801.00690, 2018. [Online]. Available: http://arxiv.org/abs/1801.00690

[2] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *International Conference on Intelligent Robots and Systems*, Oct 2012, pp. 5026–5033.

[3] J. Degrave, M. Hermans, J. Dambre, and F. wyffels, "A differentiable physics engine for deep learning in robotics," *Frontiers in Neurorobotics*, vol. 13, p. 6, 2019. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnbot.2019.00006

[4] M. Giftthaler, M. Neunert, M. Stäuble, M. Frigerio, C. Semini, and J. Buchli, "Automatic differentiation of rigid body dynamics for optimal control and estimation," *Advanced Robotics*, vol. 31, no. 22, pp. 1225–1237, 2017.

[5] M. Frigerio, J. Buchli, D. G. Caldwell, and C. Semini, "RobCoGen: a code generator for efficient kinematics and dynamics of articulated robots, based on Domain Specific Languages," vol. 7, no. 1, pp. 36–54, 2016.

[6] B. Carpenter, M. D. Hoffman, M. Brubaker, D. Lee, P. Li, and M. Betancourt, "The stan math library: Reverse-mode automatic differentiation in C++," *CoRR*, vol. abs/1509.07164, 2015. [Online]. Available: http://arxiv.org/abs/1509.07164

[7] J. Carpentier and N. Mansard, "Analytical derivatives of rigid body dynamics algorithms," in *Robotics: Science and Systems*, 2018.

[8] T. Koolen and R. Deits, "Julia for robotics: simulation and real-time control in a high-level programming language," in *International Conference on Robotics and Automation*, 05 2019.

[9] F. de Avila Belbute-Peres, K. Smith, K. Allen, J. Tenenbaum, and J. Z. Kolter, "End-to-end differentiable physics for learning and control," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 7178–7189. [Online]. Available: http://papers.nips.cc/paper/7948-end-to-end-differentiable-physics-for-learning-and-control.pdf

[10] B. Amos and J. Z. Kolter, "OptNet: Differentiable optimization as a layer in neural networks," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 70. PMLR, 2017, pp. 136–145.

[11] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," in *Advances in Neural Information Processing Systems*, 2018, pp. 6571–6583.

[12] C. Rackauckas, Y. Ma, V. Dixit, X. Guo, M. Innes, J. Revels, J. Nyberg, and V. Ivaturi, "A comparison of automatic differentiation and continuous sensitivity analysis for derivatives of differential equation solutions," *arXiv preprint arXiv:1812.01892*, 2018.

[13] R. Serban and A. C. Hindmarsh, "Cvodes: An ode solver with sensitivity analysis capabilities," Tech. Rep., 2003.

[14] A. W. Moore, "Fast, robust adaptive control by learning only forward models," in *Advances in Neural Information Processing Systems*, J. E. Moody, S. J. Hanson, and R. P. Lippmann, Eds. Morgan-Kaufmann, 1992, pp. 571–578. [Online]. Available: http://papers.nips.cc/paper/585-fast-robust-adaptive-control-by-learning-only-forward-models.pdf

[15] C. G. Atkeson, A. W. Moore, and S. Schaal, "Locally weighted learning for control," *Artificial Intelligence Review*, vol. 11, no. 1, pp. 75–113, Feb 1997. [Online]. Available: https://doi.org/10.1023/A:1006511328852

[16] A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, and P. Battaglia, "Graph networks as learnable physics engines for inference and control," in *International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. Stockholmsmässan, Stockholm Sweden: PMLR, 10–15 Jul 2018, pp. 4470–4479. [Online]. Available: http://proceedings.mlr.press/v80/sanchez-gonzalez18a.html

[17] Y. Li, J. Wu, R. Tedrake, J. B. Tenenbaum, and A. Torralba, "Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=rJgbSn09Ym

[18] Z. Liu, J. Wu, Z. Xu, C. Sun, K. Murphy, W. T. Freeman, and J. B. Tenenbaum, "Modeling parts, structure, and system dynamics via predictive learning," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=rJe10iC5K7

[19] P. Battaglia, R. Pascanu, M. Lai, D. J. Rezende, *et al.*, "Interaction networks for learning about objects, relations and physics," in *Advances in Neural Information Processing Systems*, 2016, pp. 4502–4510.

[20] M. B. Chang, T. Ullman, A. Torralba, and J. B. Tenenbaum, "A compositional Object-Based approach to learning physical dynamics," *International Conference on Learning Representations*, Dec. 2017.

[21] C. Schenck and D. Fox, "SPNets: Differentiable fluid dynamics for deep neural networks," *Conference on Robot Learning*, 2018.

[22] D. Mrowca, C. Zhuang, E. Wang, N. Haber, L. Fei-Fei, J. B. Tenenbaum, and D. L. Yamins, "Flexible neural representation for physics prediction," in *Advances in Neural Information Processing Systems*, 2018.

[23] Z. Xu, J. Wu, A. Zeng, J. B. Tenenbaum, and S. Song, "DensePhysNet: Learning dense physical object representations via multi-step dynamic interactions," *Robotics: Science and Systems*, June 2019.

[24] J. Wu, I. Yildirim, J. J. Lim, W. T. Freeman, and J. B. Tenenbaum, "Galileo: Perceiving physical object properties by integrating a physics engine with deep learning," in *Advances in Neural Information Processing Systems*. Cambridge, MA, USA: MIT Press, 2015, pp. 127–135. [Online]. Available: http://dl.acm.org/citation.cfm?id=2969239.2969254

[25] J. Wu, J. J. Lim, H. Zhang, J. B. Tenenbaum, and W. T. Freeman, "Physics 101: Learning physical object properties from unlabeled videos," in *British Machine Vision Conference*, 2016.

[26] J. Wu, E. Lu, P. Kohli, B. Freeman, and J. Tenenbaum, "Learning to see physics via visual de-animation," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 153–164. [Online]. Available: http://papers.nips.cc/paper/6620-learning-to-see-physics-via-visual-de-animation.pdf

[27] C. Finn, I. Goodfellow, and S. Levine, "Unsupervised learning for physical interaction through video prediction," in *Advances in Neural Information Processing Systems*, 2016, pp. 64–72.

[28] M. Janner, S. Levine, W. T. Freeman, J. B. Tenenbaum, C. Finn, and J. Wu, "Reasoning about physical interactions with object-centric models," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=HJx9EhC9tQ

[29] S. He, Y. Li, Y. Feng, S. Ho, S. Ravanbakhsh, W. Chen, and B. Póczos, "Learning to predict the cosmological structure formation," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 116, no. 28, pp. 13 825–13 832, July 2019.

[30] M. Raissi, H. Babaee, and P. Givi, "Deep learning of turbulent scalar mixing," Tech. Rep., 2018.

[31] 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

[32] S. Greydanus, M. Dzamba, and J. Yosinski, "Hamiltonian neural networks," *arXiv preprint arXiv:1906.01563*, 2019.

[33] S. Kolev and E. Todorov, "Physically consistent state estimation and system identification for contacts," *International Conference on Humanoid Robots*, pp. 1036–1043, 2015.

[34] S. Zhu, A. Kimmel, K. E. Bekris, and A. Boularias, "Fast model identification via physics engines for data-efficient policy search," in *International Joint Conferences on Artificial Intelligence*, 2018.

[35] T. Reichenbach, "A dynamic simulator for humanoid robots," *Artificial Life and Robotics*, vol. 13, no. 2, pp. 561–565, Mar 2009. [Online]. Available: https://doi.org/10.1007/s10015-008-0508-6

[36] A. Farchy, S. Barrett, P. MacAlpine, and P. Stone, "Humanoid robots learning to walk faster: From the real world to simulation and back," in *International Conference on Autonomous Agents and Multiagent Systems*, May 2013. [Online]. Available: http://www.cs.utexas.edu/users/ai-lab/?AAMAS13-Farchy

[37] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. D. Ratliff, and D. Fox, "Closing the sim-to-real loop: Adapting simulation randomization with real world experience," *International Conference on Robotics and Automation*, 2019.

[38] 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. [Online]. Available: https://doi.org/10.1007/s10846-017-0468-y

[39] J. Ko, D. J. Klein, D. Fox, and D. Haehnel, "Gaussian processes and reinforcement learning for identification and control of an autonomous blimp," in *International Conference on Robotics and Automation*, April 2007, pp. 742–747.

[40] M. Deisenroth and C. E. Rasmussen, "Pilco: A model-based and data-efficient approach to policy search," in *International Conference on machine learning*, 2011, pp. 465–472.

[41] J. Boedecker, J. T. Springenberg, J. Wülfing, and M. Riedmiller, "Approximate real-time optimal control based on sparse gaussian process models," in *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, Dec 2014, pp. 1–8.

[42] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Aggressive driving with model predictive path integral control," in *International Conference on Robotics and Automation*, May 2016, pp. 1433–1440.

[43] A. Yamaguchi and C. G. Atkeson, "Neural networks and differential dynamic programming for reinforcement learning problems," in *International Conference on Robotics and Automation*. IEEE, 2016, pp. 5434–5441.

[44] J. Fu, S. Levine, and P. Abbeel, "One-shot learning of manipulation skills with online dynamics adaptation and neural network priors," in *International Conference on Intelligent Robots and Systems*. IEEE/RSJ, 2016, pp. 4019–4026.

[45] S. Depeweg, J. M. Hernández-Lobato, F. Doshi-Velez, and S. Udluft, "Learning and policy search in stochastic dynamical systems with bayesian neural networks," *International Conference on Learning Representations*, May 2017.

[46] Y. Gal, R. McAllister, and C. E. Rasmussen, "Improving PILCO with Bayesian neural network dynamics models," in *Data-Efficient Machine Learning workshop, International Conference on Machine Learning*, 2016.

[47] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep reinforcement learning in a handful of trials using probabilistic dynamics models," in *Advances in Neural Information Processing Systems*, 2018, pp. 4754–4765.

[48] R. Featherstone, *Rigid Body Dynamics Algorithms*. Berlin, Heidelberg: Springer-Verlag, 2007.

[49] M. E. Jerrel, "Automatic differentiation and interval arithmetic for estimation of disequilibrium models," *Computational Economics*, vol. 10, no. 3, pp. 295–316, Aug 1997. [Online]. Available: https://doi.org/10.1023/A:1008633613243

[50] L. S. Pontryagin, E. Mishchenko, V. Boltyanskii, and R. Gamkrelidze, "The mathematical theory of optimal processes," 1962.

[51] G. Guennebaud, B. Jacob, *et al.*, "Eigen v3," http://eigen.tuxfamily.org, 2010.

[52] R. S. Hartenberg and J. Denavit, "A kinematic notation for lower pair mechanisms based on matrices," *Journal of applied mechanics*, vol. 77, no. 2, pp. 215–221, 1955.

[53] D. C. Liu and J. Nocedal, "On the limited memory bfgs method for large scale optimization," *Mathematical Programming*, vol. 45, no. 1, pp. 503–528, Aug 1989. [Online]. Available: https://doi.org/10.1007/BF01589116

[54] S. Agarwal, K. Mierle, and Others, "Ceres solver," http://ceres-solver.org.

[55] W. Li and E. Todorov, "Iterative linear quadratic regulator design for nonlinear biological movement systems." in *International Conference on Informatics in Control, Automation and Robotics*, 2004.