Closing the Reality Gap via Simulation-based Inference and Control

by

Eric Heiden

A Dissertation Presented to the
FACULTY OF THE USC GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(COMPUTER SCIENCE)

August 2022

Acknowledgements

This thesis is the product of a number of collaborations with many people who I would like to express my appreciation for.

First, I would like to thank my Ph.D. advisor, Gaurav Sukhatme, for enabling me to explore a variety of research areas while being a great mentor in all of my endeavors. You have helped me tremendously in shaping my work and provided an excellent collaborative atmosphere throughout my time at USC.

I want to thank Sven Koenig with whom I started my first research projects in collaboration with Luigi Palmieri and Leonard Bruns. Initiated in my Master's program, our work has been one of the key motivators for me to pursue a Ph.D. Furthermore, I would like to thank the other USC professors who are examining my thesis as part of my defense committee. Jernej Barbič, Stefanos Nikolaidis, and Satyandra K. Gupta – thank you for providing your guidance and insights.

I am grateful to my fellow Ph.D. students and post-docs who I collaborated with over the course of my program. David Millard, Chris Denniston, Ryan Julian, Ragesh Ramachandran, Gautam Salhotra, Peter Englert, Karol Hausman, James Preiss, as well as the students I had the pleasure to mentor, namely, Ziang Liu, Hejia Zhang, Yizhou Sheng, Shubham Agrawal, Vedant Mistry and Zhanpeng He – I truly enjoyed working with all of you and appreciate greatly the outstanding effort you invested into our projects.

I wish to express my gratitude to my collaborators in the industry who I worked with during my internships and beyond: Nvidia (Fabio Ramos, Miles Macklin, Yashraj Narang, Animesh Garg, Dieter Fox), Google (Erwin Coumans, Vikas Sindhwani, Jie Tan), NASA JPL (Ali Agha, Ben Morrell, Daniel Pastor,

Kamak Ebadi, Luca Carlone), as well as Vibhav Vineet from Microsoft Research. I had an amazing time learning from you during our discussions and brainstorming sessions, and I am very grateful for the help and insights you have provided me that made these experiences memorable.

I am grateful for the Google Fellowship that sponsored much of my work.

Finally, I would like to thank my friends and family for their tremendous support throughout the years.

Table of Contents

List of Tables

List of Figures

Abstract

Simulators play a crucial role in robotics – serving as training platforms for reinforcement learning agents, informing hardware design decisions, or facilitating the prototyping of new perception and control pipelines, among many other applications. While their predictive power offers generalizability and accuracy, a core challenge lies in the mismatch between the simulated and the real world. This thesis addresses the reality gap in robotics simulators from three angles.

First, through the lens of robotic control, we investigate a robot learning pipeline that transfers skills acquired in simulation to the real world by composing task embeddings, offering a solution orthogonal to commonly used transfer learning approaches. Further, we develop an adaptive model-predictive controller that leverages a differentiable physics engine as a world representation that is updatable from sensor measurements.

Next, we develop two differentiable simulators that tackle particular problems in robotic perception and manipulation. To improve the accuracy of LiDAR sensing modules, we build a physically-based model that accounts for the measurement process in continuous-wave LiDAR sensors and the interaction of laser light with various surface materials. In robotic cutting, we introduce a differentiable simulator for the slicing of deformable objects, enabling applications in system identification and trajectory optimization.

Finally, we explore techniques that extend the capabilities of simulators to enable their construction and synchronization from real-world sensor measurements. To this end, we present a Bayesian inference algorithm that finds accurate simulation parameter distributions from trajectory-based observations. Next,

we introduce a hybrid simulation approach that augments an analytical physics engine by neural networks to enable the learning of dynamical effects unaccounted for in a rigid-body simulator. In closing, we present an inference pipeline that finds the topology of articulated mechanisms from a depth or RGB video while estimating the dynamical parameters, yielding a comprehensive, interactive simulation of the real system.

Chapter 1

Introduction

Employing robots in the real world to perform a wide variety of tasks autonomously remains a great challenge to current perception, planning and control algorithms. The performance of such embodied agents in unstructured environments largely depends on their ability to reason about the environment. Whether it is learned implicitly or hand-crafted, such capability is afforded through a model of the world that empowers robots to devise and execute plans over long time horizons to solve complex tasks.

In this thesis, we propose to use simulators as world models that can be updated from sensor measurements to systematically improve their predictive accuracy on real-world problems. We follow the simulation-based inference and control methodology to create simulators and update their parameters from sensor measurements, and bridge modeling gaps through learning-based augmentations. In robotic perception and manipulation applications, we demonstrate how our approaches accurately reproduce physical processes and transfer behaviors from simulation to the real world successfully.

1.1 Specialized Representations for Motion Planning

Various specialized representations of the environment have been proposed which are used in robot autonomy pipelines that fuse perception, planning and control. These approaches, such as occupancy grids [231,

70, 315, 346], confidence-rich grid maps (CRM) [2, 127], and Gaussian Processes [246, 273], are highly interpretable in a way that humans can reason about the uncertainty of the system, and what additional measurements are necessary to improve the predictions. On the other hand, these domain-limited frameworks do not allow the robot to learn from experience or adapt to changing task requirements. Recent simultaneous localization and mapping (SLAM) approaches, such as LAMP [68] and Kimera [280], introduce capabilities to explore and navigate in large-scale environments where robots need to avoid moving objects and thereby reason about their dynamical motion. Nonetheless, the world models generated by these pipelines are limited to spatial mapping for the purpose of obstacle avoidance. Such occupancy information can be leveraged by motion planning algorithms, such as the methods we benchmark in Heiden et al. [125] for the application on nonholonomic mobile robots, or our path improvement algorithm GRIPS [126]. Given a probabilistic map representation, such as CRM, uncertainty-aware motion planners can find safe and fast trajectories, such as the method we developed in Heiden et al. [118] based on the lower confidence bound objective. Besides the rich applicability to motion planning, the present environment representations do not afford semantic reasoning about the environment, e.g., how to manipulate and interact with tools and other objects. These models are limited to the application in autonomous navigation and are inadequate for robotic manipulation in everyday household tasks, where predicting the future outcome of dynamical processes is the key ingredient to achieving intelligent behavior.

1.2 Learning-based Control

Learning-based approaches have found tremendous success in domains where large amounts of labeled data is available. Many problems in robotics, however, do not belong to such regime where training samples are easily obtainable. Instead, it is often only possible to provide few kinesthetic demonstrations [294, 131], rely on self-supervised or reinforcement learning [43, 226], or a combination of imitation and reinforcement learning (RL) [356, 254, 256, 204]. Model-based reinforcement learning explicitly leverages

learned world models, such as Gaussian Processes [60] or graph-based neural networks [292], for control generation, whereas model-free methods implicitly encode the dynamics of the environment through the learned policy network [298, 299], value or Q function [224]. While the longstanding motivation behind such approaches is to enable robots to improve by learning from their own experience, the current instantiations of state-of-the-art RL algorithms, even model-based, require extensive trial-and-error samples. This necessitates, in most cases, the use of simulators to provide a risk-free environment that runs orders of magnitude faster than real time. With regards to accountable AI, many of these approaches are not human-interpretable – they may achieve high performance on certain tasks yet it remains an open research question how the training setup must be designed to guarantee performance throughout all metrics over the tasks of interest.

1.3 Simulators and the Reality Gap

Whether control policies are learned through reinforcement learning, or feedback loops are optimized – in most approaches simulators are used to collect training experience or validate algorithms before deploying them on the real system. They are an indispensable tool for the design of new mechanisms [347] and the prototyping of complex systems [163] that would be too costly or dangerous to physically implement for each iteration of the development process. Simulators, such as physics engines, have found applications in numerous fields of science [55] and encode our understanding of the world through the laws of physics, which generalize well to large realms of systems. Their defining variables, such as the geometry of the objects, are human-interpretable and can even be verified through a variety of specialized measurement tools. An inherent problem, however, is the disparity between the simulated and the real world, i.e., the *sim2real gap*. Behaviors attained in simulation often do not directly translate to real systems, and the predicted behaviors often do not match the actual observations. Various methods have been proposed to overcome this issue, such as domain randomization [318, 255, 42, 217, 274], domain adaptation [264, 256],

and meta learning [79, 49]. These transfer learning approaches focus on the training process to improve the robustness of policies trained in simulation when being deployed in the real world.

1.4 Simulation-based Inference and Control

In this thesis, we propose methods that approach the reality gap from a simulation-based inference and control perspective. We propose to design the simulator from the ground up to be *invertible*, such that we can estimate the simulation settings from the observations of the real system. To realize this promise, we leverage *differentiable simulators* that efficiently calculate analytical gradients of their model parameters, as well as state and control variables. Similar to gradSim [239], such an approach allows us to integrate a number of models of the real world, i.e. a rendering engine reproducing a camera sensor, and articulated rigid body dynamics and contact mechanics models predicting the behavior of articulated mechanisms. Eventually, we learn these highly interpretable simulators as digital twins of the real system from pixel-based observations in our inference pipeline [120] in Sec. 4.3. Where learning-based models, such as intuitive physics approaches [292, 189, 199, 18, 41, 296, 234, 348] generalize poorly to changing conditions (e.g. the number of interacting bodies and their dynamical properties) unseen in the training dataset, simulators provide physically grounded predictions, as long as (1) the models they implement capture the relevant dynamics of the real system, and (2) the model parameters have been estimated correctly.

We develop methods that address both of these requirements in this thesis. Differentiable simulators, such as our DiSECt [121] (Sec. 3.2) and LiDAR [119] (Sec. 3.1) simulators, can be integrated with deep learning architectures and be backpropagated through to optimize them in conjunction with the neural network parameters. We leverage this idea in our NeuralSim [122] approach in Sec. 4.2, where data-driven function approximators are inserted into a differentiable physics engine to learn parts of the dynamics (such as air resistance or contact friction) unaccounted for by the analytical models. In Sec. 4.1, we introduce a Bayesian inference algorithm [117] that is tailored to trajectory observations of nonlinear systems that

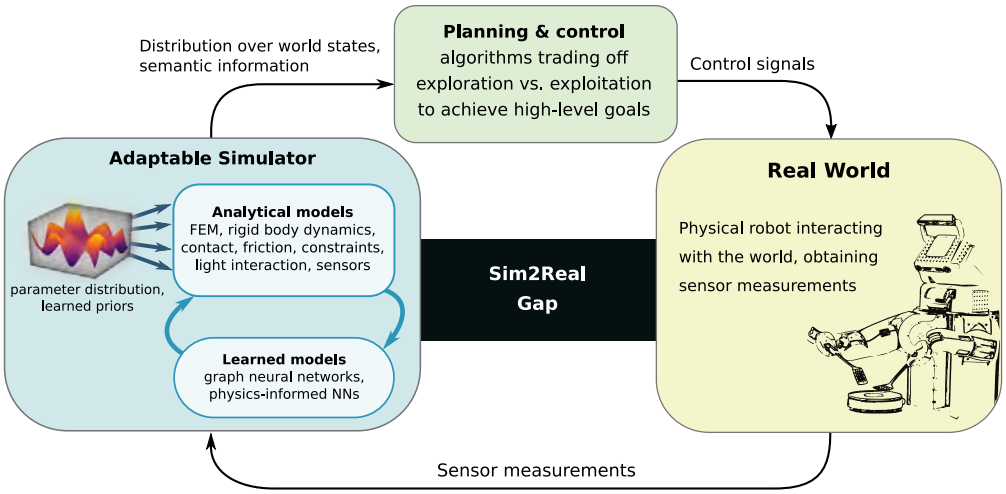


Figure 1.1: Proposed framework in which the simulator is augmented by learned models to calibrate itself to sensor measurements taken from the real robot. The simulator serves as the representation of the environment that enables high-level planning and control generation, resulting in control commands that are executed in the real world. These generated trajectories may trade off task performance and exploration in order to yield more informative sensor measurements in the system identification phase. The simulation-based control loop is closed such that the simulator is iteratively becoming more accurate in its predictions, thereby closing the sim2real gap.

allows the estimation of distributions over simulation parameters while exploiting differentiability and parallelism of a differentiable simulator. Having obtained methods for simulation-based inference, we propose an adaptive model-predictive controller in Sec. 2.2 that leverages a differentiable physics engine as a world model that can be updated from sensor measurements. As shown in Fig. 1.1, in combination with our aforementioned methods on augmenting and updating physics engines, we envision a closed-loop control pipeline that iteratively builds a simulation model of the real world while interacting with it through robotic control that relies on the semantic and dynamic information the simulator provides. Furthermore, orthogonal to previous transfer learning approaches, we present a robot learning pipeline in Sec. 2.1 that learns a rich embedding space of skills in simulation which is leveraged by the controller on the real system [156].

1.5 Outline

The thesis is structured as follows:

- Chapter 2 focuses on approaches to bridge the sim2real gap in robotic control. To this end, we introduce a robot learning pipeline that transfers skills acquired in simulation to the real world by composing task embeddings. Next, we set the background on the differentiable simulation methodology that underlies the remainder of this thesis. Here, we present methods and experimental results on system identification, optimal design, and closed-loop control.

- Chapter 3 describes two specific differentiable simulators that serve applications in robot perception and manipulation. To improve the accuracy of LiDAR sensing modules, we build a physically-based model that accounts for the measurement process in continuous-wave LiDAR sensors and the interaction of laser light with various surface materials. In robotic cutting, we introduce a differentiable simulator for the slicing of deformable objects, enabling applications in system identification and trajectory optimization.

- Chapter 4 presents a Bayesian inference algorithm that finds accurate simulation parameter distributions from trajectory-based observations. We further introduce a hybrid modeling approach that augments an analytical physics engine by neural networks to enable the learning of dynamical effects outside the domain of a rigid-body simulator. Next, we present an inference pipeline that finds the topology of articulated mechanisms from a depth or RGB video while estimating the dynamical parameters, yielding a comprehensive, interactive simulation of the real system.

- Chapter 5 concludes this thesis with future research directions that leverage the potential of simulation-based inference and control, and provides a few remarks on outstanding problems that lie on the way to this future.

Chapter 2

Interactive Sim2Real Transfer

In this chapter, we investigate approaches that transfer control behaviors from a simulator to the real system. In Sec. 2.1, we introduce a transfer learning method [156] that learns an embedding space of robotic control skills in simulation which are leveraged for planning online in the real world. Besides planning, we further investigate a model-predictive control approach [115] that composes skills learned in simulation to achieve high-level tasks on a real Sawyer robot.

Sec. 2.2 lays the foundation of the *differentiable simulation* methodology that underlies the remainder of this thesis. Differentiable simulators provide efficient means to calculate the derivatives of simulation quantities, such as dynamics parameters and state variables. Such capability enables efficient, gradient-based optimization algorithms to be applied in several application areas, such as system identification, optimal design, and control. We demonstrate the effectiveness of our simulation-based approach in all three scenarios through several experiments [124]. Additionally, as we detailed in Heiden, Millard, and Sukhatme [123], we derive a learning-based model that can approximate multi-modal distributions over the simulation parameters from observation trajectories. This section concludes with the derivation of a closed-loop control algorithm that interweaves parameter estimation and trajectory optimization to yield an adaptive model-predictive controller which is able to control highly nonlinear systems.

2.1 Scaling Simulation-to-Real Transfer by Learning a Latent Space of Robot Skills

We present a strategy for simulation-to-real transfer, which builds on recent advances in robot skill decomposition. Rather than focusing on minimizing the simulation-reality gap, we propose a method for increasing the sample efficiency and robustness of existing simulation-to-real approaches which exploits hierarchy and online adaptation. Instead of learning a unique policy for each desired robotic task, we learn a diverse set of skills and their variations, and embed those skill variations in a continuously-parameterized space. We then interpolate, search, and plan in this space to find a transferable policy which solves more complex, high-level tasks by combining low-level skills and their variations. In this work, we first characterize the behavior of this learned skill space, by experimenting with several techniques for composing pre-learned latent skills. We then discuss an algorithm which allows our method to perform long-horizon tasks never seen in simulation, by intelligently sequencing short-horizon latent skills. Our algorithm adapts to unseen tasks online by repeatedly choosing new skills from the latent space, using live sensor data and simulation to predict which latent skill will perform best next in the real world. Importantly, our method learns to control a real robot in joint-space to achieve these high-level tasks with little or no on-robot time, despite the fact that the low-level policies may not be perfectly transferable from simulation to real, and that the low-level skills were not trained on any examples of high-level tasks. In addition to our results indicating a lower sample complexity for families of tasks, we believe that our method provides a promising template for combining learning-based methods with proven classical robotics algorithms such as model-predictive control (MPC).

2.1.1 Introduction

The Constructivist hypothesis proposes that humans learn to perform new behaviors by using what they already know [52]. To learn new behaviors, it proposes that humans leverage their prior experiences across

behaviors, and that they also generalize and compose previously-learned behaviors into new ones, rather than learning them from scratch [63]. Whether we can make robots learn so efficiently is an open question.

Instead of drawing on prior experience when faced with a new task, the agents considered in most work on "deep" reinforcement learning (RL) acquire new behavior from scratch. They achieve superhuman performance in various continuous control [191] and game play domains [224], but can do so only for a single particular task. This in turn hampers their sample efficiency by requiring millions of samples to converge [65] in a new environment. While recent attempts in deep RL for robotics are encouraging [183, 44, 103], data efficiency and generality remain the main challenges of such approaches. Learning from scratch using these algorithms on a real robot is therefore a resource-intensive endeavor, e.g. by requiring multiple robots to learn in parallel for weeks [184].

One promising approach to overcome the sample inefficiency in deep RL for robotics is to train agents entirely in faster-than-real-time simulation, and transfer the learned policies to a real robot (i.e. sim2real). Sim2real approaches suffer from the "reality gap" problem: it is practically impossible to implement a perfect-fidelity simulation which exhibits all of the physical intricacies of the real world. A learner trained in simulation and then deployed in the real world will eventually encounter a mismatch between its training-time and test-time experiences, and must either generalize across the reality gap, or adapt itself to the new real-world environment. Many sim2real methods focus directly on closing the reality gap or transferring across it, at the expense of significant up-front engineering effort and often additional sample complexity. Instead, we propose a method for increasing the sample efficiency of almost any sim2real method, which allows this effort and sample complexity to be amortized among many tasks.

Our contribution is a method for increasing the sample efficiency and robustness of the sim2real framework which exploits hierarchy, while retaining the flexibility and expressiveness of end-to-end RL approaches. As we will show, our method's hierarchical formulation also provides a template for mixing reinforcement learning for proven robotics algorithms, such as search-based planning and model-predictive control (MPC).

Consider the illustrative example of block stacking. One approach is to learn a single monolithic policy which, given any arrangement of blocks on a table, grasps, moves, and stacks each block to form a tower. This formulation is succinct, but requires learning a single sophisticated policy in simulation and then transferring that to the real world. We observe that block stacking–and many other practical robotics tasks–is easily decomposed into a few reusable primitive skills (e.g. locate and grasp a block, move a grasped block over the stack location, place a grasped block on top of a stack), and divide the problem into two parts: learning to perform and mix the skills in general in simulation, and learning to combine these skills into particular policies which achieve high-level tasks in the real world.

This article draws on earlier work published in a conference paper [155]. Here, we provide more detailed explanations and discussions of our sim2real framework. Additionally, this article newly introduces an algorithm and supporting experiments, based on the template of model-predictive control, which allows our method to be used for zero-shot transfer learning with real robots. Our algorithm is simple, and leverages both latent skill spaces and online simulation to minimize the real-robot samples used to acquire a new task.

2.1.2 Related Work

Learning skill representations Learning skill representations to aid in generalization has been proposed in works old and new. Previous works proposed frameworks such as Associative Skill Memories [251] and probabilistic movement primitives [285] to acquire a set of reusable skills.

Multi-task learning and hierarchical reinforcement learning Our approach builds on the work of Hausman et al. [113], which learns a latent space which parameterizes a set of motion skills, and shows them to be temporally composable using interpolation between coordinates in the latent space. Importantly, our work shows that this idea, which was previously only evaluated in simulation using large amounts of training data, can be applied to real robotics problems with practical amounts of training data using a sim2real framework. In addition to learning reusable skills as in Hausman et al. [113], we characterize the behavior of these latent spaces, evaluate several methods for learning to compose latent skills to achieve high-level tasks with real robots, and propose a simple algorithm which allows these latent skill spaces to be used for zero-shot adaptation to new tasks on real robots.

Similar latent-space methods have been recently used for hierarchical RL in simulation [116, 107, 278]. Other approaches introduce particular model architectures for multitask learning, such as Progressive Neural Networks [287], Attention Networks [270], and ensembles of multiple expert models [271, 180].

Using latent skill representations for exploration Exploration is a key problem in robot learning, and our method uses latent skill representations to address this problem. Using learned latent spaces to make exploration more tractable is also studied by Gu et al. [103], Eysenbach et al. [72] and Gupta et al. [106]. Our method exploits a latent space for task-oriented exploration: it uses model-predictive control and simulation to choose latent skills which are locally-optimal for completing unseen tasks, then executes those latent skills on the real robot.

Transfer learning Common approaches to simulation-to-reality (sim2real) transfer learning include randomizing the dynamics parameters of the simulation [255], and varying the visual appearance of the environment [289, 318, 150], both of which scale linearly or quadratically the amount of computation needed to learn a transfer policy. Another approach is explicit alignment: given a mapping of common

features between the source and target domains, domain-invariant state representations [322], or priors on the relevance of input features [174], can further improve transferability.

The method we introduce in this work is orthogonal to these existing sim2real approaches. Instead of varying or adapting the simulation parameters, we approach the transfer learning problem from a control perspective. We aim to learn a policy that can accomplish a diverse set of composable skills in simulation, so that it can generalize to many tasks when applied in the real world. Our goal is to reduce the training effort needed on the real system by leverage the library of skills learned previously in the simulator, and subsequently only learning a high-level composing policy that leverages this embedding space to accomplish complex behaviors. As we show in our experiments, this framework leads to increased sample efficiency compared to typical reinforcement learning approaches, which need to retrain the entire policy to accomplish a new composition of skills. Ultimately, our approach can be combined with other sim2real methods, such as domain randomization or domain adaption of the simulator, to further improve the robustness of the transfer learning process.

Other strategies, such as that of Barrett, Taylor, and Stone [16] reuse models trained in simulation to make sim-to-real transfer more efficient. In contrast to our approach, however, this work requires an explicit pre-defined mapping between seen and unseen tasks. Yu et al. [351] learn control policies in a physical simulator and additionally implements online system identification to obtain a dynamics model that is used by the policy to adapt in the real world. Similarly, Preiss, Hausman, and Sukhatme [264] use a latent space for system identification to help specialize a policy to different domains. Sæmundsson, Hofmann, and Deisenroth [290] and Clavera et al. [49] use meta-learning and learned representations to generalize from pre-trained seen tasks to unseen tasks. These approaches, however, require that the unseen tasks to be very similar to the pre-trained tasks, and is few-shot rather than zero-shot. Our method is zero-shot with respect to real environment samples, and can be used to learn unseen tasks that are significantly

out-of-distribution. In addition, our method can be used for composing learned skills in the time domain to achieve unseen tasks that are more complex than the underlying pre-trained task set.

Reinforcement learning with MPC Using reinforcement learning with model-predictive control (MPC) [300] has been explored in the past. Kamthe and Deisenroth [157] proposed using MPC to increase the data efficiency of reinforcement algorithms by training probabilistic transition models for planning. Zhang et al. [354] employ MPC to generate training data for a neural network policy. In our work, we take a different approach by exploiting our learned latent space and simulation directly to find policies for novel tasks online.

Relationship to meta-learning Our work is related to meta-learning methods [79, 281, 272], which seek to learn a single shared policy that can easily generalize to all skills in a task distribution. In this work, we learn an explicit latent space of robot skills that allows us to interpolate and sequence skills. Similarly, unlike recurrent meta-learning methods [66, 222], which implicitly address sequencing of a family of sub-skills to achieve goals, our method addresses generalization of single skills while providing an explicit representation of the relationship between skills. We show that this explicit representation allows us to combine our method with many algorithms for robot autonomy, such as optimal control, search-based planning, and manual programming, in addition to learning-based methods. Furthermore, our method can be used to augment most existing reinforcement learning algorithms, rather than requiring the formulation of an entirely new family of algorithms to achieve its goals.

Simultaneous work The MPC-based experiments in this work are closely related to simultaneous work performed by Co-Reyes et al. [278]. Whereas our method learns an explicit skill representations using pre-chosen skills identified by a known ID, Co-Reyes et al. [278] learn an implicit skill representation by clustering trajectories of states and rewards in a latent space. Furthermore, we focus on MPC-based

planning in the latent space to achieve robotic tasks learned online with a real robot, while their analysis focuses on the machine learning behind this family of methods and uses simulation experiments.

2.1.3 Technical Approach

Our work synthesizes two recent methods in deep RL–pre-training in simulation and learning composable motion policies–to make deep reinforcement learning more practical for real robots. Our strategy is to split the learning process into a two-level hierarchy (Fig. 2.2), with low-level skill policies and their latent space parameterization learned in simulation, and high-level task policies learned or planned either in simulation or on the real robot, using the imperfectly-transferred low-level skills.

2.1.3.1 Preliminaries

Similarly to single-task reinforcement learning, our objective is to learn a policy $\pi_{\theta,\phi}$, a function approximator parameterized by θ and ϕ, which maximizes the expected total γ-discounted reward in a Markov Decision Process (MDP) with state space $s \in \mathcal{S}$ and action space $a \in \mathcal{A}$. In our multi-task RL setting, we characterize a set of low-level tasks (primitive skills) with task IDs $\mathcal{T} = \{1, \ldots, N\}$, and accompanying, per-task reward functions $r^t(s, a)$. We then define our overall policy $\pi_{\theta,\phi}(s, t) = p_{\theta,\phi}(a|s, t)$, as conditioned on this task context variable $t \in \mathcal{T}$ in addition to the state $s \in \mathcal{S}$. Since this is a multi-task reinforcement learning setting, the objective changes to maximize the expected total discounted reward averaged over all the tasks T.

2.1.3.2 Overview

To introduce our skill embedding method, we further decompose the overall policy into two components: a skill embedding distribution $p_\phi(t) = p_\phi(z|t)$, parameterized by ϕ, mapping task t to a distribution of latent vectors z, and a latent-conditioned policy $\pi_\theta(s, z) = p_\theta(a|s, z)$.

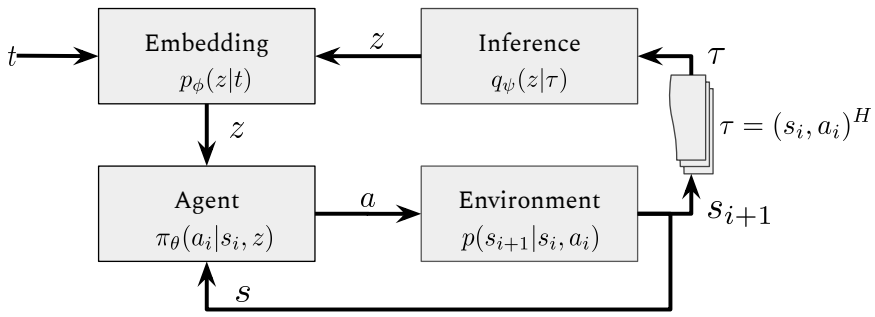


Figure 2.1: Block diagram of proposed architecture for joint learning of a latent space of robotic skills and policies for those skills.

Note that the true skill identity t is hidden from the policy behind the embedding distribution p_ϕ. This forces the policy to learn high-reward primitives that are capable of interpreting many different z's, as opposed to a single task $t \in \mathcal{T}$. In particular, we optimize the embedding distribution p_ϕ to be of high-entropy given the task, resulting in one-to-many mapping between the task t and its latent representation z.

To aid in learning the embedding distribution, we introduce an inference function $q_\psi(\tau) = p(z|\tau)$ which, given a state-action trajectory window $\tau = (s_i, a_i)^H$ of length H, predicts the latent vector z which produced the trajectory. We sample z from $p_\phi(z|t)$ once at the beginning of the rollout, then fed to the low-level skill policy $\pi_\theta(s, z)$), ensuring that all time steps in τ are correlated with the same z. The goal of the additional inference network is to encourage diversity of solutions for a specific task. In particular, we use it to formulate an RL objective that encourages the policy to produce a different trajectory for different latents z. We achieve this goal by augmenting the reward function as discussed in the next section.

This parameterization allows us to separate the representation of skills p_ϕ from their controllers $\pi_\theta(s, z)$. This separation enables our algorithm to explore and manipulate the skill space as a set of abstract vectors representing skill variations. In Sec. 2.1.6.1, we show how we can use simulation to predict the performance of those skill variations on new tasks. Once equipped with these two tools – a representation of learned robot skills, and a model for predicting those skills' outcomes given a world state – we can apply many

proven and efficient robotics algorithms to the high-level representation while still using learning-based methods for low-level control.

2.1.3.3 Skill Embedding Learning Algorithm

The derivation of our method is based on entropy-regularized RL, where there is an additional policy entropy reward term that encourages diverse solutions to a given task, i.e.:

$$\max_{\pi} \mathbb{E}_{\pi, p_0, t\in\mathcal{T}}\Big[\sum_{i=0}^{\infty}\gamma^i\big(r^t(s_i,a_i)+\alpha\mathbb{H}[\pi_\theta(a_i|s_i,t)]\big)\;\Big|\; a_i\sim\pi(\cdot|s,t), s_{i+1}\sim p(s_{i+1}|a_i,s_i)\Big],$$

where $p_0(s_0)$ is the initial state distribution and α is a reward weighting term.

As shown in Hausman et al. [113], the policy entropy part of the objective can be lower-bounded using Jensen's inequality for the latent policy class as following:

$$\begin{aligned}\mathbb{H}[\pi_\theta(a_i|s_i,t)] &= \mathbb{E}_\pi[-\log\pi_\theta(a_i|s_i,t)]\\ &\geq -\mathbb{E}_{\pi_\theta(a,z|s,t)}\Big[-\log q_\psi(z|(a_i,s_i)^H)\Big]+\mathbb{H}[p_\phi(z|t)]+\mathbb{E}_{p_\phi(z|t)}\Big[\mathbb{H}[\pi_\theta(a_i|s_i,z)]\Big].\end{aligned}$$

This objective results in an augmented reward function $\hat{r}(\cdot)$ that can be optimized using any parametric reinforcement learning architecture.

$$\mathcal{L}=\max_{\pi}\;\mathbb{E}_{\substack{\pi(a,z|s,t)\\ t\in T}}\left[\sum_{i=0}^{\infty}\gamma^i\hat{r}(s_i,a_i,z,t)\right],$$

where

$$\hat{r}(s_i, a_i, z, t) = r^t(s_i, a_i) + \alpha_1 \mathbb{E}_{t \in T} \left[\mathbb{H} \left(p_\phi(z|t) \right) \right] + \alpha_2 \log q_\psi(z|(s_i, a_i)^H) + \alpha_3 \mathbb{H} \left(\pi_\theta(a_i|s_i, z) \right) \tag{2.1}$$

We describe the resulting reward augmentations that allow the RL algorithm to train our skill embedding framework below:

1. **Embedding entropy** – $\mathbb{E}_{t \in T} \left[\mathbb{H} \left(p_\phi(z|t) \right) \right]$

   We reward the algorithm for choosing an embedding function that produces a wide distribution of latent vectors z when fed a single task one-hot vector t. This allows the latent space to represent many variations on a skill, rather than assigning each skill to just a single latent.

2. **Inference cross-entropy** – $\log q_\psi(z|\tau = (s_i, a_i)^H)$

   We reward the agent for following trajectories which achieve low cross-entropy with the inference distribution. This encourages the policy to produce distinct trajectories for different latent vectors (identifiability). We discuss this component in more detail below.

3. **Policy entropy** – $\mathbb{H} \left(\pi_\theta(a_i|s_i, z) \right)$

   This term encourages the policy to take many different actions in response to the same state s and latent vector z, so that the policy does not collapse to a single solution for each task. This ensures that the policy can represent many different ways of achieving each task.

During training, rather than revealing the task t to the policy, once per rollout we feed the task ID t, encoded as a one-hot vector, through the stochastic embedding function p_ϕ to produce a latent vector z. The same value of z is fed to the policy for the entire rollout, so that all steps in a trajectory are correlated

with the same value of z. During pre-training, we ensure that each pre-training task t is represented equally in each training batch by sampling skills in a round-robin fashion. This prevents the algorithm from overfitting to any one particular skill.

We train the policy and embedding networks using Proximal Policy Optimization [299], though our method may be used by any parametric reinforcement learning algorithm. We use the MuJoCo physics engine [319] to implement our Sawyer robot simulation environments. We represent the policy, embedding, and inference functions using multivariate Gaussian distributions whose mean and diagonal covariance are parameterized by the output of a multi-layer perceptron. The policy and embedding distributions are jointly optimized by the reinforcement learning algorithm, while we train the inference distribution using supervised learning and a simple cross-entropy loss.

2.1.3.4 Skill Embedding Criterion

In order for the learned latent space to be useful for completing unseen tasks, we seek to constrain the policy and embedding distributions to satisfy two important properties:

1. **High Entropy:** Each task ID t should induce a distribution over latent vectors z which is as wide as possible, representing many variations of a single skill.

2. **Identifiability:** Given a state-trajectory window $(s_i, a_i)^H$ produced by the policy, the inference network should be able to predict with high confidence the latent vector z which was fed to the policy to produce that trajectory. This means the policy must act differently in response to different latent vectors.

When applied together, these properties ensure that during training the embedding produces many different latent parameterizations of each skill (high-entropy), while simultaneously ensuring that the policy learns distinct high-reward variations of that skill in response to each latent parameterization (identifiability). This dual constraint, as implemented by the augmented reward function in Eq. 2.1 is the key for

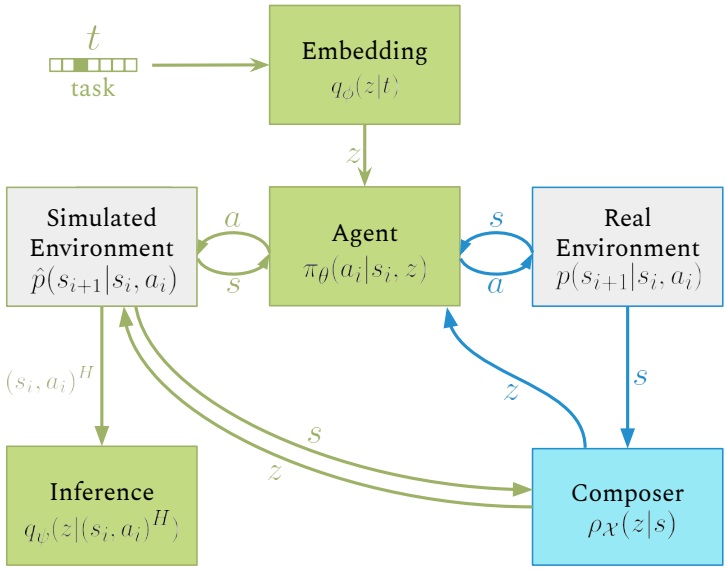


Figure 2.2: Block diagram of proposed architecture for transfer learning. Shared low-level skill components are shown in green. The high level task-specific component is shown in blue.

using model predictive control or other proven algorithms in the latent space, as discussed in Sec. 2.1.5 and Sec. 2.1.6.

2.1.4 Simulation-to-Real Training and Transfer Method

The full robot training and transfer method consists of three stages, and an optional fourth stage.

2.1.4.1 Stage 1: Pre-Training in Simulation while Learning Skill Embeddings

We begin by training in simulation a multi-task policy π_θ for all low-level skills, and a composable parameterization of that library $p_\phi(z|t)$ (i.e. a skill embedding). This stage may be performed using any deep RL algorithm, along with the modified policy architecture and loss function described above. Our implementation uses Proximal Policy Optimization [299] and the MuJoCo physics engine [319].

2.1.4.2 Stage 2: Learning Hierarchical Policies

In the second stage, we learn a high-level "composer" policy, represented in general by a probability distribution $\rho_\chi(z|s)$ over the latent vector z. The composer actuates the low-level policy $\pi_\theta(a|s, z)$ by choosing

among latent skills z. In effect, each latent skill z represents a unique feedback control policy, and the latent space a continuous and explorable space of these controllers, which is supported by a basis set of the pre-training skills $\mathcal{T}$. This hierarchical organization admits our novel approach to transfer learning and adaptation: by freezing the low-level skill policy and embedding functions, and exploring only in the pre-learned latent space to acquire new tasks, we can transfer a multitude of high-level task policies derived from the low-level skills.

This stage can be performed directly on the the real robot or in simulation. As we show in Sec. 2.1.5 and Sec. 2.1.6, composer policies may treat the latent space as either a discrete or continuous space, and may be found using learning, search-based planning, or even manual sequencing and interpolation. To succeed, the composer policy must explore the latent space of pre-learned skills, and learn to exploit the behaviors the low-level policy exhibits when stimulated with different latent vectors. We hypothesize that this is possible because of the richness and diversity of low-level skill variations learned in simulation, which the composer policy can exploit by actuating the skill embedding to choose among these variations.

2.1.4.3 Stage 3: Transfer and Execution

Lastly, we transfer the low-level skill policy, embedding and high-level composer policies to a real robot, and execute the entire system to perform high-level tasks. Alternatively, as discussed in Sec. 2.1.6, we can transfer only the low-level skill policy and skill embedding to the real robot directly, and use an adaptation algorithm to achieve new tasks on the real robot directly, by exploiting the imperfectly-transferred latent skills.

2.1.4.4 Post-smoothing Trajectories

The trajectories generated by the stochastic control policy exhibit jerky behavior, as can be seen even for the basic reaching tasks in simulation (Fig. 2.5 top) and on the real robot (Fig. 2.5 center). While such

noisy motions are crucial during the training of our reinforcement learning framework since it enables the exploration of effective policies – during execution time on the real robot we want to achieve trajectories that require less control effort. To realize this goal, we apply post-smoothing through a finite impulse response (FIR) filter with a 10-step filter window to the action output of the control policy. Each dimension of the action vector $a \sim \pi_\theta$ is filtered through a low-pass filter online during the execution on the real robot.

2.1.5 Characterizing the Behavior of the Latent Skill Space

Our goal in this work is to arrive at a method which will allow us to achieve sample efficiency and adaptation of sim2real policies by learning latent spaces of robot controllers. Prior work (as addressed in Sec. 2.1.2) defines an algorithm for learning such spaces and briefly explores ways of manipulating them, but does not address their sample efficiency and, importantly for our goal, does not characterize the behavior of these spaces, i.e. how the latent controllers behave as we visit points in the space which do not coincide with the pre-chosen basis skills T.

As such, our goal in the following experiments is then to characterize the behavior of these RL-learned latent skill spaces outside of the well-known basis skills, to evaluate the method's suitability for sample-efficient learning and adaptation, and to inform the design of a more general adaptation method, which we describe in Sec. 2.1.6.

2.1.5.1 Experiments

Point Environment Before experimenting on complex robotics problems, we evaluate our approach in a point mass environment. Its low-dimensional state and action spaces, and high interpretability, make this environment our most basic test bed. We use it for verifying the principles of our method and tuning

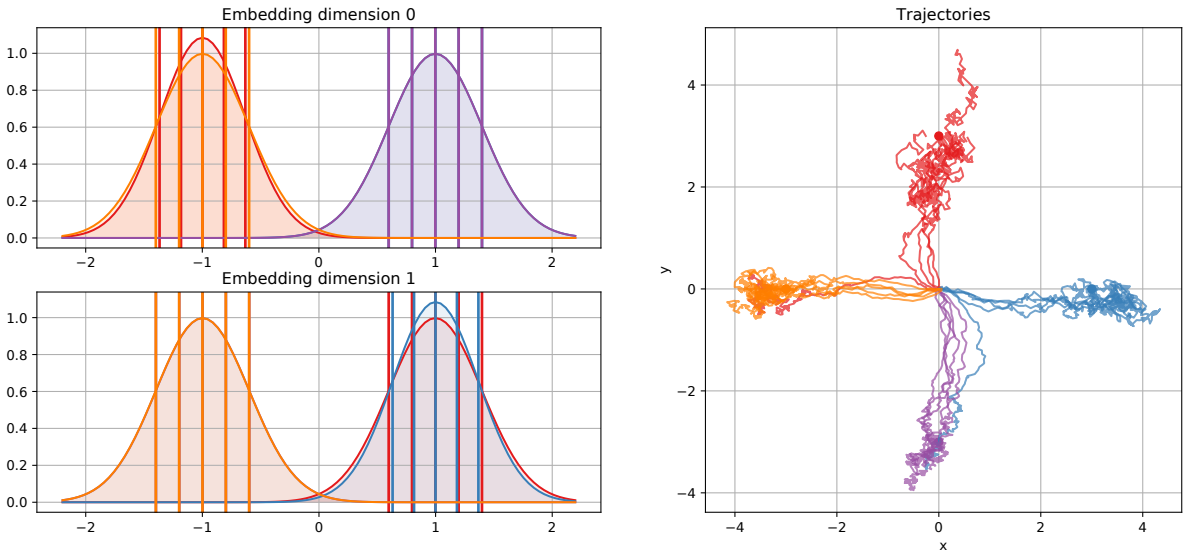


Figure 2.3: Skill embedding distribution which successfully disentangles the four different skills in the point mass environment using the embedding function.

its hyperparameters before we deploy it to more complex experiments. Portrayed in Fig. 2.3 is a multi-task instantiation of this environment with four goals (skills).

At each time step, the policy receives as state the point's position and chooses an instantaneous two-dimensional velocity vector as its action, limited to 0.1 $\frac{\text{unit}}{\text{step}}$. The policy receives a negative reward equal to the distance between the point s and the goal position g_i, i.e. $r_i = -||s - g_i||^2$.

After 15,000 time steps, the embedding network learns a multimodal embedding distribution to represent the four skills (Fig. 2.3). Introducing entropy regularization [113] to the policy alters the trajectories significantly: instead of steering to the goal position in a straight line, the entropy-regularized policy encodes a distribution over possible solutions. Each latent vector produces a different solution. This illustrates that our approach is able to learn multiple distinct solutions for the same skill, and that those solutions are addressable using the latent vector input.

Sawyer Experiment: Reaching Now that we have used the simple point environment to verify the fundamentals of the latent skill learning algorithm, we can begin to characterize the behavior of these

Figure 2.4: Multitask environment in simulation (left) and reality (right) for the reaching experiment. In pre-training, the robot learns a low-level skill for servoing its gripper to each of eight goal positions.

latent spaces for real robotics problems. Our goal in these experiments is to assess whether the method's behavior in simple environments such as the point experiment transitions to the higher-dimensional action and observation spaces of real robots.

We start by transitioning to pre-training using a 6-DOF reaching experiment, which is analogous to the point experiment in the previous section. In the following experiments, we then characterize the behavior of this pre-trained latent skill policy using several composition and interpolation techniques.

We ask the Sawyer robot to move its gripper to within 5 cm of a goal point in 3D space. Like the point experiment, in these experiments the policy is pre-trained with a simple goal-reaching reward function for each basis skill, i.e. $r_i = -||e - g_i||_2^2$, where e is the end-point position of the gripper and g_i is a 3-dimensional Cartesian goal position. Unlike the point environment, the agent's action space is incremental joint position movements of up to ± 0.04 rad for each joint. Joint position control for this action space is implemented by the robot's internal joint position controllers, and these position controllers are allowed to settle before the agent may take another action. The policy receives a state observation with the robot's seven joint angles, plus the Cartesian position of the robot's gripper. Importantly, this agent must control the robot in joint space, not Cartesian space, while avoiding collisions with the environment and itself.

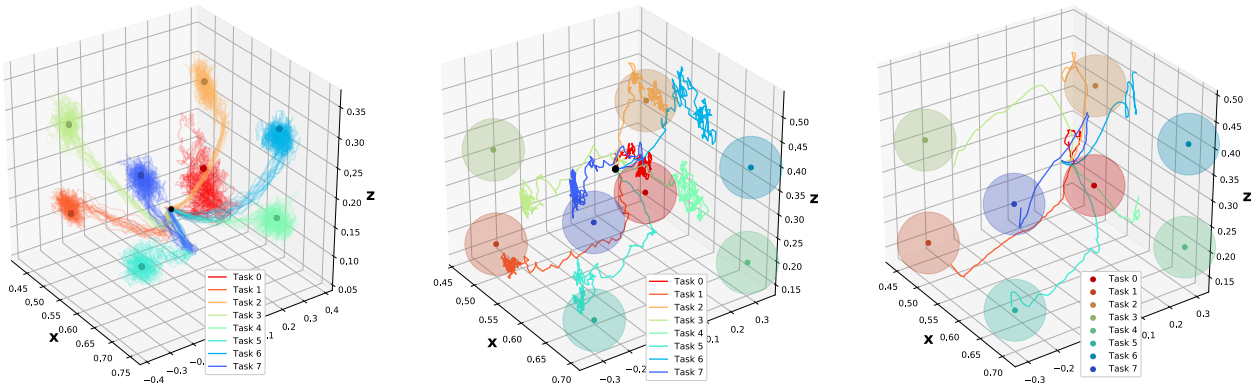


Figure 2.5: Gripper position trajectories for each of the eight skills from the low-level pre-trained reaching policy, after 100 training iterations. *Left:* simulation, *Center:* real robot, *Right:* smoothed trajectories on the real robot. Shaded areas indicate the 5 cm goal regions.

We trained the low-level policy on eight goal positions in simulation, forming a 3D cuboid enclosing a volume in front of the robot (Fig. 2.4). As shown in Fig. 2.5, the embedding policy successfully attains accurate goal reaching performance in simulation (top), and without any fine-tuning generates trajectories that reach within the 5 cm goal radius for most skills on the real robot (center). While the trajectories appear jerky, through online filtering (described in Section 2.1.4.4), the trajectories on the real robot (Fig. 2.5 right) become significantly smoother. Although the FIR filter causes a delay in the control response, we did not observe a reduction in accuracy for the skills we investigated throughout our experiments. The composer policies feed latent vectors to the pre-trained low-level skill policy to achieve high-level tasks such as reaching previously-unvisited points (Fig. 2.6).

Composition Experiments Having verified that the latent skill learning method transitions to controlling a real robot using a sim2real policy, we experiment with several techniques for manipulating the robot in latent skill space. Recall from Section 2.1.3.3 that each vector in the latent skill space represents a unique feedback controller for the robot, so these experiments are exploring a *space of controllers*. As these experiments are descriptive rather than prescriptive, there is no reward function *per se* for each of these experiments, except where otherwise noted.

All Sawyer composition experiments use the same low-level skill policy, pre-trained in simulation. We experimented both with composition methods which directly train new high-level skills on the robot (direct), and with methods which train new high-level skills using a second stage of pre-training in simulation before transfer (sim2real).

Skill interpolation in the latent space (direct) The goal of this experiment is to characterize the "shape" or algebraic behavior of the skill latent space, e.g. "Is combining two or more skills to form a third skill as simple as taking their linear combination, or does the latent space have a non-linear algebra?"

We evaluate the embedding function to obtain the mean latent vector for each of the 8 pre-training skills, then feed linear interpolations of these means to the latent input of the low-level skill policy, transferred directly to the real robot. For a latent pair (z_a, z_b), our experiment feeds z_a for 1s, then $z_i = \lambda_i z_a + (1 - \lambda_i) z_b$ for $\lambda_i \in [0, 1]$ for 1s, and finally z_b for 1s. We observe that the linear interpolation in latent space induces an implicit collision-free motion plan between the two points, despite the fact that pre-training never experienced this state trajectory. In one experiment, we used this method iteratively to control the Sawyer robot to draw a U-shaped path around the workspace (Fig. 2.6).

This experiment demonstrates that, while the algorithm learns a latent space which correctly orients the the relative locations of pre-training skills in latent space (Fig. 2.5), the behavior of the controllers lying in-between these well-known skills does not satisfy a simple linear algebra.

End-to-end learning in the latent space (sim2real) This experiment seeks to assess whether we may use these skill latent spaces to learn quickly using model-free reinforcement learning algorithms like the agent was pre-trained with, e.g. "Can the latent skill z be used as the action space for another model-free reinforcement learning algorithm, allowing us to deploy this method recursively?." Note that this experiment uses a different algorithm (DDPG) from a different family of algorithms (off-policy) from the pre-training step (PPO and on-policy policy gradients, respectively).

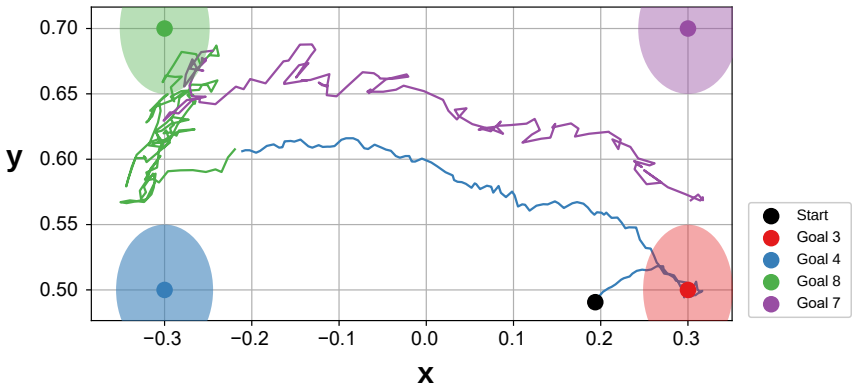


Figure 2.6: Gripper position trajectory while interpolating latents between skills 3, 4, 8 and 7 for the embedded reaching skill policy on the real robot.

Using Deep Deterministic Policy Gradients (DDPG) [191], an off-policy reinforcement learning algorithm, we trained a composer policy to modulate the latent vector to reach a previously-unseen point. The reward function and state spaces for this experiment are identical to the pre-training phase but using an unseen goal, and the action space is simply the latent skill input vector z. We then transferred the composer and low-level policies to the real robot. The policy achieved an end-effector distance error of 5 cm, the threshold for skill completion as defined by our reward function (Fig. 2.7). As off-policy reinforcement learning algorithms consider a task complete as soon as this completion threshold is achieved, DDPG does not learn to get any closer to the goal.

This experiment demonstrates that the latent skill space can still be used with model-free reinforcement learning algorithms to quickly learn new policies and then transfer them to real robots. In the next experiment, we assess whether we can use even-simpler algorithms to achieve the same effect.

Search-based planning in the latent space (sim2real and direct) Having shown we can efficiently explore the latent skill skill spaces using a fairly complex algorithm (DDPG), our goal in this experiment is to assess whether that complexity is warranted, e.g. "Given a latent skill space, can we use a very simple algorithm to achieve sample-efficient adaptation?.

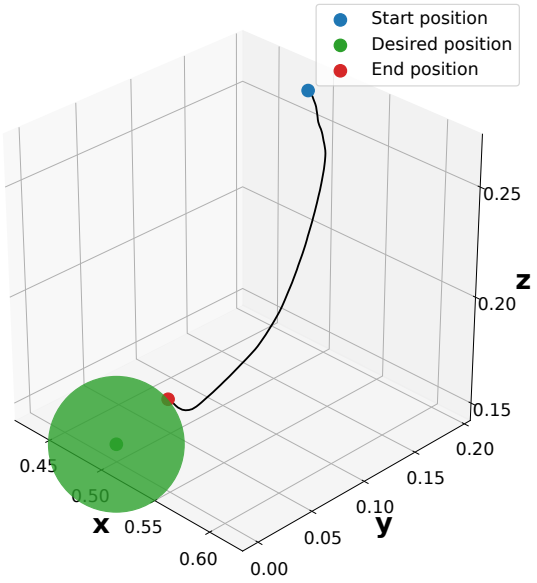


Figure 2.7: End-effector position trajectory for composed reaching policy, trained with DDPG to reach an unseen point (green). The green ball around the desired position visualizes the goal radius of 5 cm.

We used Uniform Cost Search [286, Chapter II] (UCS) in the latent space to find a motion plan (i.e. sequence of latent vectors) for moving the robot's gripper along a triangular trajectory. UCS is a specialized form of Dijkstra's algorithm that finds the shortest path between a single start and goal node in a graph, and is perhaps the simplest general graph-based planning algorithm imaginable. Our search space treats the latent vector corresponding to each skill ID as a discrete option. We execute a plan by feeding each latent in in the sequence to the low-level policy for ≈1s, during which the low-level policy executes in closed-loop. All joint position control commands are processed by the robot's on-board controllers as described before, but we do not necessarily allow each command to settle before issuing a new one. The cost (reward) function used for this task is identical to the one from pre-training, but using an unseen goal location.

In simulation, this strategy found a plan for tracing a triangle in less than 1min, and that plan successfully transferred to the real robot (Fig. 2.8). We replicated this experiment directly on the real robot, with no intermediate simulation stage. It took 24 min of real robot execution time to find a motion plan for the triangle tracing task.

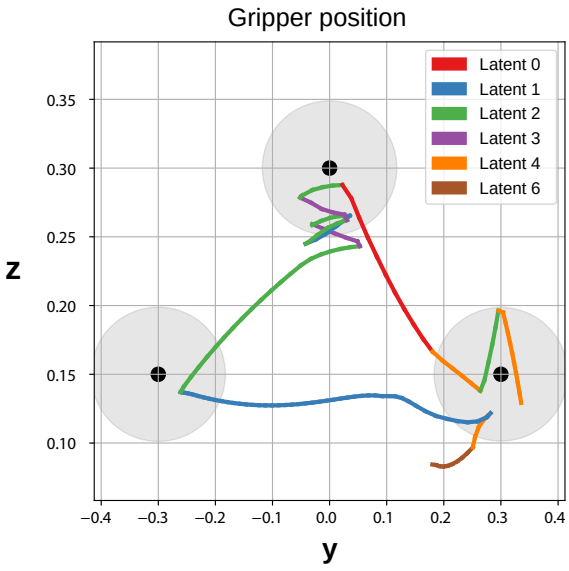


Figure 2.8: Gripper position trajectory reaching points (black) in a triangle within a radius of 5 cm (grey) composed by search-based planning in the latent space.

This experiment demonstrates that learning and then exploring in the latent skill space dramatically increases sample efficiency of acquiring a new skill, and that we can achieve this sample efficiency using a very simple algorithm. Training this policy without the latent space on a real robot would require an intractable number of trials.

Sawyer Experiment: Box Pushing Our Sawyer Reacher experiments demonstrated that the fundamentals of the skill embedding learning and transfer method can be applied to real robots and algorithms for sample-efficient learning and adaptation, but they only assessed the method's performance using a free-space motion task, which is comparatively easy to model in simulation. In these experiments, we seek to characterize the behavior of the method on a skill which is intractable to perfectly model in simulation, because it involves robot-object contact forces and object-object friction.

The goal of the following experiments is to characterize the ability of simple composition strategies to adapt in a sim2real setting, despite the fact that we know that the pre-training policy cannot transfer perfectly from simulation to real training. "Can the skill embedding learning method be used to achieve useful tasks using a real robot, even if the pre-trained skills from simulation do not transfer perfectly?"

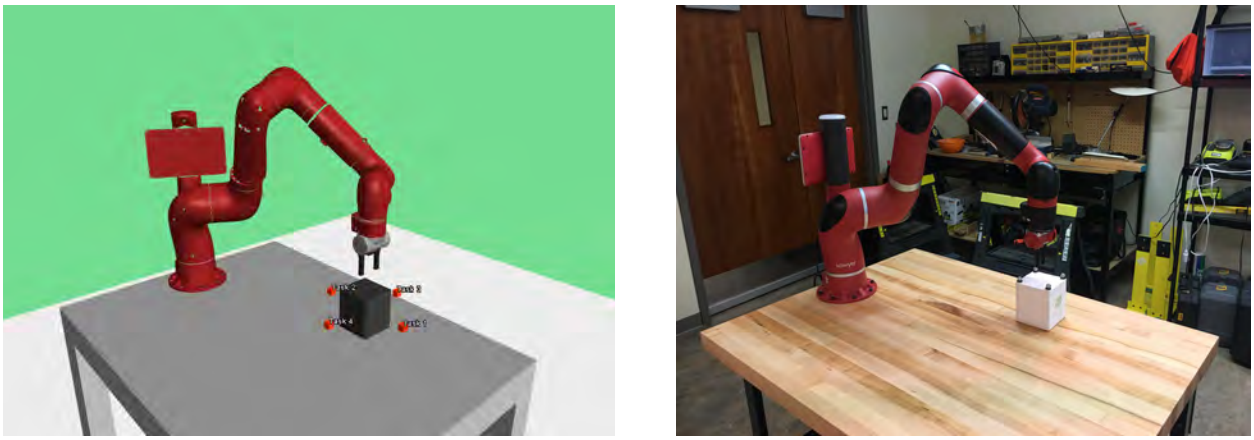


Figure 2.9: Multitask environment in simulation (left) and reality (right) for the box pushing experiment. Four goals are located on the table at a distance of 15 cm from the block's initial position.

Carefully note that the skill embedding learning algorithm can be applied in tandem with any number of other sim2real transfer methods (Sec. 2.1.2), but we intentionally omit these methods as they would confound the results of these experiments.

We ask the Sawyer robot to push a box to a goal location relative to its starting position, as defined by a 2D displacement vector in the table plane. As in the reaching experiments from Sec. 2.1.5.1, the policy receives a state observation with the robot's seven joint angles, but this time the robot's gripper position is replaced with a relative Cartesian position vector between the robot's gripper and the box's centroid. As before, the policy chooses incremental joint movements (up to ± 0.04 rad) as actions, and these actions are executed by the robot's on-board controllers are joint position commands which are allowed to settle between each action. As in the reaching experiment, we use for the basis skills a simple distance reward function defined on the x-y position of the box's centroid, i.e. $r_i = -||b - g_i||_2^2$, where b is the x-y position of the box centroid and g_i is a 2-dimensional Cartesian goal position for the box.

In the real experimental environment, we track the position of the box using motion capture and merge this observation with proprioceptive joint angles from the robot. In this work we do not yet assess the method for end-to-end learning, e.g. from image pixels to joint commands, so we believe the use of motion

capture is representative of many other perception methods used by robots in the field, e.g. reconstruction from 3D point clouds or stereoscopic vision.

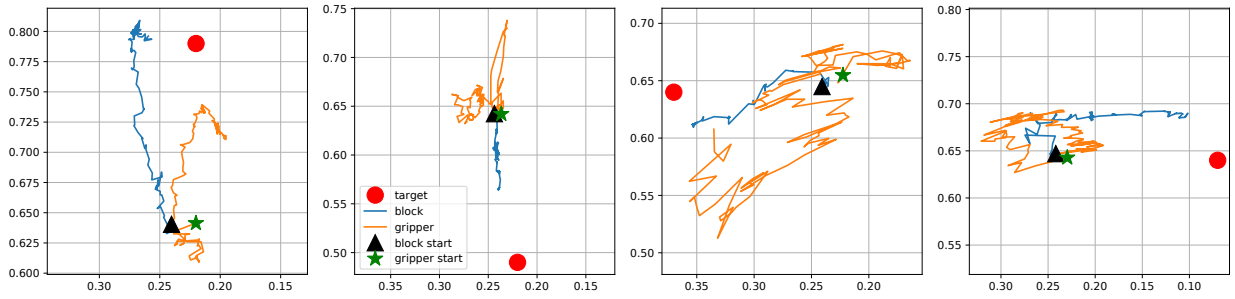


Figure 2.10: Trajectories of the block (blue) and gripper position (orange) created by execution of the embedded skill policy for each of the four pre-trained skills (red dots).

We trained the low-level pushing policy on four goal positions in simulation: up, down, left, and right of the box's starting point (Fig. 2.10). Clearly, this skill does not transfer perfectly from simulation to the real robot.

Composition Experiments Now that we have established that the simulation policy transfer imperfectly to the real robot as expected, we characterize the behavior of the same composition methods from Section 2.1.5, but now in an environment where the policy does not transfer perfectly from simulation to real. We omit many details of the composition methods here for brevity. For detailed explanation and motivation for the composition method experiments explored here, please refer to Section 2.1.5.1.

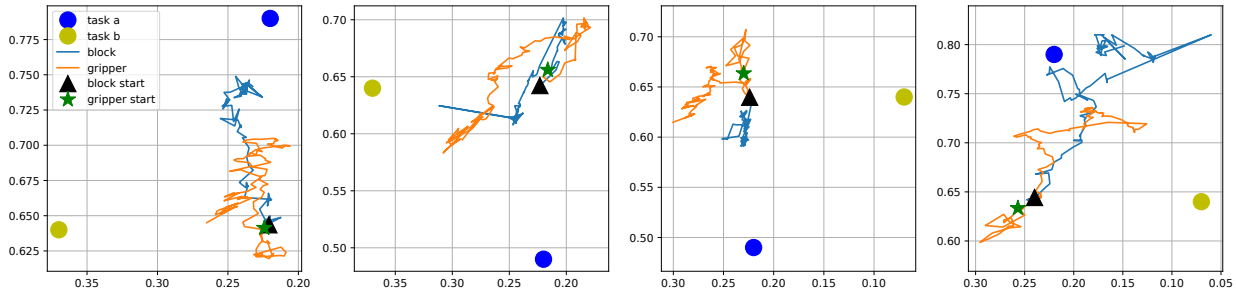


Figure 2.11: Trajectories of the block (blue) and gripper position (orange) created by executing the pre-trained embedded skill policy while feeding the mean latent vector of two neighboring skills. The robot pushes the block to positions between the goal locations of the two pre-trained skills.

Skill interpolation in the latent space (direct) We evaluated the embedding function to obtain the mean latent vector for each of the four pre-trained pushing skills (i.e. up, down, left, and right of start position). We then fed the mean latent of adjacent skills (e.g. $z_{\text{up-left}} = 1/2(z_{\text{up}} + z_{\text{left}})$) while executing the pre-trained policy directly on the robot (Fig. 2.11). The composer policies feed latent vectors to the pre-trained skill policy to push the box to positions which were never seen during training (Fig. 2.11 and Fig. 2.12).

We find that in general this strategy induces the policy to move the block to a position between the two pre-trained skill goals. However, as in the interpolative reaching experiment (Sec. 2.1.5.1, Fig. 2.6), the magnitude and direction of block movement was not easily predictable from the pre-trained goal locations. This behavior was not reliable for half of the synthetic goals we tried, indicating that simple interpolation is not a robust adaptation strategy for these tasks.

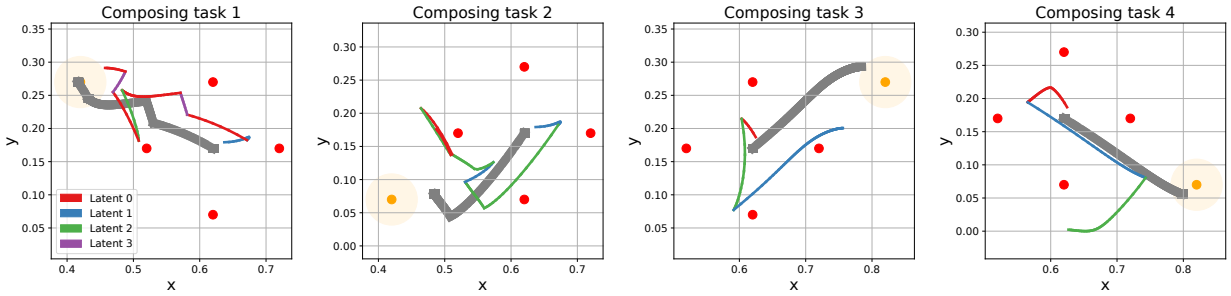


Figure 2.12: Search-based planning in the latent space achieves plans for pushing tasks where the goal position (orange dot) for the block is outside the region of tasks (red dots) on which the low-level skill was trained. Colored line segments trace the path of the robot gripper, and indicate which skill latent was used for that trajectory segment. The block's position is traced by the gray lines.

Search-based planning in the latent space (sim2real) Similar to the search-based composer on the reaching experiment, we used Uniform Cost Search in the latent space to find a motion plan (sequence of latent vectors) for pushing the block to unseen goal locations (Fig. 2.12). The cost (reward) function for this task is identical to that from pre-training, but using an unseen box goal location. We found that search-based planning was able to find a latent-space plan to push the block to any location within the

convex hull formed by the four pre-trained goal locations. Additionally, our planner was able to push blocks to some targets significantly outside this area (up to 20 cm). Unfortunately, we were not able to reliably transfer these composed policies to the robot.

We attribute these failures to transfer partially to the non-uniform geometry of the embedding space, and partially to the difficulty of transferring contact-based motion policies learned in simulation, and discuss these results further in Section 2.1.5.2. In Sec. 2.1.6, we develop a method which can transfer these sim2real composition policies, despite imperfect transfer of skill dynamics from simulation to real.

2.1.5.2 Analysis

The point environment experiments verify the principles of our method, and the single-skill Sawyer experiments demonstrate its applicability to real robotics tasks. Recall that all Sawyer skill policies used only joint space control to actuate the robot, meaning that the skill policies and composer needed to learn how using the robot to achieve task-space goals without colliding the robot with the world or itself.

The Sawyer composition experiments provide the most insight into the potential of latent skill decomposition methods for scaling simulation-to-real transfer in robotics. The method allows us to reduce a complex control problem–joint-space control to achieve task-space objectives–into a simpler one: control in latent skill-space to achieve task-space objectives.

We found that the method performs best on new skills which are interpolations of existing skills. We pre-trained on just eight reaching skills with full end-to-end learning in simulation, and all skills were always trained starting from the same initial position. Despite this narrow initialization, our method learned a latent representation which allowed later algorithms to quickly find policies which reach to virtually any goal inside the manifold of the pre-training goals. Composed policies were also able to induce non-colliding joint-space motion plans between pre-trained goals (Fig. 2.8).

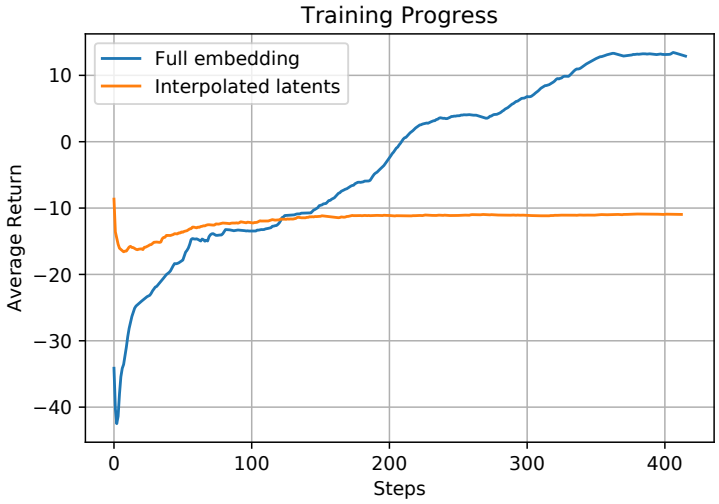


Figure 2.13: Training returns for a composing policy using an embedding trained on eight-goal reacher to reach a new point between the trained goals.

Secondly, a major strength of the method is its ability to combine with a variety of existing, well-characterized algorithms for robotic autonomy. In addition to model-free reinforcement learning, we successfully used manual programming (interpolation) and search-based planning on the latent space to quickly reach both goals and sequences of goals that were unseen during pre-training (Fig. 2.8, 2.11, and 2.12). Interestingly, we found that the latent space is useful for control not only in its continuous form, but also via a discrete approximation formed by the mean latent vectors of the pre-training skills. This enables our method to leverage a large array of efficient discrete planning and optimization algorithms, for sequencing low-level skills to achieve long-horizon, high-level goals.

Conversely, algorithms which operate on full continuous spaces can exploit the continuous latent space. We find that a DDPG-based composer with access only to a discrete latent space (formed from the latent means of eight pre-trained reaching skills and interpolations of those skills) is significantly outperformed by a DDPG composer that leverages the entire embedding space as its action space (Fig. 2.13). This implies that the embedding function contains information on how to achieve skills beyond the instantiations the skill policy was pre-trained on.

The method in its current form has two major challenges.

First is the difficulty of the simulation-to-real transfer problem even in the single-skill domain. We found in the Sawyer box-pushing experiment (Fig. 2.10) that our ability to train transferable policies was limited by our simulation environment's ability to accurately model friction. This is a well-known weakness in physics simulators for robotics. A more subtle challenge is evident in Fig. 2.5, which shows that our reaching policy did not transfer with complete accuracy to the real robot despite it being free-space motion skill. We speculate that this is a consequence of the policy overfitting to the latent input during pre-training in simulation. If the skill latent vector provides all the information the policy needs to execute an open-loop trajectory to reach the goal, it is unlikely to learn closed-loop behavior.

The second major challenge is constraining the properties of the latent space, and reliably training good embedding functions, which we found somewhat unstable and hard to tune. The present algorithm formulation places few constraints on the algebraic and geometric relationships between different skill embeddings. This leads to counterintuitive results, such as the mean of two pushing policies pushing in the expected direction but with unpredictable magnitude (Fig. 2.11), or the latent vector which induces a reach directly between two goals (e.g A and B) actually residing much closer to the latent vector for goal A than for goal B (Fig. 2.6). This lack of constraints also makes it harder for composing algorithms to plan or learn in the latent space.

2.1.6 Using Model Predictive Control for Zero-Shot Sequencing in Skill Latent Space

Now that we have characterized the behavior of the learned skill latent space, we can develop a fully-specified algorithm for exploiting that space to achieve unseen tasks. Our algorithm allows policies learned under our method to adapt to unseen, long-horizon robotics tasks with *no additional real-world samples of those tasks*, which is why we refer to it as a "zero-shot" learning method.

Before we discuss our proposed method, we recall some insights from our experiments in Sec. 2.1.5 to motivate its design. These experiments showed that skill embedding learning methods can be used for

sim2real robot learning, and be used to quickly acquire new skills which are higher-order than their pre-training skills using a variety of methods, from the complex (DDPG) to the very simple (UCS). However, these experiments also showed that the latent skill learning algorithm produces latent spaces which do not necessarily conform to notions of linearity or smoothness (e.g. Fig. 2.6). This makes defining a general procedure for adapting in latent space challenging. The characterization experiments also showed that the latent skill learning algorithm is still susceptible to failure to transfer across the sim2real dynamics gap (e.g. Sec. 2.1.5.1) which affects other RL algorithms.

As we discussed in Sec. 2.1.2, there are many approaches in the literature for addressing both of these problems individually. For instance, one could use e.g. dynamics randomization for the reality gap problem [255], and a meta-reinforcement learning method to enable fast adaptation [79]. Applying one of these augmentations to reinforcement learning adds an additional level of considerable design and sample complexity, and applying more than one at a time produces a proposed algorithm which is *very* complex.

With an eye towards this complexity conundrum, we seek a third, simpler path which recasts the transfer and adaptation problem as one of control rather than learning. We consider these learned latent skills to be partial descriptions of controllers we need to high-level tasks, such as drawing in free space. They are partial because they only define controllers which can achieve part of a task (e.g. transitioning between waypoints) and their implicit internal dynamics models do not represent the real world. "Given partial low-level skill controllers, can we still achieve useful high-level tasks?"

We propose SPC as a method for achieving success in the face of partially-useful controllers. SPC combines a proven robotics algorithm—model-predictive control (MPC)–with pre-trained latent skill policies and online simulation, to predict the behavior of simulation-trained controllers in the real world, and to plan around the irregular shape of the latent skill space. We initialize our online simulations with just a single state observation from the real world at each planning step, to prevent the planning process from

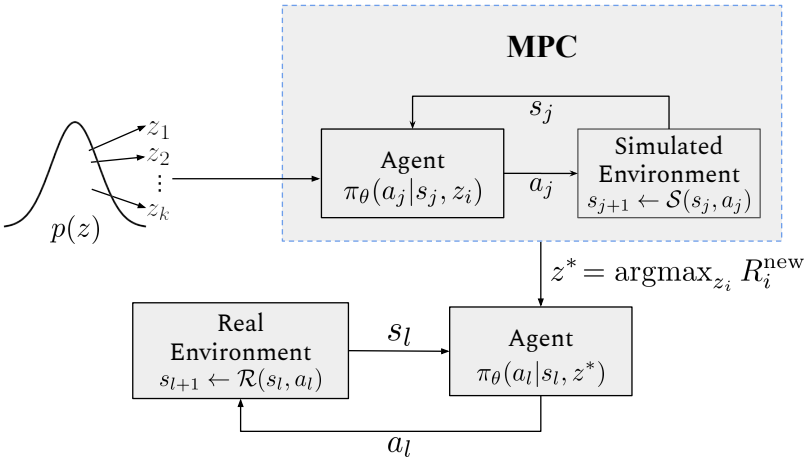


Figure 2.14: Architecture of our simulator-predictive control (SPC) framework where the optimal embedding is found in simulation to be executed in the real environment.

drifting too far from the real robot state. This allows us to utilize these locally-useful pre-trained embedded skill policies to achieve longer-horizon tasks using simple algorithmic components rather than more complex ones.

2.1.6.1 Method

Reusing the Simulation for Foresight In order to evaluate the fitness of pre-trained latent skills for new tasks, we take advantage of the simulation that we used to train the primitive skills. For each adaptation step, we set the state of the simulation environment to the observed state of the real environment. This equips our robot with the ability to predict the behavior of pre-learned latent skills in response to new situations. Since the policy is trained in simulation, we can reuse the simulation from the pre-training step as a tool for approximate foresight when adapting to unseen tasks. This allows us to select a latent skill that is locally-optimal for a task at the current timestep, even if that task was seen not during training.

We show that this scheme allows us to perform zero-shot task execution and composition for families of related tasks. This is in contrast to existing methods, which have mostly focused on direct alignment between simulation and real [327], or data augmentation to generalize the policy using brute force [255, 289]. Despite much work on sim2real methods, neither of these approaches has demonstrated the ability to provide the adaptation capability needed for general-purpose robots in the real world. We believe that our

method enables a third path towards adaptation which warrants exploration, as a higher-level complement to these effective low-level approaches.

Algorithm 1 Simulator-Predictive Control (SPC)

Require: Latent-conditioned policy $\pi_\theta(a|s,z)$,
Skill embedding distribution $p_\phi(z|t)$,
Skill distribution prior $p(t)$,
Simulation environment $\mathcal{S}(s'|s,a)$,
Real environment $\mathcal{R}(s'|s,a)$,
New task t^{new} with associated reward function $r^{\text{new}}(s,a)$,
RL discount factor γ,
MPC horizon T,
Real environment horizon N.

while t^{new} is not complete **do**
  Sample $\mathcal{Z} = \{z_1, \ldots, z_k\} \sim p(z) = \mathbb{E}_{t\sim p(t)} p_\phi(z|t)$
  Observe s_{real} from $\mathcal{R}$
  for $z_i \in \mathcal{Z}$ **do**
    Set inital state of $\mathcal{S}$ to s_{real}
    for $j \in \{1, \ldots, T\}$ **do**
      Sample $a_j \sim \pi_\theta(a_j|s_j, z_i)$
      Execute simulation $s_{j+1} \leftarrow \mathcal{S}(s_j, a_j)$
    end for
    Calculate $R_i^{\text{new}} = \sum_{j=0}^{T} \gamma^j r^{\text{new}}(s_j, a_j)$
  end for
  Choose $z^* = \text{argmax}_{z_i} R_i^{\text{new}}$
  for $l \in \{1, \ldots, N\}$ **do**
    Sample $a_l \sim \pi_\theta(a_l|s_l, z^*)$
    Execute real environment $s_{l+1} \leftarrow \mathcal{R}(s_l, a_l)$
  end for
end while

Adaptation Algorithm Formally, we denote the new task t^{new} corresponding to reward function r^{new}, the real environment in which we attempt this task $\mathcal{R}(s'|s,a)$, and the RL discount factor γ. We use the simulation environment $\mathcal{S}(s'|s,a)$, frozen skill embedding $p_\phi(z|t)$, and latent-conditioned primitive skill policy $\pi_\theta(a|s,z)$, all trained as described in Section 2.1.3.3, to apply model-predictive control (MPC) in the latent space as described in Algorithm 1.

We first sample k candidate latents $\mathcal{Z} = \{z_1, \ldots, z_k\}$ according to $p(z) = \mathbb{E}_{t\sim p(t)} p_\phi(z|t)$. We observe the state s_{real} in the real environment $\mathcal{R}$.

For each candidate latent z_i, we set the initial state of the simulation $\mathcal{S}$ to s_{real}. For a horizon of T time steps, we sample the frozen policy π_θ, conditioned on the candidate latent $a_j \sim \pi_\theta(a_j|s_j, z_i)$, and execute the actions a_j in the simulation environment $\mathcal{S}$, yielding the total discounted reward $R_i^{\text{new}} = \sum_{j=0}^{T} \gamma^j r^{\text{new}}(s_j, a_j)$ for each candidate latent. We then choose the candidate latent acquiring the highest reward $z^* = \text{argmax}_i R_i^{\text{new}}$, and use it to condition and sample the frozen policy $a_l \sim \pi_\theta(a_j|s_j, z^*)$ to control the real environment $\mathcal{R}$ for a horizon of $N < T$ timesteps.

We repeat this MPC process to choose and execute new latent skills in sequence, until the new task has been achieved. Owing its novel use of online simulation, we call our algorithm *Simulator-Predictive Control* (SPC).

The choice of MPC horizon T has a significant effect on the performance of our approach. Since our latent variable encodes a skill which only partially completes the task, executing a single skill for too long unnecessarily penalizes a locally-useful skill for not being globally optimal. Hence, we set the MPC horizon T to not more than twice the number of steps N that a latent is actuated in the real environment.

2.1.6.2 Experiments

Now that we have defined our new SPC method, we seek to assess its ability to simultaneously enable zero-shot sim2real transfer and adaptation. To do this, we use more-elaborate versions of the composition experiments from Sec. 2.1.5. Unlike the experiments from the previous section, the goals of these experiments is *prescriptive*, i.e. we have a particular behavior in mind and seek to assess whether our new SPC method can be used to achieve this behavior. Recall once more that the pre-trained policies are never trained on the tasks in these experiments, and that all of these experiments are in a zero-shot setting with respect to the real robot, i.e. the new task is to be performed by the method with no on-robot training time.

We evaluate our SPC approach by completing two sequencing tasks on a Sawyer robot: drawing a sequence of points and pushing a box along a sequential path, which are analogous to their matching experiments in Sec. 2.1.5.

For each of the experiments, the robot must complete an overall task by sequencing skills learned during the embedding learning process. Sequencing skills poses a challenge to conventional RL algorithms due to the sparsity of rewards in sequencing tasks [5]. Because the agent only receives a reward for completing several correct complex actions in a row, exploration under these sequencing tasks is very difficult for conventional RL algorithms. By reusing the skills we have consolidated in the embedding space, we show a high-level controller can effectively compose these skills to achieve long-horizon tasks.

In these experiments, the pre-training reward functions are identical to those from Sec. 2.1.5, and the adaptation-time (i.e. sequencing) reward function uses a simple switching scheme: for each waypoint, the reward function is the simple goal-distance reward function from pre-trainng. Once the robot achieves a waypoint, the reward function *switches* to the goal-distance reward for the next waypoint. For instance, for the drawing experniments below, the reward function while the robot moves from the origin to the first waypoint g_1 is $r = -||e - g_1||_2^2$ where e is the gripper end-point position and g_1 is the 3-dimensional position of the first waypoint, once it reaches g_1 the reward function is $r = -||e - g_2||_2^2$ where g_2 is the 3-dimensional position of the second waypoint, etc.

Note that in the results below, the high-entropy (i.e. semi-random) skill policy forces the gripper to move along oscillating paths between points, rather than straight lines. While the sampled actions from the stochastic control policy result in trajectories that exhibit significant jerkiness, to smooth the motions, the same filtering technique we show in Section 2.1.4.4 can be applied to the output obtained from SPC.

Sawyer: Drawing a Sequence of Points The goal of this experiment is to assess whether we can use SPC to achieve a new higher-order task with zero-shot adaptation, by using SPC with our skill embedding

learning algorithm. This tests one hypothesis from Sec. 2.1.1, which is that we can use skill embedding learning algorithms to quickly achieve more complex tasks than those we pre-trained on.

We ask the Sawyer robot to move its end-effector to a sequence of points in 3D space using joint-space control to draw a shape. The policy receives as observation the robot's seven joint angles as well as the Cartesian position of the robot's gripper, and outputs incremental joint positions (up to 0.04 rads) as actions. We use the Euclidean distance between the gripper position and the current target as the cost function for the pre-trained skills.

We pre-trained the policy and the embedding networks on eight goal positions in simulation, forming a 3D rectangular prism enclosing the workspace. This induces a reaching skill set in latent space. We then use our method to choose a sequence of latent vectors which allow the robot to draw an unseen shape. For both simulation and real robot experiments, we attempted two unseen tasks: drawing a rectangle in 3D space (Fig. 2.15 and 2.16) and drawing a triangle in 3D space (Fig. 2.17 and 2.18).

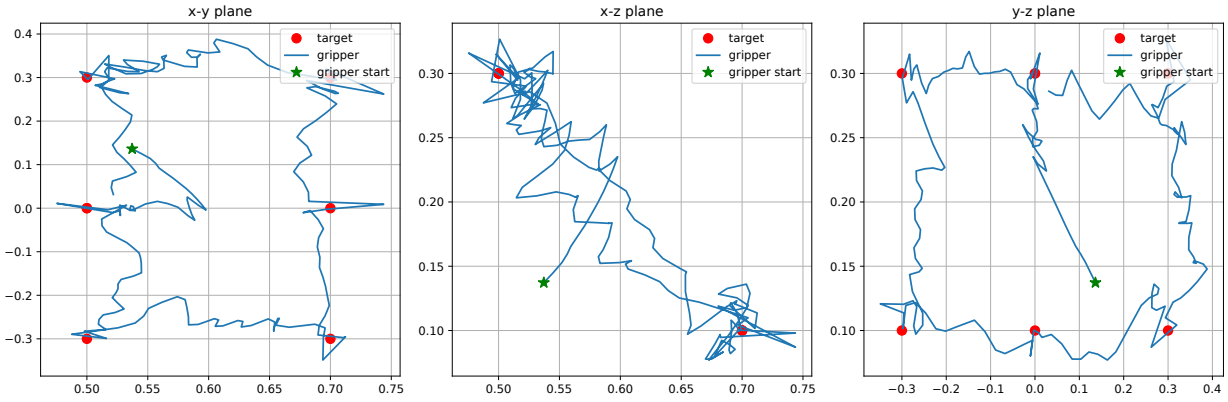


Figure 2.15: Gripper position plots for the unseen rectangle-drawing experiment in simulation. The unseen task is to move the gripper to draw a rectangle, and the pre-trained skill set is reaching in 3D space using joint-space control.

Using SPC and despite imperfect skill policy transfer, the robot successfully draws approximate rectangle and triangle shapes in 3D space. It does this in a single trial with no new on-robot learning time (zero-shot), using latent skill policies which were trained only in simulation on 8 pre-training skills which enclose the workspace, visiting many goals it was never trained on during pre-training. This experiment

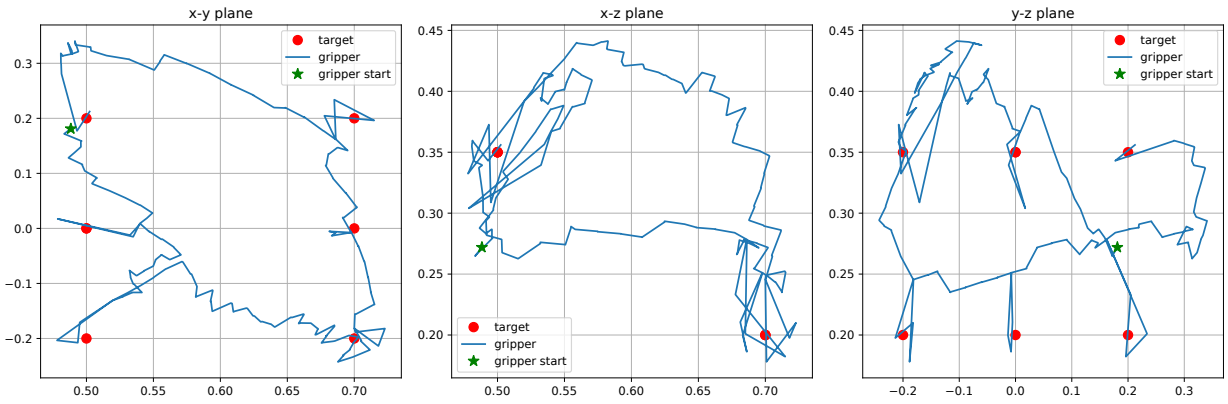


Figure 2.16: Gripper position plots for unseen rectangle-drawing experiment on the real robot. The unseen task is to move the gripper to draw a rectangle, and the pre-trained skill set is reaching in 3D space using joint-space control.

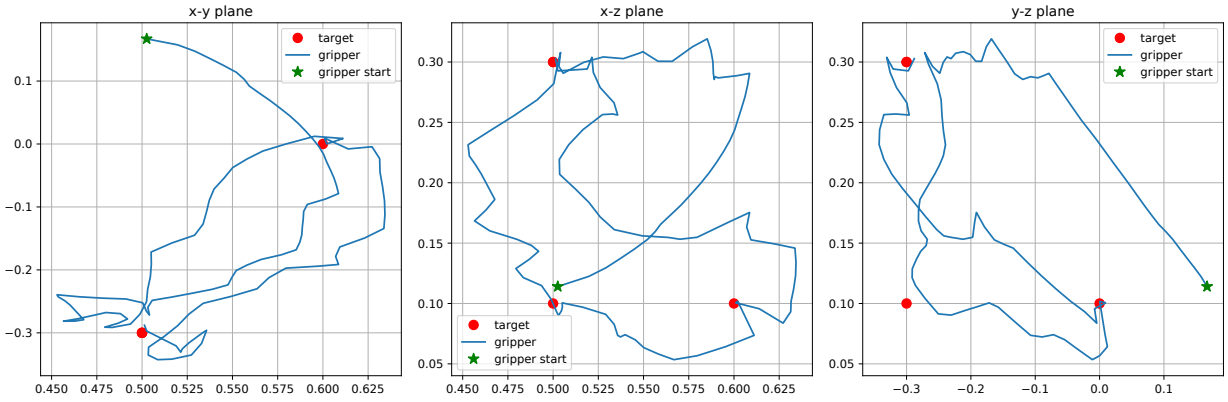


Figure 2.17: Gripper position plots in unseen triangle-drawing experiment in simulation. The unseen task is to move the gripper to draw a triangle, and the pre-trained skill set is reaching in 3D space using joint-space control.

confirms our hypothesis in Sec. 2.1.1, which is that latent skill spaces encode partial information on how to complete whole families of tasks, and they can be used with other methods which augment that information to quickly complete never-before-seen higher-order tasks. Below in Sec.2.1.6.3, we show that these sequencing tasks are extremely difficult to learn, even in very simple RL environments.

Sawyer: Pushing the Box through a Sequence of Waypoints Our goal in this experiment is to test whether SPC can adapt sim2real skill embedding policies, even in the face of imperfect dynamics transfer. Recall in Sec. 2.1.5.1 that the box-pushing policy composed in simulation failed to transfer due to dynamics

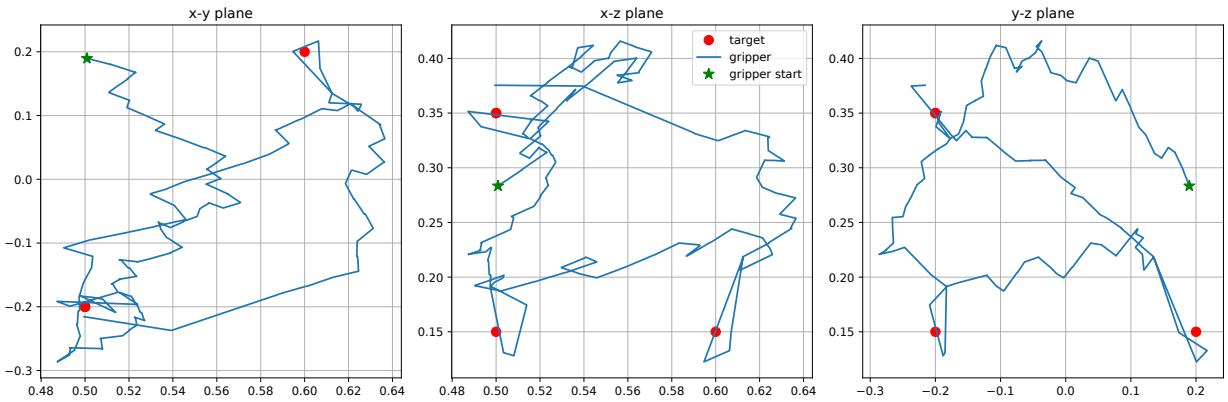


Figure 2.18: Gripper position plots in unseen triangle-drawing experiment on the real robot. The unseen task is to move the gripper to draw a triangle, and the pre-trained skill set is reaching in 3D space using joint-space control.

gaps between the simulation and real robot. In this experiment, we see whether SPC can cross the sim2real dynamics gap and still achieve zero-shot adaptation. This experiment tests a hypothesis we discussed in Sec. 2.1.6, which is that we can use simulation as a tool for *approximate foresight* in sim2real problems. We are also testing whether we can use a proven robotics algorithm (MPC) with experimental RL algorithms, to make a system which can achieve this new capability (zero-shot sim2real transfer and adaptation).

We ask the robot to push a box along a sequence of points in the table plane using task-space control. As in Sec. 2.1.5.1, we use the Euclidean distance between the position of the box and the current target position as the skill reward function, and switch reward functions between waypoints as described above. The policy receives a state observation with the relative position vector between the robot's gripper and the box's centroid, and outputs incremental gripper movements (up to ± 0.03 cm) as actions.

We first pre-train a policy to push the box to one of four goal locations relative to its starting position in simulation. This induces a planar pushing skill set. We trained the low-level multi-task policy with four skills in simulation: 20cm up, down, left, and right of the box starting position. We then use our adaptation algorithm to choose latent vectors online which will push the box through a series of waypoints. In the simulation experiments, we use our method to push the box through three waypoints. In the real robot

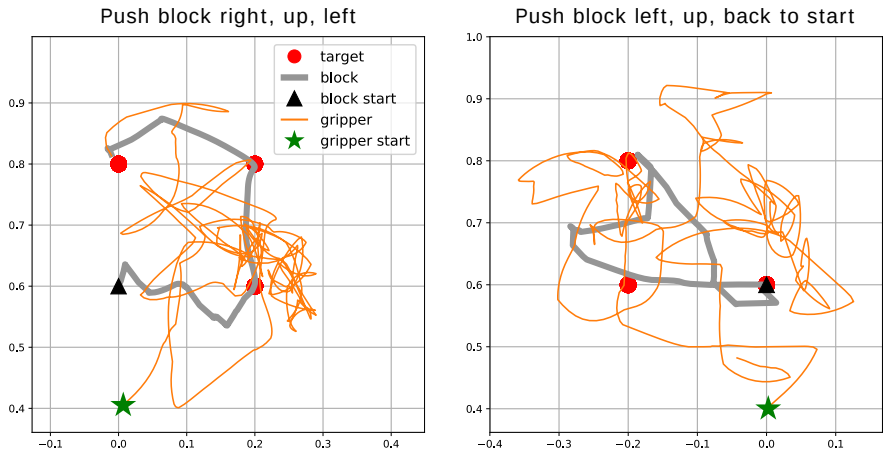


Figure 2.19: Plot of block positions and gripper positions for the box-pushing experiments in simulation using SPC (Algorithm 1). In the first experiment (left), the robot pushes the box to the right, up and then left. In the second experiment (right), the robot pushes the box to the left, then up, and then back to its starting position. In these experiments, the unseen task is to move a box through a series of waypoints, and the pre-trained skillset is pushing a box along a straight line using task-space control.

experiments, we use our method to complete two unseen tasks: pushing up-then-left and pushing left-then-down.

These experiments confirm our hypothesis, which is that the SPC method can be used to augment embedded skill policies to cross a sim2real gap and simultaneously achieve new, higher-order tasks.

2.1.6.3 Results

Sawyer: Drawing a Sequence of Points In the unseen drawing experiments, we sampled $k = 15$ vectors from the skill latent distribution, and for each of them performed an MPC optimization with a horizon of $T = 4$ steps. We then execute the latent with highest reward for $N = 2$ steps on the target robot. In simulation experiments, the Sawyer robot successfully drew a rectangle by sequencing 54 latents (Fig. 2.15) and drew a triangle with 56 latents (Fig. 2.17). In the real robot experiments, the Sawyer Robot successfully completed the unseen rectangle-drawing task by choosing 62 latents (Fig. 2.16) in two minutes of real time and completed the unseen triangle-drawing task by choosing 53 latents (Fig. 2.18) in less than two minutes.

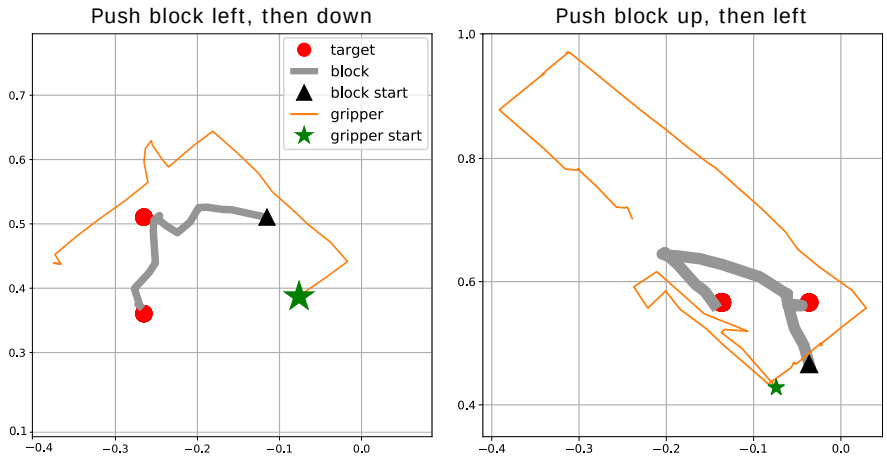


Figure 2.20: Trajectories of the positions of the block and the gripper in the xy-plane for the box-pushing experiments using SPC (Algorithm 1) on the real robot. In these experiments, the unseen task is to move a box through a series of waypoints, and the pre-trained skillset is pushing a box along a straight line using task-space control. In the first experiment (left), the robot pushes the box to the left, and then down. In the second experiment (right), the robot pushes the box up, and then to the left.

We investigate the performance of PPO, a state-of-the-art reinforcement learning algorithm, optimizing a stochastic policy which does not learn a skill embedding function on a sequential point mass task (Fig. 2.21 bottom). In this environment, a two-dimensional point is the state space and the action space is the two-dimensional change in state. If the current waypoint (goal) is reached, the state is reset to the origin from where the next waypoint is to be reached, until all waypoints are reached and the episode ends. In contrast to our earlier sequencing tasks using the composer policy, designing an adequate reward function that encourages the baseline algorithm to learn a policy that can move the point along the four waypoints poses a major challenge.

A naive attempt of rewarding the negative distance between the current state and the next waypoint has the side-effect that the policy keeps making circles around the initial goal point without actually hitting it. After touching a waypoint, the reward will be highly negative before the next waypoint is reached, making it highly unlikely for a conventional algorithm, which maximizes the expected future reward, to ever explore the desired sequencing behavior. Instead, we carefully design the reward function to be

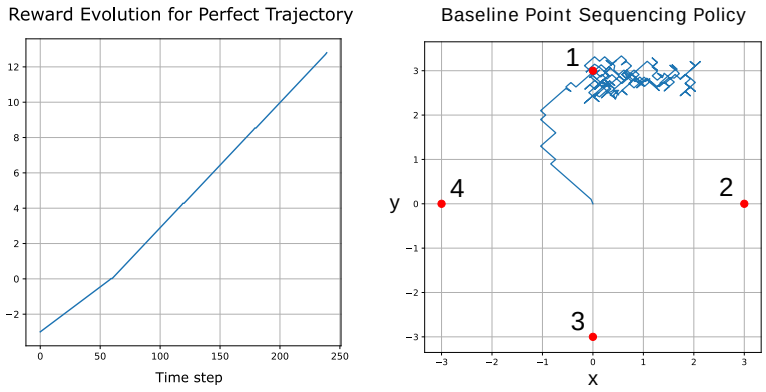


Figure 2.21: *Left:* Evolution of the reward function r_{pointseq} (Eq. 2.2) is monotonic for the perfect trajectory that directly reaches to all goals in sequence. *Right:* Baseline policy that is trained to achieve a sequencing task in a 2D point mass environment, where the objective is to move the two-dimensional point clock-wise between four waypoints (red). Even with the complex hand-engineered reward function given in Eq. 2.2 which provides a monotonically reward landscape as the waypoints are reached, PPO with a policy that is not equipped with a skill embedding function fails to find a successful sequencing behavior.

monotonically increasing even when a waypoint is reached (Fig. 2.21 top). We keep track of the number of waypoints achieved and provide a very high completion bonus ρ:

$$r_{\text{pointseq}}(s) = \#(\text{goals reached}) \cdot \rho - ||s - g_i||_2^2. \tag{2.2}$$

As shown in one of the rollouts in Fig. 2.21 (bottom), despite extensive experimentation with the training set up, our baseline failed to achieve motions that reach to more than a single waypoint.

Sawyer: Pushing a Box along a Sequence of Waypoints In the pusher sequencing experiments, we sample $k = 50$ vectors from the latent distribution. We use an MPC optimization with a simulation horizon of $T = 30$ steps, and execute each chosen latent in the environment for $N = 10$ steps. In simulation experiments (Fig. 2.19), the robot completed the unseen up-left task less than 30 seconds of equivalent real time and the unseen right-up-left task less than 40 seconds of equivalent real time. In the real robot experiments (Fig. 2.20), the robot successfully completed the unseen left-down task by choosing three latents over approximately one minute of real time, and the unseen push up-left task by choosing eight latents in about 1.5 minutes of real time.

2.1.6.4 Analysis

The experimental results show that SPC can learn composable skills and then quickly compose them online to achieve unseen tasks. Our approach is fast in wall clock time because we perform the model prediction in simulation instead of on the real robot. Note that our approach can utilize the continuous space of latent skills, whereas previous search methods only use an artificial discretization of the continuous latent space. In the unseen box-pushing real robot experiment (Fig. 2.20, *Right*), the Sawyer robot pushes the box towards the bottom-right right of the workspace to fix an error it made earlier in the task. This reactive behavior was never explicitly trained during the pre-training phase in simulation. This demonstrates that by exploring in the latent space, our adaptation method successfully composes skills to accomplish tasks even when it encounters situations which were never seen during the pre-training process.

The experiments show that latent skill learning and SPC can adapt to mild sim2real dynamics gaps, however the primary advantage of these methods is sample-efficient adaptation for sim2real robot learning, not dynamics transfer. As we mentioned in Section 2.1.3, a real implementation of this method in the field would likely combine embedded skill learning and adaptation with a dynamics-transfer focused sim2real method to achieve the best transfer performance.

2.1.7 Conclusion

Our experiments illustrate the promise and challenges of applying state-of-the-art deep reinforcement learning to real-world robotics problems. For instance, our policies were able to learn and generalize task-space control and even motion planning skills, starting from joint-space control, with no hand-engineering for those use cases. Simultaneously, the training and transfer process requires careful engineering and some hand-tuning.

We believe this control-oriented perspective provides a promising example of how to combine new learning-based robotics methods with proven algorithms like MPC, by demonstrating how to intentionally

learn robotic policies which are manipulable by those algorithms without sacrificing their best properties, such as expressiveness and the ability to learn from data.

In future work, we plan to use our method as an exploration strategy for learning long-horizon tasks using reinforcement learning on real robots. We will also study how to automatically identify and learn appropriate skill sets for task families from data, and how to intentionally learn skill latent spaces which are amenable to exploration and control. We also look forward to further exploring the relationship between our method and meta-learning techniques, and the combination of our method with techniques for learning representations of the observation space.

Acknowledgements

The authors would like to thank Angel Gonzalez Garcia, Jonathon Shen, and Chang Su for their work on the garage [277] reinforcement learning software toolkit, on which the software for this work was based. Furthermore, we thank Tao Yao and Avnish Narayan for their help with the robot experiments and the implementation of filtering mechanisms. We would also like to thank the members of the Robotics and Embedded Systems and the Computational Learning and Motor Control Laboratories at USC for their help and insightful discussions, and the Robotic Artificial Intelligence and Learning Laboratory at UC Berkeley for their gracious hospitality.

2.2 Differentiable Physics for Optimal Control and System Identification

Intelligent agents need a physical understanding of the world to predict the impact of their actions in the future. While learning-based models of the environment dynamics have contributed to significant improvements in sample efficiency compared to model-free reinforcement learning algorithms, they typically fail to generalize to system states beyond the training data, while often grounding their predictions on non-interpretable latent variables. We introduce Interactive Differentiable Simulation (IDS), a differentiable physics engine, that allows for efficient, accurate inference of physical properties of rigid-body systems. Integrated into deep learning architectures, our model is able to accomplish system identification using visual input, leading to an interpretable model of the world whose parameters have physical meaning. We present experiments showing automatic task-based robot design and parameter estimation for nonlinear dynamical systems by automatically calculating gradients in IDS. When integrated into an adaptive model-predictive control algorithm, our approach exhibits orders of magnitude improvements in sample efficiency over model-free reinforcement learning algorithms on challenging nonlinear control domains.

2.2.1 Introduction

A key ingredient to achieving intelligent behavior is physical understanding that equips agents with the ability to reason about the effects of physical processes by observing their initial conditions. Under the umbrella of intuitive physics, specialized models, such as interaction and graph neural networks, have been proposed to learn dynamics from data to predict the motion of objects over long time horizons. By labelling the training data given to these models by physical quantities, they are able to produce behavior that is conditioned on actual physical parameters, such as masses or friction coefficients, allowing for plausible estimation of physical properties and improved generalizability.

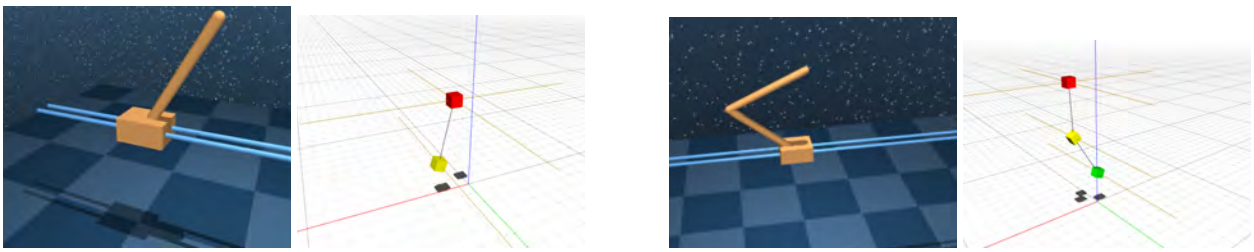


Figure 2.22: Visualizations of the simulated mechanical systems. Single cartpole environment (left). Double cartpole environment (right). Both are actuated by a linear force applied to the cart. The cart is constrained to the rail, but may move infinitely in either direction. Blue backgrounds show environments from DeepMind Control Suite [313] in the MuJoCo [319] physics simulator. Visualizations with white background show Interactive Differentiable Simulation (IDS), our approach.

In this work, we introduce Interactive Differentiable Simulation (IDS), a differentiable physics simulator for rigid body dynamics. Instead of learning every aspect of such dynamics from data, our engine constrains the learning problem to the prediction of a small number of physical parameters that influence the motion and interaction of bodies.

A differentiable physics engine provides many advantages when used as part of a learning process. Physically accurate simulation obeys dynamics laws of real systems, including conservation of energy and momentum. Furthermore, joint constraints are enforced with no room outside of the model for error. The parameters of a physics engine are well-defined and correspond to properties of real systems, including multi-body geometries, masses, and inertia matrices. Learning these parameters provides a significantly interpretable parameter space, and can benefit classical control and estimation algorithms. Further, due to the high inductive bias, model parameters need not be jointly retrained for differing degrees of freedom or a reconfigured dynamics environment.

2.2.2 Related Work

Differentiable physics has recently attracted significant research efforts. Degrave et al. [59] implemented a differentiable physics engine in the automatic differentiation framework Theano. Giftthaler et al. [92] presented a rigid-body-dynamics simulator that allows for the computation of derivatives through code generation via RobCoGen [85]. Similarly, IDS uses Stan Math [39], 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 [40] to facilitate optimal control and inverse kinematics. While such manually derived gradients can be computed efficiently, they are less general than our approach since they can only be used to optimize for a number of hand-engineered quantities. More recently, Koolen and Deits [169] 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 [14] differentiate using the differentiable quadratic program solver OptNet [4]. 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.

Learning dynamics models has a tradition in the field of robotics and control theory. Early works on forward models [230] and locally weighted regression [11] yielded control algorithms that learn from previous experiences. Given gradients through the solution of differential equations has been further leveraged for system identification [267].

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 [292, 189, 199], particularly interaction networks [18, 41, 296, 234, 348] 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

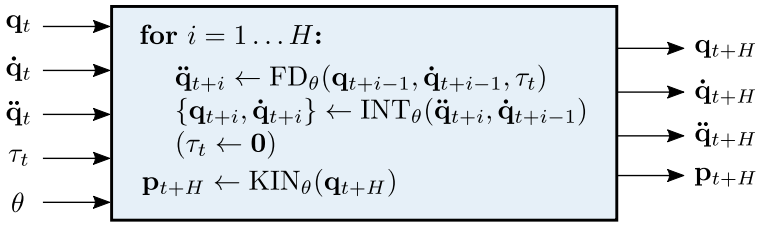


Figure 2.23: IDS deep learning layer with its possible inputs and outputs, unrolling our proposed dynamics model over H time steps to compute future system quantities given current joint coordinates $(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$, joint forces $\boldsymbol{\tau}_t$ and model parameters θ. $\mathrm{FD}(\cdot)$ computes the joint velocities, $\mathrm{INT}(\cdot)$ integrates accelerations and velocities, and $\mathrm{KIN}(\cdot)$ computes the 3D positions of all objects in the system in world coordinates. Depending on which quantities are of interest, only parts of the inputs and outputs are used. In some use cases, the joint forces are set to zero after the first computation to prevent their repeated application to the system. Being entirely differentiable, gradients of the motion of objects w.r.t the input parameters and forces are available to drive inference, design and control algorithms.

been proposed [343, 341, 342, 80, 152]. *Physics-informed learning* imposes a stronger inductive bias on the learning problem to model particular physical processes, such as cosmological structure formation [114] or turbulence models [268]. Deep Lagrangian Networks [205] and Hamiltonian Networks [98] 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 Section 2.2.4.3, has been less explored. While many previous works have shown to adapt simulators to the real world using system identification and state estimation [168, 355], few have shown adaptive model-based control schemes that actively close the feedback loop between the real and the simulated system [276, 73, 42]. Instead of using a simulator, model-based reinforcement learning is a broader field [262], 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 [166, 60, 31] and neural networks [335, 349] have been proposed to learn the dynamics and optimize policies. Bayesian neural networks in particular have been used to learn the dynamics in model-based reinforcement learning approaches [86, 61, 87, 48].

2.2.3 Differentiable Rigid Body Dynamics

In this work, we introduce a physical simulator for rigid-body dynamics. The motion of kinematic chains of multi-body systems can be described using the Newton-Euler equations:

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

Here, $\mathbf{q}$, $\dot{\mathbf{q}}$ and $\ddot{\mathbf{q}}$ are real vectors of generalized* position, velocity and acceleration coordinates, and $\boldsymbol{\tau}$ is a vector of generalized forces. $\mathbf{H}$ is the generalized joint-space inertia matrix (JSIM), also referred to as mass matrix, and depends on $\mathbf{q}$. Coriolis forces, centrifugal forces, gravity and other forces acting on the system, are accounted for by the bias force matrix $\mathbf{C}$ that depends on $\mathbf{q}$ and $\dot{\mathbf{q}}$. Since all bodies are connected via joints, including free-floating bodies which connect to a static world body via free-space joints with seven degrees of freedom (DOF), i.e. 3D position and orientation in quaternion coordinates, their positions and orientations in world coordinates $\mathbf{p}$ are computed by the forward kinematics function $\mathrm{KIN}(\cdot)$ (Fig. 2.23) using the joint angles and the bodies' relative transforms to their respective joints through which they are attached to their parent body.

2.2.3.1 Forward Dynamics

Forward dynamics $\mathrm{FD}(\cdot)$ (cf. Fig. 2.23) is the mapping from positions, velocities and forces to accelerations. We efficiently compute the forward dynamics using the Articulated Body Algorithm (ABA) [76]. Given a descriptive model consisting of joints, bodies, and predecessor/successor relationships, we build a kinematic tree (cf. Fig. 2.24) that specifies the dynamics of the system. In our simulator, bodies comprise

*"Generalized coordinates" sparsely encode only particular degrees of freedom in the kinematic chain such that bodies connected by joints are guaranteed to remain connected.

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.

Longer kinematic chains increase the sensitivity of a mechanism. Small errors in the joint forces due to limits in the numerical precision of floating point numbers can result in large errors of the resulting joint accelerations [76]. The relationship of the sensitivity of a kinematic chain and its joint-space inertia matrix has been studied empirically in [75], where the condition number of $\mathbf{H}$ was found to be in the order of $O(n^4)$ for a mechanism consisting of n bodies. Besides to simulation, this result has implications to control problems in robotics where long chains of rigid bodies should be avoided and an effort be made to model systems as branched kinematic trees to reduce the depth, if possible.

2.2.3.2 Integration

The model in combination with the forward dynamics algorithm form an ordinary differential equation (ODE) that needs to be integrated in time to numerically compute the trajectories of the bodies.

Following the calculation of the accelerations, we implement semi-implicit Euler integration (referred to as $\mathrm{INT}(\cdot)$ in Fig. 2.23) to compute the velocities and positions of the joints and bodies at the current instant t given time step Δ_t:

$$\dot{\mathbf{q}}_t = \dot{\mathbf{q}}_{t-1} + \Delta_t \ddot{\mathbf{q}}_t \qquad\qquad \mathbf{q}_t = \mathbf{q}_{t-1} + \Delta_t \dot{\mathbf{q}}_t$$

Additionally, higher-order integrators, such as the midpoint and Runge-Kutta fourth-order method have been implemented to allow for a more accurate discretization of the rigid-body dynamics ODE. Even more advanced ODE solvers feature adaptive time stepping by analyzing the sensitivity of the system at the current instant. Such approaches keep the tracking error within desired bounds. The sensitivity analysis

is driven by gradients of the dynamics with respect to the state being integrated. Thus, the differentiability of our system can further benefit more accurate simulations and is subject to future research.

Algorithm 2 Forward kinematics

for i = $1..N_B$ **do**

$\boldsymbol{X}_{\mathrm{J}} \leftarrow \textsc{CalculateJoint}(i, \mathbf{q}[i])$

${}^{i}\boldsymbol{X}_{\lambda(i)} \leftarrow \boldsymbol{X}_{\mathrm{J}} \boldsymbol{X}_{T(i)}$

if $\lambda(i) \neq 0$ **then**

${}^{i}\boldsymbol{X}_{0} \leftarrow {}^{i}\boldsymbol{X}_{\lambda(i)} {}^{\lambda(i)}\boldsymbol{X}_{0}$

end if

end for

2.2.3.3 Forward Kinematics

Algorithm 2 computes the forward kinematics of a chain of bodies connected via joints. $\textsc{CalculateJoint}(i, \mathbf{q}[i])$ computes the joint transform of joint i given its coordinates $\mathbf{q}[i]$[†]. We implement different types of joints, such as prismatic, revolute, spherical and free-floating joints, which each specify a different implementation given their generalized coordinates.

While all kinematic trees are assumed to be connected to a fixed base in the basic form of ABA, floating-base systems can be accommodated for by using free-floating joints that allow seven degrees of freedom (quaternion rotation and 3D position) of a body connected to the base.

2.2.3.4 Automatic Differentiation

While the analytical gradients of the rigid-body dynamics algorithms can be derived manually [40], we choose to implement the entire physics engine in the reverse-mode automatic differentiation (AD) framework Stan Math [39]. Automatic differentiation allows us to compute gradients of any quantity involved

[†]To unclutter the notation, $\mathbf{q}[i]$ refers to a subset of the system's overall coordinates $\mathbf{q}$, denoting the coordinates of joint i.

in the simulation of complex systems, opening avenues to state estimation, optimal control and system design. Enabled by low-level optimization, our C++ implementation is designed to lay the foundations for real-time performance on physical robotic systems in the future.

Reverse-mode AD, also known as back-propagation, computes gradients by building a form of stack, which is commonly referred to as "tape", to keep track of the operations and operands during computations that involve variables whose gradients need to be computed. While modern AD frameworks such as Stan Math employ schemes to save memory and computation time [39], we observe that the number of variables on the AD stack grows linearly (Fig. 2.25). Over the course of the simulation of a double pendulum over 1000 time steps, we notice significant differences in the number of AD variables allocated by the various integrators: Euler integration requires the smallest stack with slightly more than ten million variables on the tape after the simulation. Heun's method and second-order Runge-Kutta (RK2) both take up around 17 million variables, while fourth-order Runge-Kutta (RK4) integration requires almost 30 million variables. While the gain in accuracy for larger time steps can potentially offset the increase in memory consumption of higher-order integrators, the problem of linear growth in variables with each simulation step remains. In the future we plan to integrate the approach from Neural ODEs [46] to compute gradients through the solution of a black-box ODE solver by augmenting the system dynamics with adjoints of the states and system parameters. Such adjoint approach and other techniques for sensitivity analysis [267] would allow us to use more accurate ODE solvers, such as adaptive stepping integrators like the Runge-Kutta-Fehlberg method, without the need to resort to automatic differentiation to back-propagate through each operation of the integrator.

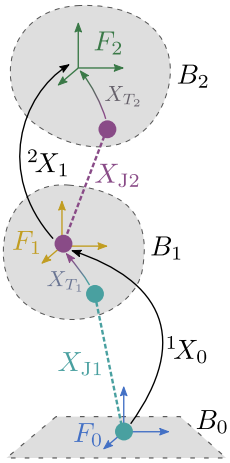


Figure 2.24: Kinematic chain with body frames F_0, F_1 and F_2 consisting of two rigid bodies, B_1 and B_2, connected via joint J2 which determines the joint transform $\boldsymbol{X}_{\text{J2}}$. B_1 is connected to the base body B_0 via joint J1 resulting in a joint transform $\boldsymbol{X}_{\text{J1}}$. Transforms from body frame i to body frame j are denoted as ${}^j\boldsymbol{X}_i$.

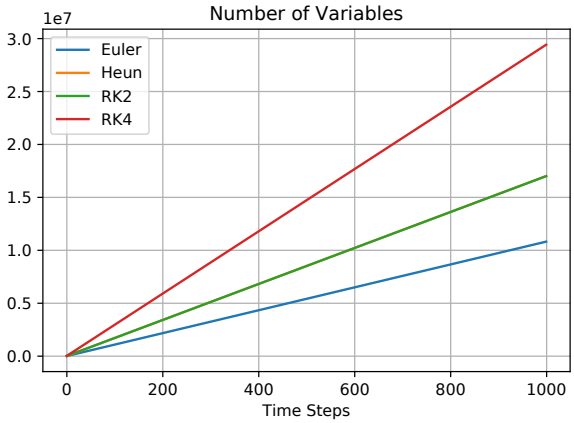


Figure 2.25: Number of variables on the automatic differentiation (AD) stack over the simulation of 1000 time steps of a double cartpole using different integrators. Second-order Runge-Kutta (RK2) and Heun's method require the same stack size.

2.2.4 Experiments

2.2.4.1 System Identification

To demonstrate the capability of our approach to estimate physical parameters of chaotic systems in the real world, we estimate the kinematic parameters of a double pendulum. The dynamics of a compound pendulum are fully determined by the length of each link. Using a VICON motion capture system, we obtain sub-millimeter accurate positional trajectories of the pendulum bobs.

Estimating the length of each link is trivial given the Cartesian coordinates of the pendulum masses. Instead, we estimate the link lengths from a trajectory $[\mathbf{q}_0, \ldots, \mathbf{q}_T]$, where $\mathbf{q}_t$ are the generalized joint positions at time step t. We model the double pendulum using spherical joints whose 3D rotations are represented by quaternions. Given the initial configuration of the pendulum $\mathbf{q}_0 = \mathbf{q}_0^*$ with zero velocity, we minimize

$$\mathcal{L} = \sum_t ||z(f_\theta(\mathbf{q}_{t-1})) - z(\mathbf{q}_t^*)||_2^2, \tag{2.3}$$

where z computes unit heading vectors for given quaternions, and $\mathbf{q}_t^*$ are the joint coordinates of the real pendulum. f_θ denotes our physics simulator, which computes a single step of the system (corresponding to $H = 1$ in Fig. 2.23). The pendulum is parameterized by $\theta \in \mathbb{R}^2$, representing the two link lengths.

Leveraging the differentiability of f_θ, we employ the gradient-based Adam optimizer to estimate the link lengths. We converge after ca. 80 epochs for trajectories 10 steps long, sampled at 25 Hz (cf. Fig. 2.26). Our performance is hampered by the fact that we do not currently simulate joint damping where the pendulum's motion is subject to friction at the connections between the string and the bobs. Accounting for such behavior, in future work, the parameters of such a damping model could be jointly estimated.

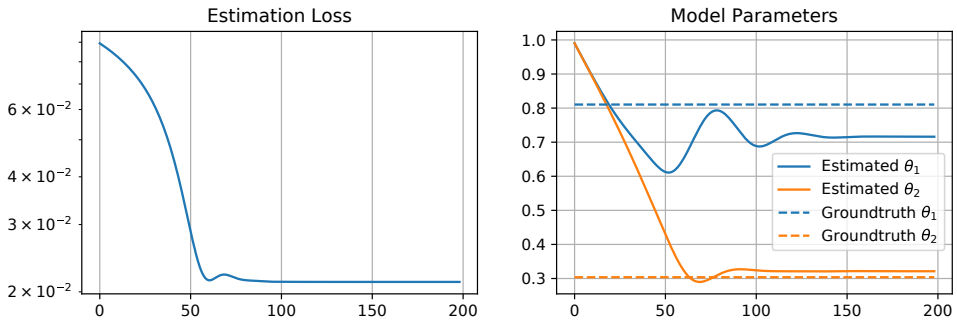


Figure 2.26: Estimation of the two link lengths (in meters) of a real double pendulum modeled via spherical joints.

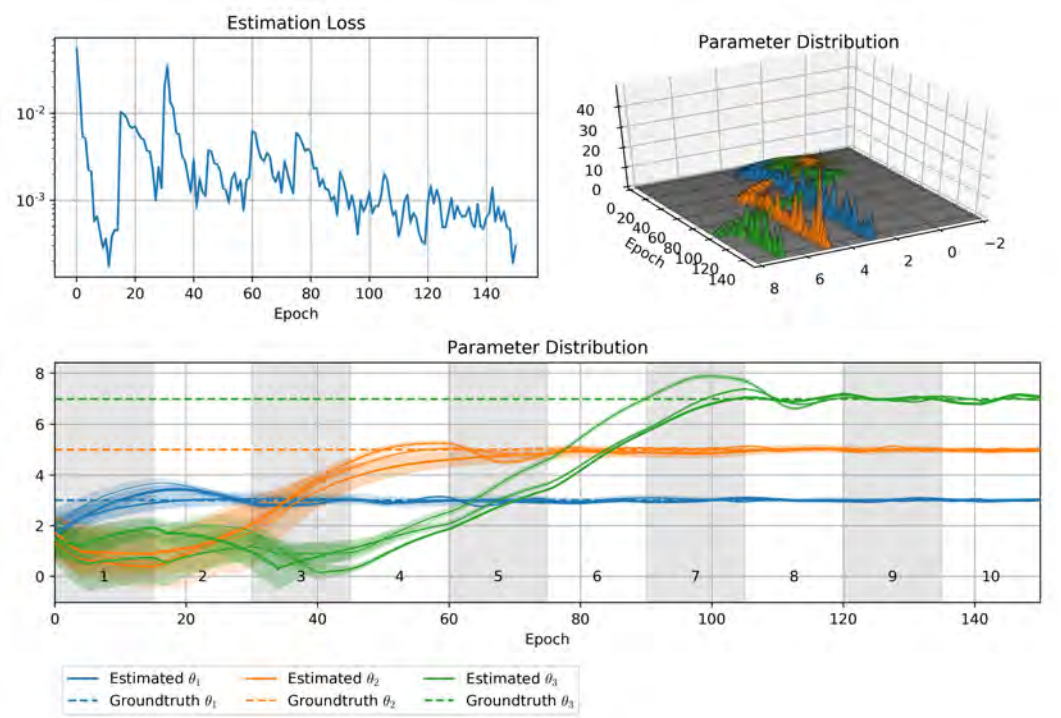


Figure 2.27: Online probabilistic estimation of the lengths of a three-link compound pendulum using a Gaussian Mixture Model. *Top left:* loss evolution, *top right:* 3D visualization of the evolution of the GMM kernels, *bottom:* GMM kernel evolution (means are solid lines, $\mu_i \pm \sigma_i$ are shaded areas). Numbers on vertical stripes indicate number of real world samples observed.

Probabilistic Parameter Estimation Minimizing the sum-of-squares error in Eq. 2.3 by directly adjusting the model parameters θ results in the approximation of the mean of the observed data (cf. Bishop [28]).

While in the previous estimation experiment, the parameters are defined uniquely over a long-enough trajectory, such assumption needs not hold anymore for more complex models, or very few samples from the real system. Furthermore, some models of real-world systems can have coupled variables. For instance, if one wants to estimate both the length and the center of mass of a pendulum given a trajectory of joint angles, the two quantities of interest would mirror each other. Since we are interested in a general estimation approach that is able to capture potential couplings between model parameters and yield results under the presence of noisy observations, we investigate a multi-modal probabilistic estimation method to infer physical parameters.

We consider a three-link compound pendulum parameterized by the link lengths $\theta \in \mathbb{R}^3$. Instead of having access to the entire joint angle trajectory from the real system, we investigate an online estimation process where the parameters need to be inferred as samples $\mathbf{q}_t^*$ are observed successively. We model the distribution over θ by a Gaussian Mixture Model (GMM) consisting of three multivariate Gaussian kernels conditioned on learned variables for mixing coefficient ϕ_i, mean μ_i and variance σ_i of kernel i (cf. Fig. 2.28). To compute a sample θ from a GMM, we need to sample from a categorical distribution $\gamma \sim \text{Discrete}(\phi_1, \phi_2, \phi_3)$ in order to pick the index γ of the Gaussian kernel we eventually sample from. To achieve an end-to-end differentiable stochastic computation graph [297] where we can optimize the GMM parameters μ_i, σ_i, ϕ_i w.r.t the final loss, we approximate this operation using Gumbel Softmax [151, 209]:

$$\gamma_k = \frac{\exp\left((\log \phi_k + G_k)/\lambda\right)}{\sum_{i=1}^{n} \exp\left((\log \phi_i + G_i)/\lambda\right)},$$

where $G_i, G_k \sim$ Gumbel and λ (set to $2/3$) is the temperature defining the "spikiness" of the approximation [209]. Next, we sample from each Gaussian kernel and sum up these samples weighted by γ_i to obtain the physical model parameters θ that we feed into the physics engine which computes a trajectory of joint positions. Minimizing the Huber loss between the observed and generated trajectories via the Adam optimizer, the Gaussian kernels converge after about 120 training epochs (Fig. 2.27) to the true model parameters, while the variance is constantly decreasing.

Inferring Physical Properties from Vision To act autonomously, intelligent agents need to understand the world through high-dimensional sensor inputs, like cameras. We demonstrate that our approach is able to infer the relevant physical parameters of the environment dynamics from these types of high-dimensional observations. We optimize the weights of an autoencoder network [129] trained to predict the future visual state of a dynamical system, with our physics layer serving as the bottleneck layer. In this exemplar scenario, given an image of a three-link compound pendulum simulated in the MuJoCo physics

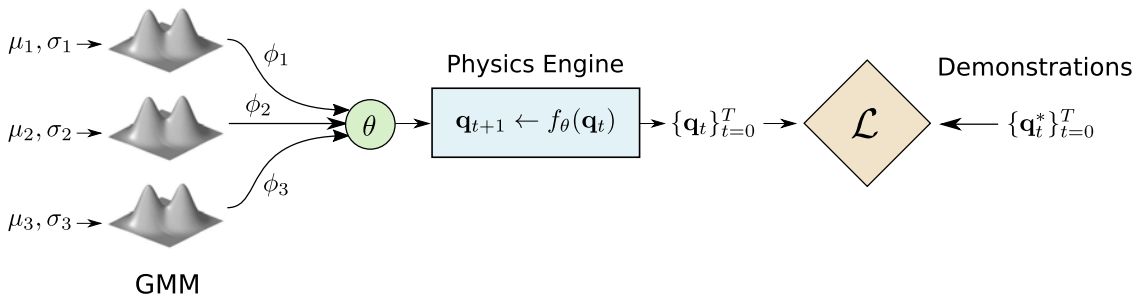


Figure 2.28: Stochastic computation graph [297] of our multimodal estimation pipeline based on a three-component Gaussian Mixture Model (GMM) from which the sampled model parameters θ are given to the differentiable physics engine which in turn deterministically computes trajectories $\{\mathbf{q}\}_{t=0}^T$. The overall loss between the trajectories and demonstrations $\{\mathbf{q}^*\}_{t=0}^T$ is minimized, with gradients flowing end-to-end from the GMM parameters μ_i, σ_i, ϕ_i to $\mathcal{L}$.

simulator [319] at time t, the model is tasked to predict the future rendering of this pendulum H time steps ahead. Compound pendula are known to exhibit chaotic behavior, i.e. given slightly different initial conditions (such as link lengths, starting angles, etc.), the trajectories drift apart significantly. Therefore, IDS must recover the true physical parameters accurately in order to generate valid motions that match the training data well into the future.

We model the encoder f_{enc} and the decoder f_{dec} as neural networks consisting of two 256-unit hidden layers mapping from 100×100 grayscale images to a six-dimensional vector of joint positions $\mathbf{q}$ and velocities $\dot{\mathbf{q}}$, and vice-versa. Inserted between both networks, we place an IDS layer (Fig. 2.23) to forward-simulate the given joint coordinates from time t to time $t + H$, where H is the number of time steps of the prediction horizon. Given that the input data uses a time step $\Delta_t = 0.05\,\text{s}$, the goal is to predict the state of the pendulum $1\,\text{s}$ into the future. While the linear layers of f_{enc} and f_{dec} are parameterized by weights and biases, IDS, referred to as f_{phy}, is conditioned on physical parameters θ_{phy} which, in the case of our compound pendulum, are the lengths of the three links $\{l_0, l_1, l_2\}$. We choose arbitrary values $\{1, 5, 0.5\}$ to initialize these parameters.

Given a dataset D of ground-truth pairs of images (o_t^*, o_{t+H}^*) and ground-truth joint coordinates $(\mathbf{q}_t^*, \mathbf{q}_{t+1}^*, \dot{\mathbf{q}}_t^*, \dot{\mathbf{q}}_{t+1}^*)$, we optimize a triplet loss using the Adam optimizer [164] that jointly trains the individual components of the autoencoder:

$$\begin{aligned}
\mathcal{L}_{\text{enc}} &= ||f_{\text{enc}}(o_t) - [\mathbf{q}_t^*, \dot{\mathbf{q}}_t^*]^T||_2^2 \\
\mathcal{L}_{\text{phy}} &= ||f_{\text{phy}}([\mathbf{q}_t, \dot{\mathbf{q}}_t]) - [\mathbf{q}_{t+H}^*, \dot{\mathbf{q}}_{t+H}^*]^T||_2^2 \\
\mathcal{L}_{\text{dec}} &= ||f_{\text{dec}}([\mathbf{q}_{t+H}, \dot{\mathbf{q}}_{t+H}]) - o_{t+H}^*||_2^2 \\
\mathcal{L}(\cdot; \theta_{\text{enc}}, \theta_{\text{phy}}, \theta_{\text{dec}}) &= \sum_D \mathcal{L}_{\text{enc}}(\cdot; \theta_{\text{enc}}) + \mathcal{L}_{\text{phy}}(\cdot; \theta_{\text{phy}}) + \mathcal{L}_{\text{dec}}(\cdot; \theta_{\text{dec}})
\end{aligned} \tag{2.4}$$

We note that within ca. 200 training iterations the physical parameters θ_{phy} converge to the true parameters of the dynamical system ($l_0 = l_1 = l_2 = 3$), as shown in Fig. 2.30.

As a baseline from the intuitive physics literature, we train a graph neural network model based on [292] on the first 800 frames of a 3-link pendulum motion. When we let the graph network predict 20 time steps into the future from a point after these 800 training samples, it returns false predictions where the pendulum is continuing to swing up, even though such motion would violate physical laws. Such behavior is typical for fully learned models, which mostly achieve accurate predictions within the domain of the training dataset. By contrast, IDS imposes a strong inductive bias, which allows the estimator to make accurate predictions far into the future (Fig. 2.30).

2.2.4.2 Automatic Robot Design

Industrial robotic applications often require a robot to follow a given path in task space which the end-effector should track. 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

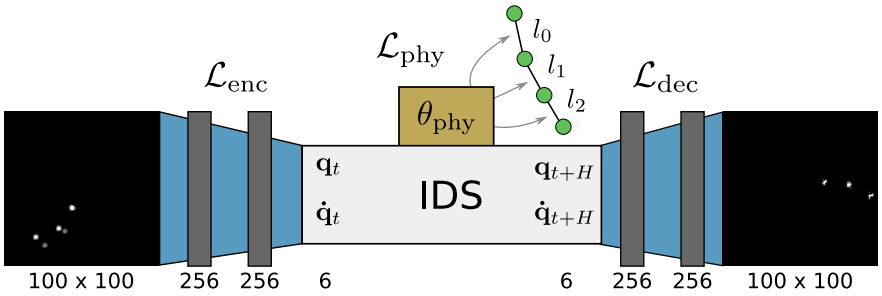


Figure 2.29: Architecture of the autoencoder encoding grayscale images of a three-link pendulum simulated in MuJoCo to joint positions and velocities, $\mathbf{q}$, $\dot{\mathbf{q}}$, respectively, advancing the state of the system by H time steps and producing an output image of the future state the system. The parameters of the encoder, decoder and our physics layer are optimized jointly to minimize the triplet loss from Eq. 2.4.

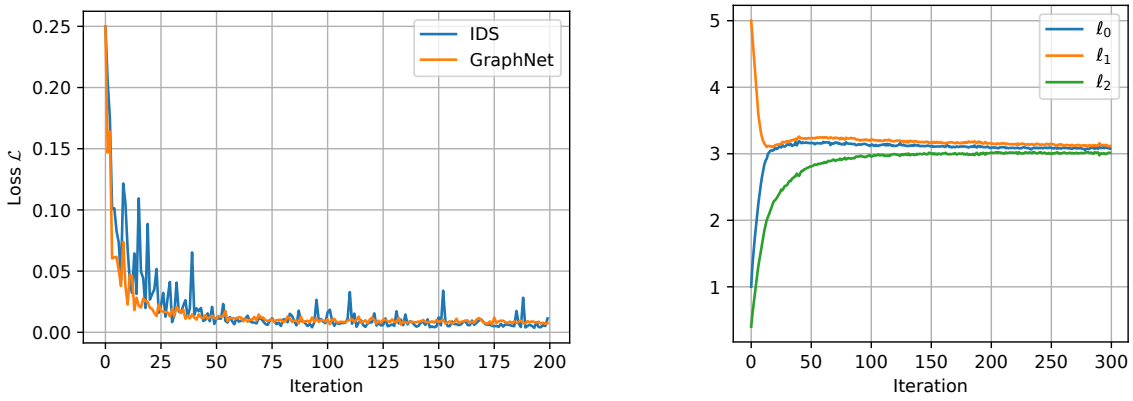


Figure 2.30: *Left:* learning curve of the triple loss $\mathcal{L}$ (Eq. 2.4). An autoencoder trained to predict the future using our physics module trains at a comparable rate to the state-of-the-art intuitive physics approach based on graph neural networks [292] as the predictive bottleneck. *Right:* integrated as the bottleneck in a predictive autoencoder, IDS accurately infers the model parameters θ_{phy} of a compound pendulum, which represent the three link lengths.

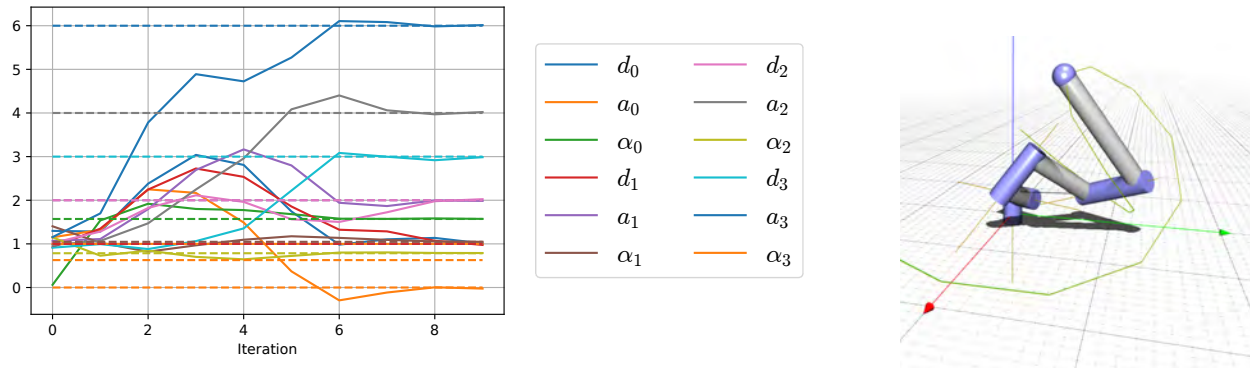


Figure 2.31: Optimization of a 4-DOF robot arm design parameterized by the Denavit-Hartenberg (DH) parameters to match the robot arm that generated a given trajectory. *Left:* evolution of the DH parameters over the optimization iterations. *Right:* visualization of a 4-DOF robot arm and its trajectory in our simulator.

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. For each joint i, the DH parameters are $(d_i, \theta_i, a_i, \alpha_i)$. The preceding motor axis is denoted by z_{i-1} and the current motor axis is denoted by z_i. d_i describes the distance to the joint i projected onto z_{i-1} and θ_i specifies the angle of rotation about z_{i-1}. a_i specifies the distance to joint i in the direction orthogonal to z_{i-1} and α_i describes the angle between z_i and z_{i-1}, rotated about the x-axis of the preceding motor coordinate frame. We are primarily interested in arms with motorized revolute joints, such that θ_i becomes the i-th parameter of our joint state $\mathbf{q}$ and is not part of our parameter vector θ that is to be estimated. We can thus fully specify the relevant kinematic properties of a serial robot arm with N degrees of freedom (DOF) as $\theta = \{d_0, a_0, \alpha_0, \ldots, d_{N-1}, a_{N-1}, \alpha_{N-1}\}$.

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, i.e. the end-point of the last link in the kinematic chain. Given a joint-space trajectory $[\mathbf{q}_0, \mathbf{q}_1, \ldots, \mathbf{q}_T]$, we seek to find the best N-DOF robot arm design, parameterized by DH parameters $\theta^* \in \mathbb{R}^{3N}$, that most closely matches the specified end-effector trajectory:

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

where the forward kinematics function $\operatorname{KIN}(\cdot)$ maps from joint space to the Cartesian coordinates of the end-effector, conditioned on DH parameters θ. Since we compute $\operatorname{KIN}(\cdot)$ using our engine, we may compute derivatives of arbitrary inputs to this function (cf. Fig. 2.23), and use gradient-based optimization through L-BFGS [193] from the Ceres optimization library [1] to converge to arm designs which accurately perform the trajectory-following task, as shown in Fig. 2.31.

2.2.4.3 Adaptive Model-Predictive Control

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 [187] (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

$$\boldsymbol{J} = \sum_{k=0}^{H-1} (\tilde{\mathbf{x}}_k^{\mathrm{T}} Q \tilde{\mathbf{x}}_k + \mathbf{u}_k^{\mathrm{T}} R \mathbf{u}_k) + \tilde{\mathbf{x}}_H^{\mathrm{T}} S \tilde{\mathbf{x}}_H, \tag{2.5}$$

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[‡] $\mathbf{x}^* = (0, 0, 0, 1, 0, 1, 0, 0, 0, 0)$, at which the pole is upright at zero 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

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

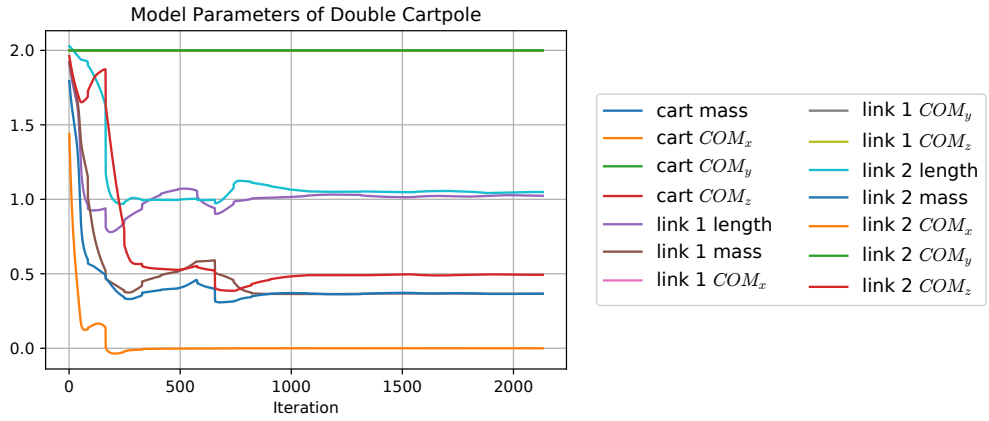


Figure 2.32: Convergence of the physical parameters of a double cartpole, over all model fitting iterations combined, using Adaptive MPC (Algorithm 3) in the DeepMind Control Suite environment.

employed in a different environment. To this end, we incorporate IDS 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 [313] environments that are based on the MuJoCo physics simulator.

Algorithm 3 Adaptive MPC algorithm using differentiable physics model f_θ.

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**
      $\{\mathbf{u}^*\}_t^{t+H} \leftarrow \underset{\mathbf{u}_{1:H}}{\operatorname{argmin}}\, \boldsymbol{J}$ ▷ Trajectory optimization using iLQR with cost from Eq. 2.5
        s.t. $\mathbf{x}_1 = \mathbf{x}_t,\ \ \mathbf{x}_{i+1} = f_\theta(\mathbf{x}_i, u_i),\ \ \underline{\mathbf{u}} \leq \mathbf{u} \leq \bar{\mathbf{u}}$
      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_θ to R by minimizing the state-action prediction loss (Eq. 2.6)
  end for

We fit the parameters θ of the system by minimizing the state-action prediction loss:

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \sum_t ||f_\theta(\mathbf{x}_t, \mathbf{u}_t) - \mathbf{x}_{t+1}||_2^2 \tag{2.6}$$

Thanks to the low dimensionality of the model parameter vector θ (for a double cartpole there are 14 parameters, cf. Fig. 2.32), efficient optimizers such as the quasi-Newton optimizer L-BFGS are applicable,

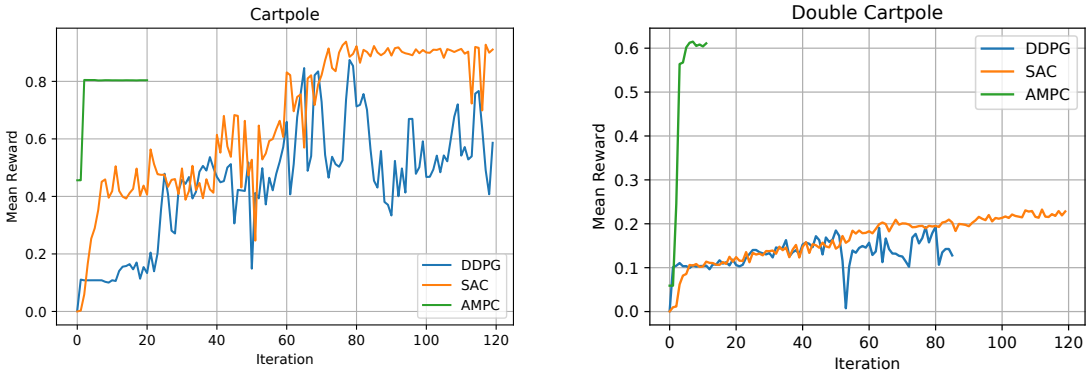


Figure 2.33: Evolution of the mean reward per training episode in the single (left) and double (right) cartpole environments of the model-free reinforcement learning algorithms SAC and DDPG, and our method, adaptive model-predictive control (AMPC).

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 double cartpole (Fig. 2.32). As shown in Fig. 2.22, the models we start from in IDS 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 IDS 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. On the example of a cartpole, Fig. 2.34 visualizes the predicted and actual dynamics for each state dimension after the first (left) and third (right) episode.

Having a current model of the dynamics fitted to the true system dynamics, adaptive MPC significantly outperforms two model-free reinforcement learning baselines, Deep Deterministic Policy Gradient [191] and Soft Actor-Critic [108], in sample efficiency. Both baseline algorithms operate on the same state space

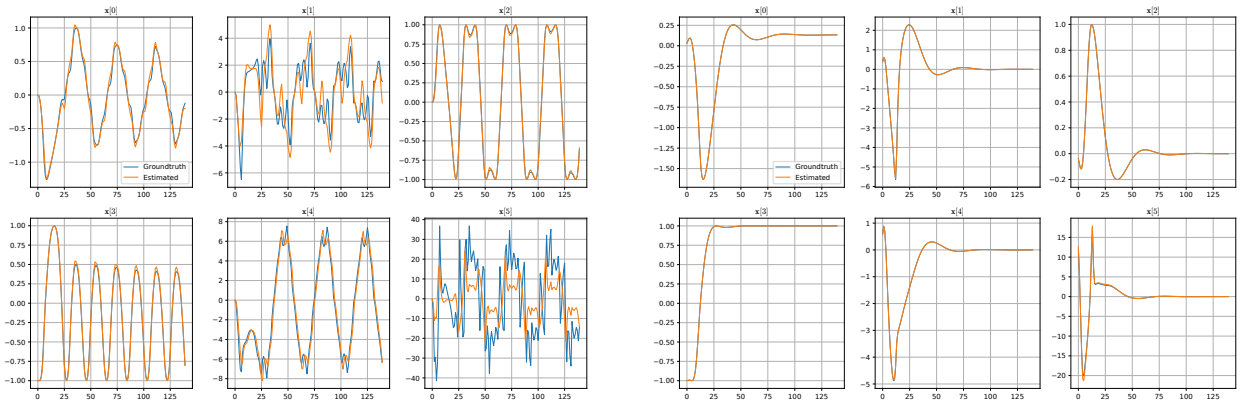


Figure 2.34: Trajectory of the six-dimensional cartpole state[§] over 140 time steps after the first (left) and third (right) episode of Adaptive MPC (Algorithm 3). After two episodes, the differentiable physics model (orange) has converged to model parameters that allow it to accurately predict the cartpole dynamics modelled in MuJoCo (blue). Since by the third episode the control algorithm has converged to a successful cartpole swing-up, the trajectory differ significantly from the first roll-out.

as Adaptive MPC, while receiving a dense reward that matches the negative of our cost function. Although DDPG and SAC are able to eventually attain higher average rewards than adaptive MPC on the single cartpole swing-up task (Fig. 2.33), we note that the iLQR trajectory optimization constraints the force applied to the cartpole within a $[-200\,\mathrm{N}, 200\,\mathrm{N}]$ interval, which caps the overall achievable reward as it takes more time to swing up the cartpole with less forceful movements.

2.2.5 Conclusion

We introduced interactive differentiable simulation (IDS), a novel differentiable layer in the deep learning toolbox that allows for inference of physical parameters, optimal control and system design. Being constrained to the laws of physics, such as conservation of energy and momentum, our proposed model is interpretable in that its parameters have physical meaning. Combined with established learning algorithms from computer vision and receding horizon planning, we have shown how such a physics model can lead to significant improvements in sample efficiency and generalizability. Within a handful of trials in the 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. In conjunction with a differentiable mixture model, IDS can be used to identify multimodal distributions of physical parameters given observations from a real mechanical system.

Chapter 3

Differentiable Simulation

As we introduced in Sec. 2.2, differentiable physics engines enable the tuning of their simulation parameters through gradient-based optimization. We explored articulated rigid-body systems to develop algorithms for parameter inference and model-predictive control. In the following, we focus our differentiable simulation approach on two important areas in robotic perception and manipulation.

In Sec. 3.1, we develop a physically-based model of the sensing process inside a continuous-wave LiDAR sensor that considers the interaction of light and various surface materials, exhibiting diffuse and specular reflection, as well as refraction. This work, that we presented in Heiden et al. [119], enables the localization of the sensor and tracking of highly reflective materials. Additionally, we demonstrate how the influence of the albedo of a material on the computed depth range can be reduced by learning a phenomenological model of the photodiode from real data.

In Sec. 3.2, we present DiSECt, a differentiable simulator for robotic cutting [121]. DiSECt accurately reproduces the forces acting on the knife while a robot slices and presses through various deformable materials, as we validate on real data. The hundreds of simulation parameters underlying our simulation can be tuned efficiently through gradient-based optimization, yielding a close match to real force profiles, as well as simulated nodal field trajectories. By minimizing the effort it takes the robot to cut a material,

slicing motions emerge that successfully reduce the applied knife force considerably on real-robot cutting scenarios.

3.1 Physics-based Simulation of Continuous-Wave LIDAR for Localization, Calibration and Tracking

Light Detection and Ranging (LIDAR) sensors play an important role in the perception stack of autonomous robots, supplying mapping and localization pipelines with depth measurements of the environment. While their accuracy outperforms other types of depth sensors, such as stereo or time-of-flight cameras, the accurate modeling of LIDAR sensors requires laborious manual calibration that typically does not take into account the interaction of laser light with different surface types, incidence angles and other phenomena that significantly influence measurements. In this work, we introduce a physically plausible model of a 2D continuous-wave LIDAR that accounts for the surface-light interactions and simulates the measurement process in the Hokuyo URG-04LX LIDAR. Through automatic differentiation, we employ gradient-based optimization to estimate model parameters from real sensor measurements.

3.1.1 Introduction

Light detection and ranging (LIDAR) sensors, also known as laser range finders (LRF), are active depth sensors that transmit and receive laser light to measure the distance to objects that reflect light back to the sensor. Since the advent of compact planar LIDARs, such as Sick LMS 200 or Hokuyo URG-04LX, they have become an important depth sensor in the perception stack of autonomous mobile robots.

LRF sensors exhibit unique noise characteristics. Their measurements are influenced by the distance to the object, the angle of incidence, the surface properties of objects (e.g. reflectivity and color), the environment temperature, and many other factors, which have been studied extensively [38, 165]. The goal of such characterizations is to derive an empirical sensor model "top-down": given LIDAR measurements under varying conditions, such as surface types, inclination angles, etc., they produce a probabilistic model of noise characteristics for a particular sensor. This statistical model then further informs uncertainty-aware localization and mapping algorithms.

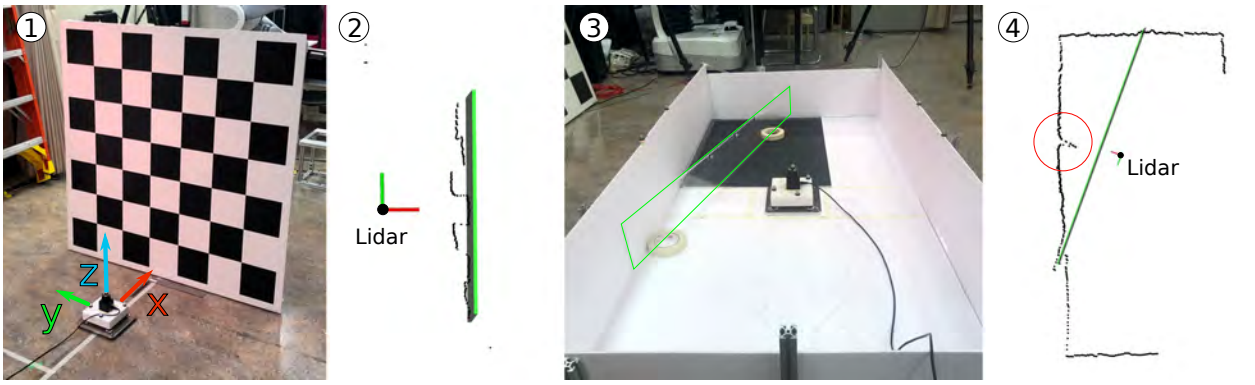


Figure 3.1: Various effects encountered on the Hokuyo URG-04LX laser scanner. (1) LIDAR at a distance of 0.5 m in front of a checkerboard. (2) Measured point cloud obtained from the checkerboard with notable staircase pattern. (3) LIDAR in the cuboid environment, used for the localization and tracking experiments, with a sheet made of transparent plastic (outlined in green). (4) Sensed point cloud where the plastic sheet is invisible, except for a distortion that appears perpendicular to the LIDAR.

In this work, we take a different approach: starting from first principles on how laser light interacts with surfaces under various conditions, we implement a physically plausible simulation that recovers various effects encountered with physical LIDARs, such as spurious measurements, reflected and refracted rays. We simulate the measurement process of continuous-wave LIDARs where the phase shift between two amplitude-modulated laser light waves is calculated to measure the distance. Using automatic differentiation, we compute the gradients of all parameters in our simulation with respect to the simulated measurements and apply gradient-based optimizers to find the simulation parameters that most closely match the true observations.

To the best of our knowledge, our approach is the first physically plausible simulation that captures the measurement process of a continuous-wave LIDAR. Our real-world experiments show that our proposed model can accurately reproduce light-surface phenomena, such as refraction and specular reflection, that have not been considered in most previous LIDAR models.

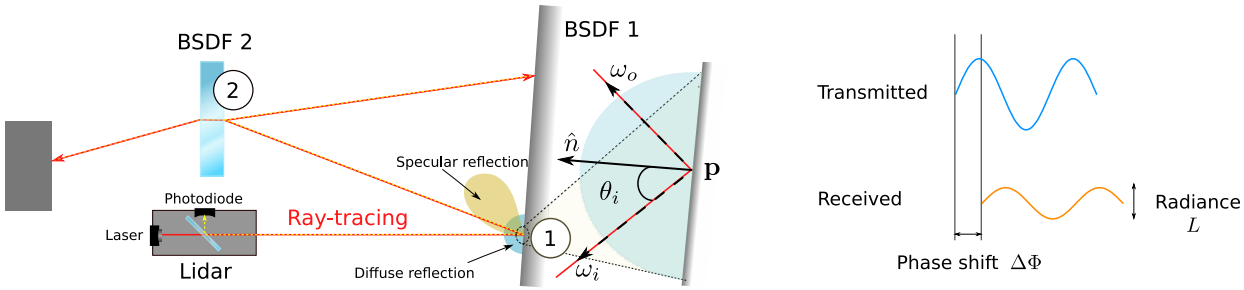


Figure 3.2: *Left:* LIDAR principle where light emanating from a laser is captured by the photodiode on the other side of the half-transparent mirror. The laser ray is traced through the scene to compute the received radiance L (Eq. 3.1) by considering various surface interactions through bidirectional scattering distribution functions (BSDF). BSDF (1) shows the uniform scattering of light due to blackbody radiation, and diffusive and specular reflection spread due to the microfacets in the region around the intersection point $\mathbf{p}$ with surface normal $\hat{n}$. The magnified view of (1) depicts the vectors used for the formulation of BRDFs. BSDF (2) transmits and refracts a part of the light, while reflecting the other part (similar to glass). *Right:* attenuation in the LIDAR received wave compared to the transmitted wave. The phase shift of the received wave resulting from the light's travel time from the laser to an object and back to the photodiode is used to compute the distance (cf. Eq. 3.3).

3.1.2 Related Work

Detailed characterizations of 2D LIDAR sensors have been presented for the Hokuyo laser scanner URG-04LX in Hirohiko Kawata et al. [130], Okubo, Ye, and Borenstein [247], Kneip et al. [165], and Breda [32]. The focus of these works is on experimentally analyzing how the LIDAR measurements are influenced by object color, material, incidence angle, etc. Our goal, in contrast, is to model the physics behind light-surface interaction and the LIDAR's measurement process to allow for the automatic inference of the conditions that caused the observed measurements. A detailed analysis of the physical properties of LIDARs is given in Rosenberger et al. [279], where the authors provide suggestions on how certain real-world effects can be modeled in simulation.

LIDARs and other active range finders are typically modelled in robot simulators to allow for the development of localization and mapping algorithms. Instead of accounting for the intricacies unique to these sensors – as done in the works on characterization – most general-purpose simulators for robotics, such as Gazebo [167] and MuJoCo [319], typically implement z-buffering or ray-casting to compute the depth and perturb it by zero-mean Gaussian noise, leading to a significant gap between simulation and

reality [67]. In this work, we aim for a more detailed simulation that takes into account the interaction of laser light with different surfaces, and many other physical phenomena, as described in Sec. 3.1.3. Similar to [102], we implement ray tracing to account for refractive and reflective surfaces. Additionally, we model the sampling process taking place in continuous-wave LRFs, as well as the detailed light-surface interaction that considers transmitting materials, such as glass and plastic.

Using gradients of models and algorithms has a long history in engineering, e.g. where gradients of ray-tracing equations have been derived to optimize the design of optical systems [77]. With the advent of automatic differentiation frameworks, accurate gradients can be computed algorithmically, powering model-predictive optimal control algorithms based on robotic simulators [92, 14, 124], design optimization [77], and various other use cases [17, 53]. In the context of simulating vision, prior work in computer graphics has focused on differentiable rendering systems [203, 128, 186, 197, 243] that tackle the problem of inverse graphics. Overall, our framework follows a similar idea of constructing a physics-based model of the real system for which the parameters can be efficiently estimated using gradient-based optimization.

3.1.3 LIDAR Model

In the following section, we describe our simulation of a continuous-wave LIDAR, starting from the physics of light-surface interaction to the particular measurement process in the considered sensor.

3.1.3.1 Surface Interaction

When the laser light hits an object at a point $\mathbf{p}$, depending on the surface and angle of inclination, the beam is reflected, scattered and transmitted.

In computer graphics, researchers and practitioners use the rendering equation [259]

$$L_0(\mathbf{p}, \omega_o) = L_e(\mathbf{p}, \omega_o) + \int_\Omega f(\mathbf{p}, \omega_o, \omega_i) L_i(\mathbf{p}, \omega_i) |\cos \theta_i| \, \mathrm{d}\omega_i, \tag{3.1}$$

which expresses the radiance of the reflected light ray from point $\mathbf{p}$ along the direction ω_o, when a light ray of radiance $L_i(\mathbf{p}, \omega_i)$ tracking the direction ω_i collides with a medium at $\mathbf{p}$, integrated over all incoming light directions ω_i from the unit hemisphere Ω. θ_i is the angle made by the incident light ray with the surface normal at $\mathbf{p}$. These quantities are depicted in Fig. 3.2 (left).

The term $L_e(\mathbf{p}, \omega_o)$ captures the effects of blackbody emissions from the material on which the light was incident. All substances having a temperature above absolute zero emit electromagnetic radiation. The radiance of such electromagnetic radiation with a wavelength λ emitted by a blackbody at a particular temperature can be derived using Planck's law [259, Chapter 12.1.1].

The integral term in Eq. 3.1 quantifies the amount of radiance imbibed by the reflected light ray along ω_o from the light rays incident at $\mathbf{p}$. This term is referred to as a bidirectional reflectance distribution function (BRDF). Fresnel equations give the fraction of light reflected along ω_o, denoted as $F_r(\omega_o)$. Thus, if we set the reflection distribution function $f(\mathbf{p}, \omega_o, \omega_i) = F_r(\omega_o)\frac{\delta(\omega_i - \omega_o)}{|\cos\theta_i|}$, then the integral reduces to $F_r(\omega_o)L_i(\mathbf{p}, \omega_i)$. Incidentally, $F_r(\omega_o)$ is a function of the refractive indices of the participating media and can be computed in closed form [259, Chapter 8.2.1].

Apart from being specularly reflected, light can be scattered and diffused by the interacting surface. Diffuse reflection can be described using a BRDF (see Fig. 3.2 for an illustration of uniform diffusion around a point, highlighted by the turquoise hemisphere). To accurately simulate the diffusive and reflective properties of surfaces, they are modelled as a collection of small microfacets. The distribution of microfacet orientations is described statistically to account for the roughness, specular and diffusive properties of surfaces.

Perfect diffusion is described by the Lambertian model that relates the intensity of the reflected light to the cosine of its incidence angle, resulting in surfaces (parameterized by brightness k_R) that reflect light into all directions equally:

$$f(\mathbf{p}, \omega_o, \omega_i) = \frac{k_R}{\pi} \tag{3.2}$$

Besides perfect diffuse reflection, we additionally implement the Oren-Nayar model that represents the microfacet orientation angles by a Gaussian distribution which is parameterized by its standard deviation. Common LIDAR models assume Lambertian surfaces everywhere. In practice, this assumption is often invalidated – particularly when mobile robots are employed in indoor areas, such as office spaces or shopping malls, where many highly reflective or transparent surfaces, such as windows, are present.

A bidirectional transmittance distribution function (BTDF) captures the amount of radiance carried by the light transmitted into the reflecting medium (BSDF(2) in Fig. 3.2 shows light transmission in glass). Since $F_r(\omega_o)$ is the fraction of light reflected, $1 - F_r(\omega_o)$ gives the fraction of light transmitted or refracted. Therefore, by introducing Snell's law of refraction to the integral term in Eq. 3.1, a BTDF L_0^T can be written as

$$L_0^T(\mathbf{p}, \omega_r) = \frac{\eta_o^2}{\eta_i^2}(1 - F_r(\omega_o))L_i(\mathbf{p}, \omega_i),$$

where ω_r is the refracted direction of the light ray, η_o and η_i are the refractive indices of the participating media.

The combination of BTDFs and BRDFs describes the scattering of light upon its encounter with a medium. Any such combination is called a bidirectional scattering distribution function (BSDF). A material is often a combination of multiple BSDF – for example, transparent plastic, besides diffusing light slightly, has specular reflection and transmission components.

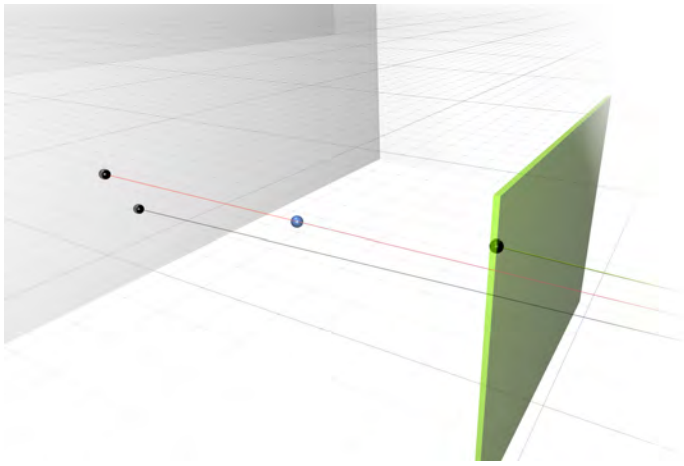


Figure 3.3: Spurious measurement simulated by sub-sampling the laser beam using three rays (see Fig. 3.5 II). Black spheres visualize endpoints of sub-samples, blue point is the final measurement, lying between the green obstacle closer to the LIDAR and the wall further from it.

The integral in Eq. 3.1 needs to be solved for each light ray which can in most cases only be approximated through sampling. Instead of sampling rays from the light source, we assume the LIDAR is the only source of radiance, and trace rays starting from the sensor's location. Such technique is widely known as Whitted integration [333], a recursive ray-tracing algorithm that generates a tree of light rays (we use a maximum recursion depth of five). At each surface interaction, the raytracing tree is expanded by new rays that are cast into the scene according to the BSDF's defined reflective and refractive properties.

In contrast to the rendering of a typical 2D image, we have to take into account the time it takes for the laser light to be reflected back so that we can simulate the measurement process in a LIDAR. Therefore, for each interaction at point $\mathbf{p}$, we record both the distance travelled so far along the render tree and compute the radiance received from $\mathbf{p}$. As the light travels a distance R, its intensity I attenuates according to the inverse square law $I \propto \frac{1}{R^2}$.

Based on the physics-based rendering toolkit `pbrt` [259], we implement ray-tracing with a watertight ray-triangle intersection algorithm and represent all scene geometry as triangle meshes. To efficiently find intersecting meshes, we store the scene geometry in a k-d tree.

3.1.3.2 Measuring Distance

There are two main types of operating principles for LIDARs [303]: pulsed systems emit a laser pulse and measure the time until the pulse is received to compute the distance. Continuous-wave (CW) LIDARs emit a sinusoidal signal of known wavelength to estimate the distance by measuring the phase difference between the transmitted and received signal (Fig. 3.2 right). CW LIDARs typically require less powerful lasers, thus reducing hardware costs.

The LIDAR which we model in this work is the URG-04LX LRF by Hokuyo Automatic Co., Ltd. Being part of the URG series, as described in [158], it is a low-cost, planar CW LIDAR with a field-of-view of $240°$, transmitting laser rays into 682 directions (thus achieving an angular resolution of $0.352°$) at a 10 Hz frequency. It modulates the amplitude of the signal to measure distance.

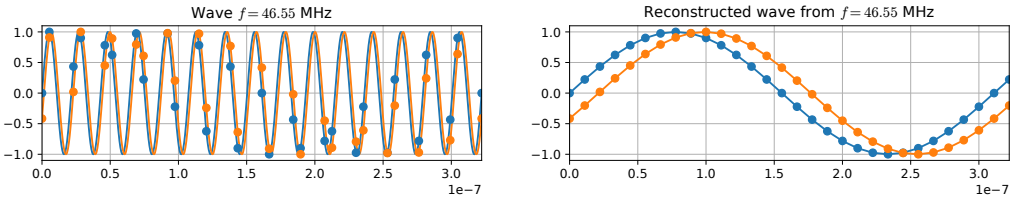


Figure 3.4: Measurement process in the CW LIDAR URG-04LX [130]. Received (orange) and transmitted (blue) signal at a 46.55 MHz frequency with a phase shift $\Delta\Phi$ resulting from a ranging distance of three meters. *Left:* analogue-to-digital conversion over 15 periods at 30 sampling steps s_i (dots) at times t_i (Eq. 3.4). *Right:* higher-resolution waveform retained from the coarse samples s_i of the received and transmitted signals.

For a pulse ranging LIDAR, given the two-way travel time t_R of the emitted light, the measured range $\hat{R}$ is computed by $\hat{R} = t_R c/2$, where c is the speed of light. As we are interested in modeling CW LIDARs, the range is computed from the measured phase difference between the transmitted and received signal. $\hat{R}$ is proportional to period time $T = 1/f$ and phase difference $\Delta\Phi$:

$$\hat{R} = \frac{c}{2}\frac{T}{2\pi}\Delta\Phi = \frac{1}{4\pi}\frac{c}{f}\Delta\Phi. \tag{3.3}$$

In the case of the Hokuyo LRF, the amplitude-modulated laser light is received by an avalanche photodiode [158]. Throughout the ray-tracing process for a single measurement ray, we compute both the actual range R^* traveled up to the light-surface interaction plus the received radiance L (cf. Section 3.1.3.1). Each such return produces two new *received* waveforms over 15 periods at $f_1 = 46.55\,\text{MHz}$ and $f_2 = 53.2\,\text{MHz}$ frequencies with phase shifts $\Delta\Phi^*(f_1)$, $\Delta\Phi^*(f_2) + 90°$,* and amplitude $\propto L$ (Fig. 3.2 right), respectively. For the two frequencies f_1 and f_2, we sum up the received waveforms until the ray-tracing process is complete.

Next, the measurement process of the Hokuyo LIDAR begins by sampling both waveforms through 30 samples s_i at times t_i. The samples are reordered to form a high-resolution waveform (Fig. 3.4 right). Using a simple discrete Fourier series approximation, the phase is computed via

$$\Phi = \arctan\left(\frac{\sum_i s_i \cos t_i}{\sum_i s_i \sin t_i}\right). \tag{3.4}$$

Analogously, the phase is computed for the transmitted signal and the phase difference to the received signal is computed by subtracting both phases. From this phase shift $\Delta\Phi$, the measured distance is computed via Eq. 3.3. The two phase shifts from the 46.55 MHz and 53.2 MHz waveforms are compared in order to resolve ambiguity in the phase difference of a single wave [158].

LIDARs exhibit spurious measurements (Fig. 3.3), i.e., when the laser beam partially hits an obstacle closer to the scanner and another obstacle at a greater distance, the measured depth lies somewhere between these two obstacles. This phenomenon follows from the conical shape of the laser beam. As noted by Rosenberger et al. [279], a single infinitesimally thin ray is not sufficient to sample the behavior of beam divergence. Instead, we model the laser beam using three rays which are aligned around the central laser beam ray (Fig. 3.5). At a distance of 4 m, the beam diameter is specified to be 40 mm.

*Given actual range R^*, the actual phase shift for the waveform of frequency f is computed as $\Delta\Phi^*(f) = 4\pi\frac{f}{c}\frac{1}{R^*}$.

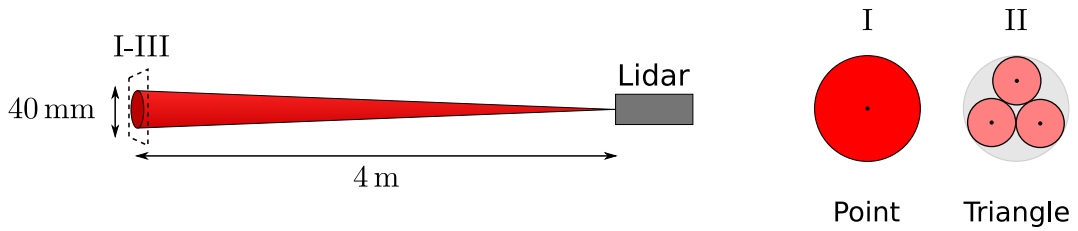


Figure 3.5: *Left:* The beam divergence of the laser light emitted by the LIDAR amounts to a beam diameter of 4 cm at a distance of 4 m. *Right:* Cross-section views of approaches to sample the laser beam signals(cone) via rays. Simulating a LIDAR using a single ray (I) does not allow for capturing the beam divergence, while three (II) or more rays offer a more accurate simulation. For three beams, the contribution of each ray is weighted equally so that the total received radiance per beam is the same as for the point under the same conditions.

3.1.4 Experimental Results

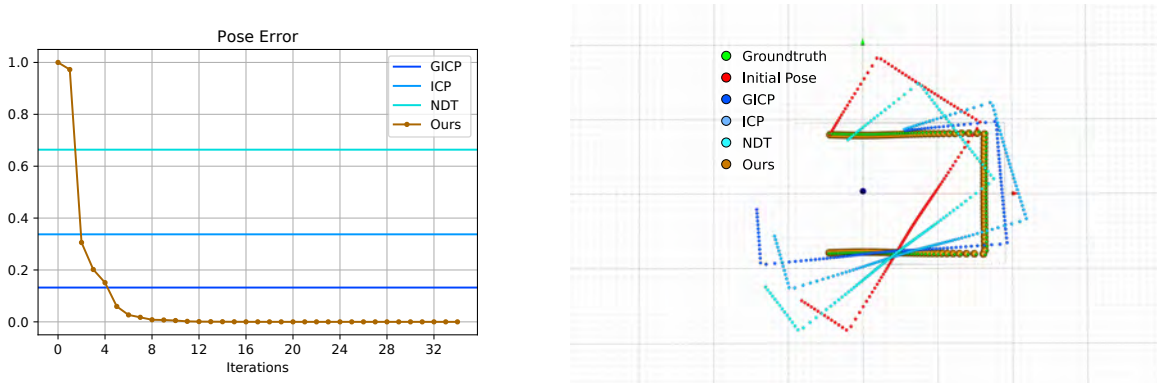


Figure 3.6: *Left:* Evolution of the pose error, i.e., the SE(2) distance (accounting for yaw angle difference) between the estimated transform and the ground-truth pose of the LIDAR, while localizing the LIDAR in a known environment given its sensor output. The x-axis represents the number of iterations, no iteration-wise information was available for the point cloud registration algorithms GICP, ICP and NDT. *Right:* Visualization of the pose estimation results by transforming the point cloud of the given depth measurements by the estimated sensor transform.

Besides retaining various effects from the real world, we demonstrate in our experiments the capability of our simulation to serve as a model that can be fitted to actual measurements from a physical LIDAR. To this end, we focus on scenarios where certain inputs to our model need to be estimated – such as the sensor's pose, material properties of intersecting surfaces, and parameters internal to the sensing process – so that the resulting simulated measurements closely match the real data.

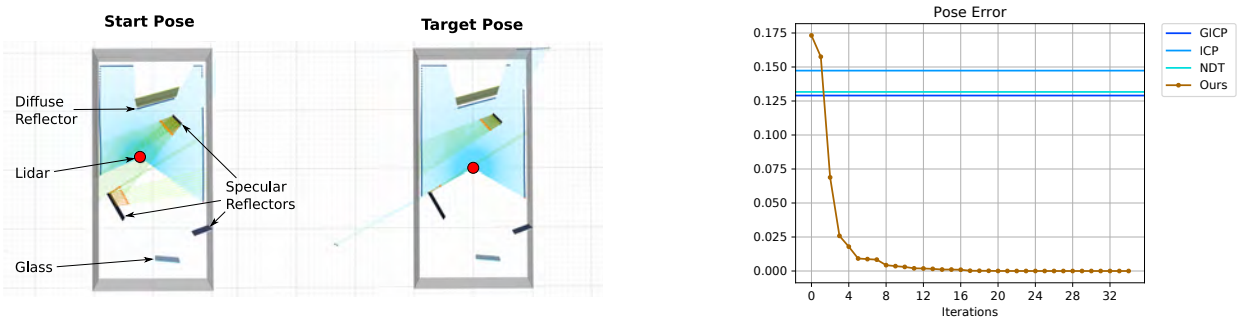


Figure 3.7: *Left:* Experimental setup of localizing the LIDAR in a known, complex environment consisting of mirrors (specular reflectors), diffuse reflectors (Lambertian) and a glass element. The LIDAR (red) takes measurements (blue dots) at the start pose, and, given the measurements from the target pose, estimate the necessary transform. *Right:* Evolution of the pose error Eq. 3.5 (SE(2) distance) between the estimated transform and the groundtruth.

We investigate several scenarios and replicate them in our simulator to study various applications of the proposed LIDAR model. As shown in Fig. 3.1 (3), we created a cuboid environment in which the LIDAR is placed. The environment has the dimensions (width $\times$ depth $\times$ wall height) 185 cm $\times$ 92 cm $\times$ 28 cm. We track the LRF using a Vicon motion capture system and place markers on the objects we want to track to achieve sub-millimeter groundtruth accuracy.

Throughout the experiments, the optimization objective is

$$\underset{\theta}{\text{minimize}} \left(\mathcal{L} = ||f(\theta) - y||^2 \right), \tag{3.5}$$

where the reality gap between simulated measurements $f(\theta)$ and actual measurements y is to be minimized by adapting the simulation parameters θ. These parameters are defined separately for each experiment.

Our simulator is implemented in C++ and uses the automatic differentiation framework Stan Math [39] to algorithmically compute gradients $\frac{\partial \mathcal{L}}{\partial \theta}$. Throughout all experiments, we use an off-the-shelf implementation of the gradient-based optimization algorithm L-BFGS with Wolfe line search from the Ceres library [1] to optimize Eq. 3.5 with the calculated gradients throughout all experiments.

3.1.4.1 Localization in a Known Environment

Being placed in a simple cuboid environment where the wall geometries and surface properties are known, the goal in this experiment is to localize the LIDAR. Such a scenario is very common in LIDAR odometry where the current set of range measurements is registered with the point cloud obtained from a previous observation to obtain the rigid transform between the two poses.

Given a set of range measurements, we optimize Eq. 3.5 for the sensor's SE(2) pose θ. The initial pose is defined as $(0\,\mathrm{m}, 0\,\mathrm{m}, 60^\circ)$, the groundtruth pose is $(0\,\mathrm{m}, 0\,\mathrm{m}, 0^\circ)$. We compare our approach to common point cloud registration algorithms. Given the depth information from the current pose (our initial guess), their task is to find the relative transform to a target point cloud, which stems from the actual LIDAR. Our baselines are Iterative Closest Point (ICP) [24], Generalized ICP (GICP) [301], and Normal Distributions Transform (NDT) [25]. We use their implementations from the Point Cloud Library [288].

As shown in Fig. 3.6 (left), the optimization using our proposed model converges after 34 iterations to the correct pose. Executed on an Intel Core i7-8700K CPU (3.70 GHz), the computation takes 1.6 s, while the respective point cloud registration algorithms converge after 10-20 iterations (since we do not have access to the iteration-wise performance of these algorithms we show a flat line in the graph). GICP achieves the highest baseline accuracy with a final pose error of 0.13. Although the registration baselines finish the computation by one to two orders of magnitude faster than our approach, we note that our current implementation can be sped up considerably through parallelization since each of the 682 LRF returns can be computed independently.

While classical point cloud registration algorithms are generally able to find good solutions for simple scenarios such as the one shown in Fig. 3.6, where the points follow a simple rectangular geometry, and the shape largely remains constant under the rigid transformation, depth measurements can become more complex when highly reflective surfaces (e.g., mirrors or glass surfaces) are present. Such phenomena become particularly critical in autonomous driving scenarios, where metallic car paint of darker colors

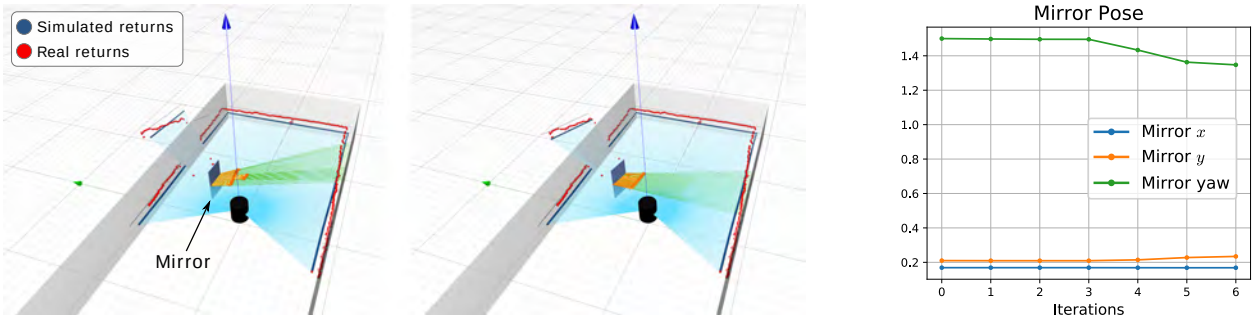


Figure 3.8: *Left:* Initial setup for the tracking experiment where a mirror needs to be localized in a cuboid environment given real sensor measurements (red dots). Orange arrows with emanating green lines indicate reflection rays causing the measurement rays that hit the mirror to appear further, as the distance to the opposite wall is measured. *Center:* Converged pose estimate of the mirror after 6 iterations. *Right:* Evolution of the mirror's SE(2) pose throughout the optimization.

absorbs the infrared light from the laser significantly [302]. Specular reflection and partial transparency in glass windows further distort the obtained depth measurements, affecting many optical sensor pipelines of robots deployed indoors.

To this end, we investigate an entirely simulated scenario where the LIDAR is exposed to multiple specular reflectors (mirrors), a glass surface and a diffuse surface (see Fig. 3.7 left). Placed at the initial SE(2) pose $(0.1\,\mathrm{m}, 0.1\,\mathrm{m}, 10°)$, the ground truth pose $(0\,\mathrm{m}, 0\,\mathrm{m}, 0°)$ needs to be inferred given the sensor measurements. While the initial location is close to the actual pose, the point cloud registration algorithms achieve significantly less improvement over the initial guess compared to our optimization-based approach. As the point clouds differ significantly, a model that relies solely on their shapes is prone to find suboptimal solutions, while the gradient-based optimization applied to Eq. 3.5 finds the accurate pose within 34 iterations (see Fig. 3.7 right).

3.1.4.2 Tracking a Mirror

Besides localizing the LIDAR, other objects in the environment can be tracked if they are modelled within the simulator. As shown in Fig. 3.8, we place a mirror near the LIDAR and estimate its SE(2) pose using gradient-based optimization. Similar to Section 3.1.4.1, our system identification approach applied to

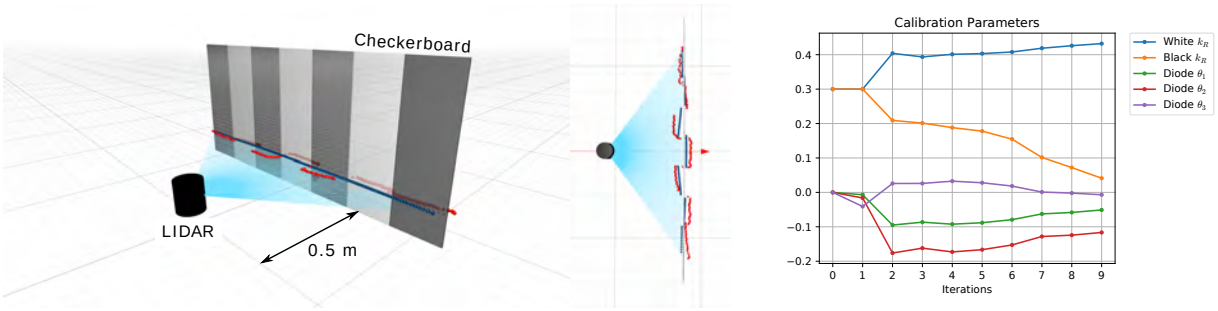


Figure 3.9: *Left:* Experimental setup of the LIDAR in front of a checkerboard, where measurements are simulated (blue lines visualize rays, blue dots are placed at the simulated depth). Measurements not hitting the checkerboard are not shown. Red dots indicate the measurements from the actual Hokuyo scanner in front of a real checkerboard. *Center:* Top-down view of the simulation with converged surface parameters and diode calibration settings. The simulated measurements (blue) appear close to the real measurements (red). *Right:* Evolution of the surface properties (Lambertian reflectance coefficients k_R for the white and black sections) and the diode calibration parameters $\theta_1, \theta_2, \theta_3$.

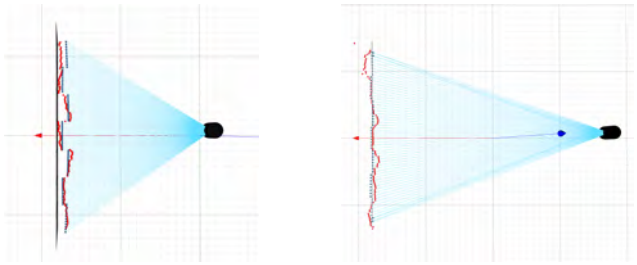


Figure 3.10: Evaluation of surface properties and diode parameters obtained through the optimization described in Section 3.1.4.3. Actual measurements from the real system are shown in red, simulated ranges in blue. *Left:* Checkerboard at a distance of 90 cm. *Right:* Checkerboard at a distance of 1.6 m.

our model estimates the correct pose by minimizing the ℓ_2-norm between the simulated and real depth measurements within six iterations (see Fig. 3.8 right).

Our current approach suffers from the loss of gradient information when the object to be tracked is moved outside the field-of-view of the LIDAR, e.g. when it is behind the walls or in the blind spot of the sensor. Approaches from the computer graphics community, such as Soft Rasterizer [197], overcome the issue of missing gradient information by relaxing the intersection checks through a convex combination of blurred edges so that even faces that are hidden contribute to the intersection test. Future work is directed toward extending our model with such techniques to make tracking and localization more amenable to gradient-based optimization.

3.1.4.3 Diode Calibration and Inferring Surface Properties

As noted in several characterizations of low-energy LIDARs, such as the URG-04LX [247], they suffer from a strong dependence of the measured depth on the received radiance. As shown in Fig. 3.1 (2), the dark sections of the checkerboard appear closer than the more reflective white sections, resulting in a staircase effect. While the exact cause of this effect is unknown to us, we seek to replicate this behavior through a data-driven approach and leverage our model to infer surface properties given the sensor's range measurements.

We model the relationship between radiance and phase shift as a quadratic polynomial and regress its coefficients through optimizing Eq. 3.5. This can be seen as a form of sum-of-squares optimization [250]. The phase shift $\Delta\Phi$ to modulate the two continuous waves in Eq. 3.3 is modified as follows: $\Delta\Phi' \leftarrow \Delta\Phi - (aL^2 + bL + c)$, where L is the received radiance and $a, b, c \in \mathbb{R}$ are the polynomial coefficients to be estimated. We model the checkerboard using dark and bright stripes of the same dimensions as the real checkerboard (Fig. 3.9 left) and assign Lambertian materials of separate reflectivity coefficients k_R (cf. Eq. 3.2) to the black and white sections.

It should be noted that this problem is particularly difficult since we do not have access to the actual intensities of each return but instead only the filtered distance measurements. Nonetheless, we are able to estimate the two different materials of a checkerboard, and converge to polynomial coefficients within 9 iterations (Fig. 3.9 right) that closely match the real measurements (Fig. 3.9 center). When we move the LIDAR from its initial configuration of 0.5 m distance to other distances in front of the checkerboard, we see that the simulated measurements still retain the staircase pattern from the real world accurately (Fig. 3.10).

3.1.5 Conclusion

We have presented a physics-based model of a continuous-wave LIDAR that allows for the optimization-based inference of model parameters. It captures the detailed interaction between the laser light and various surfaces, accounting for geometrical and material properties. Through experiments with real-world measurements from the Hokuyo URG-04LX LRF, we have demonstrated various applications of our model in localization, calibration and tracking.

Future research is directed towards extending our proposed simulation to model more powerful pulsed LIDAR sensors and investigate its application in real-world domains on a larger scale.

3.2 DiSECt: A Differentiable Simulator for Parameter Inference and Control in Robotic Cutting

Robotic cutting of soft materials is critical for applications such as food processing, household automation, and surgical manipulation. As in other areas of robotics, simulators can facilitate controller verification, policy learning, and dataset generation. Moreover, *differentiable* simulators can enable gradient-based optimization, which is invaluable for calibrating simulation parameters and optimizing controllers. In this work, we present DiSECt: the first differentiable simulator for cutting soft materials. The simulator augments the finite element method (FEM) with a continuous contact model based on signed distance fields (SDF), as well as a continuous damage model that inserts springs on opposite sides of the cutting plane and allows them to weaken until zero stiffness, enabling crack formation. Through various experiments, we evaluate the performance of the simulator. We first show that the simulator can be calibrated to match resultant forces and deformation fields from a state-of-the-art commercial solver and real-world cutting datasets, with generality across cutting velocities and object instances. We then show that Bayesian inference can be performed efficiently by leveraging the differentiability of the simulator, estimating posteriors over hundreds of parameters in a fraction of the time of derivative-free methods. Next, we illustrate that control parameters in the simulation can be optimized to minimize cutting forces via lateral slicing motions. Finally, we conduct experiments on a real robot arm equipped with a slicing knife to infer simulation parameters from force measurements. By optimizing the slicing motion of the knife, we show on fruit cutting scenarios that the average knife force can be reduced by more than 40% compared to a vertical cutting motion.

3.2.1 Introduction

Robotic cutting of soft materials is critical for various real-world applications, including food processing, household automation, surgical manipulation, and manufacturing of deformable objects. As in other areas

Figure 3.11: A rendering from our differentiable cutting simulator. DiSECt provides accurate gradients of the cutting process, allowing us to efficiently fit model parameters to real-world measurements, and optimize cutting motions.

of robotics, simulators can allow researchers to verify controllers, train control policies, and generate synthetic datasets for cutting, as well as avoid expensive or time-consuming real-world trials. In addition, cutting is inherently destructive and irreversible; thus, accurate and efficient simulators are indispensable for automating safety-critical tasks like robotic surgery.

However, simulating the cutting of soft materials is challenging. Cutting involves diverse physical phenomena, including contact, friction, elastic deformation, damage, plastic deformation, crack initiation, and/or fracture. A variety of simulation methods have been introduced, including mesh-based approaches (e.g., extensions of the finite element method (FEM)), particle-based approaches (e.g., smoothed particle hydrodynamics (SPH)), and hybrid techniques. For accuracy, these methods often require computationally-expensive re-meshing, or simulation of millions of particle interactions.

Moreover, a physically-accurate forward simulator is necessary but not sufficient for accurate and useful predictions. The material properties of real-world soft materials (e.g., ripening fruits, diseased tissue) are often unknown and highly heterogeneous, mandating calibration of material models. In addition, ideal trajectories for cutting may not be known beforehand, requiring efficient optimization of control actions. In other fields, the need to infer material and control parameters has motivated the development of differentiable simulators, which can harness efficient, gradient-based optimization methods.

We present DiSECt, the first differentiable simulator for the cutting of soft materials (Fig. 3.11). The simulator has several key features. First, contact between the cutting instrument (i.e., knife) and the object is resolved using a continuous signed distance field (SDF) representation of the knife. Second, a mesh-based representation of the object is used; given a predefined cutting plane, virtual nodes are inserted along the surface in a preprocessing step that occurs only once. Third, cracks are introduced using a continuous damage model, where springs are inserted on either side of the cutting plane. Under application of force over time, the springs can progressively weaken until zero stiffness, producing a crack. The continuous contact and damage models enable differentiability, and the springs and particles provide hundreds of degrees of freedom that can be used to calibrate the simulator against ground-truth data. Gradients of any simulation parameter are computed using automatic differentiation via source-code transformation.

This work offers the following contributions:

1. The first differentiable simulator for cutting soft materials.

2. A comparison of gradient-based and derivative-free methods for calibrating the simulator against ground-truth data from commercial solvers and real-world datasets. The comparison demonstrates that the differentiability of the simulator enables highly efficient estimation of posteriors over hundreds of simulation parameters.

3. A performance evaluation of the calibrated simulator for unseen cutting velocities and object instances. It demonstrates that the simulator can accurately predict resultant forces and nodal displacement fields, with simulation speeds that are orders of magnitude faster than a comparable commercial solver.

4. An application of the simulator to robotic control, optimizing knife motion trajectories to minimize cutting forces under time constraints.

5. Experimental results on a real robot that validate our approach in parameter inference and trajectory optimization, where optimized slicing trajectories yield cutting motions that require significantly less force than vertical cuts.

3.2.2 Related Work

3.2.2.1 Modeling and Simulation of Cutting

Analytical modeling Cutting is a branch of elastoplastic fracture mechanics [12], where theoretical analysis has primarily focused on metal cutting [221] and brittle materials [100]. A comprehensive treatment of the mechanics involved in cutting of metals, biomaterials and non-metals is given in Atkins and Atkins [12]. Analytical models of the forces acting on a knife as it cuts through soft materials have been derived in Debao Zhou et al. [58] and Mu, Xue, and Jia [236].

Mesh-based simulation Among the numerical methods that implement such analytical models, the Finite Element Method (FEM) [141, 22] is a commonly used technique to simulate deformable bodies. FEM solves partial differential equations over a given domain (defined by a mesh) by discretizing it into simpler elements, such as tetrahedra (which we use in this work). Without modifying the mesh topology, the Extended FEM (X-FEM) [225] augments mesh elements by enrichment functions to model fracture mechanics processes [159, 153, 170].

In classical FEM, topological changes that result from cutting and other fracture processes mandate an adaption of the mesh resolution so that the propagation of the crack can be accurately simulated. Approaches in mechanical engineering [8] and computer graphics [344, 26, 36, 171, 252] have been introduced that re-mesh the simulation domain to accommodate cuts and other forms of cracks.

Virtual node algorithms (VNA) [227, 304, 331] duplicate mesh elements that intersect with the cutting surface, resulting in elements with portions of real material and empty regions. At the cutting interface,

virtual nodes are introduced that allow for a two-way coupling of contact and elastic forces with the underlying mesh of the separated parts. We leverage VNA to augment our FEM simulation with extra degrees of freedom to allow the fine-grained simulation of contact between the material and the knife. Furthermore, by augmenting the mesh only once at the beginning of the cut, we avoid discontinuous re-meshing operations and are able to compute gradients from our simulator.

Mesh-free and hybrid approaches So-called mesh-free Lagrangian methods, such as smoothed-particle hydrodynamics (SPH) [93, 228], element-free Galerkin (EFG) [21], and smoothed-particle Galerkin (SPG) [339], simulate continuous media by many particles that cover the simulation domain and the dynamics is determined by a kernel that defines the interaction between spatially close particles. Hybrid approaches have been proposed, such as the Material Point Method (MPM) [138, 329, 338, 337], that combine Eulerian (grid-based) and Lagrangian (particle-based) techniques. Position-based Dynamics (PBD) [237] has been explored to simulate cutting in interactive surgery simulators [23, 248].

Robotic cutting In Long et al. [201], a robotic meat-cutting system is introduced that scans objects online to simulate deformable objects and uses impedance control to steer the knife. In Thananjeyan et al. [314] and Krishnan et al. [173], a simplified mass-spring model is used for learning to cut planar surfaces. Such trained policies are then transferred to a physical robotic surgery system. By studying the mechanics of cutting, Mu, Xue, and Jia [236] derive control strategies involving pressing, pushing, and slicing motions. Similarly, Debao Zhou et al. [58, 57] analyze cutting "by pressing and slicing," while taking into account the blade sharpness of the knife. Constrained optimization is used in Wijayarathne et al. [334] to optimize cutting trajectories while accounting for contact forces, whereas in our control experiment we furthermore account for the coupling of elastic and contact force between the knife and the deformable object being cut. In Jamdagni and Jia [149], FEM simulation for robotic cutting is devised where crack propagation is simulated at the cross sections on a high-resolution 2D mesh, which is coupled with a coarser 3D mesh

simulation. Meshes are obtained from laser scans of biomaterials, and force profiles recorded from a force sensor attached to a robot end-effector as it vertically cuts through various foodstuffs. In this work, we leverage this dataset of real-world cutting trajectories. Fully data-driven models have been learned to facilitate model-predictive control in robotic cutting [182, 223].

3.2.2.2 Differentiable Simulation

Differentiable simulation has gained increasing attention, as it allows for the use of efficient gradient-based optimization algorithms to tune simulation parameters or control policies [92, 40, 14, 124, 169, 137, 266, 119, 91, 239, 122]. Although finite differencing may be used to approximate the derivative of a simulation output, this approach suffers from accuracy problems and does not scale to large numbers of parameters [211]. In this work we employ reverse-mode automatic differentiation to obtain gradient information via source-code transformation [99, 145]. Recent work has shown the potential for source-code transformation to generate efficient parallel kernels for reverse-mode differentiation using graphics-processing units (GPUs) [139, 137].

3.2.2.3 Parameter Inference for Simulators

Simulation-based inference is a methodology that has emerged across various fields of science [55]. In parameter identification, optimization-based approaches are often utilized to obtain point estimates of the simulation parameters (such as for constitutive equations [210, 109]) that minimize the model error as measured from the dynamics of the real system. More often, when analytical gradients are unavailable or too expensive to obtain, optimization with finite differencing or gradient-free methods have been applied [220, 240], particularly for dynamical systems through system identification [200].

Probabilistic inference techniques, on the other hand, seek to infer a distribution of simulation parameters that allows downstream applications to evaluate the uncertainty of the estimates. Such methods have

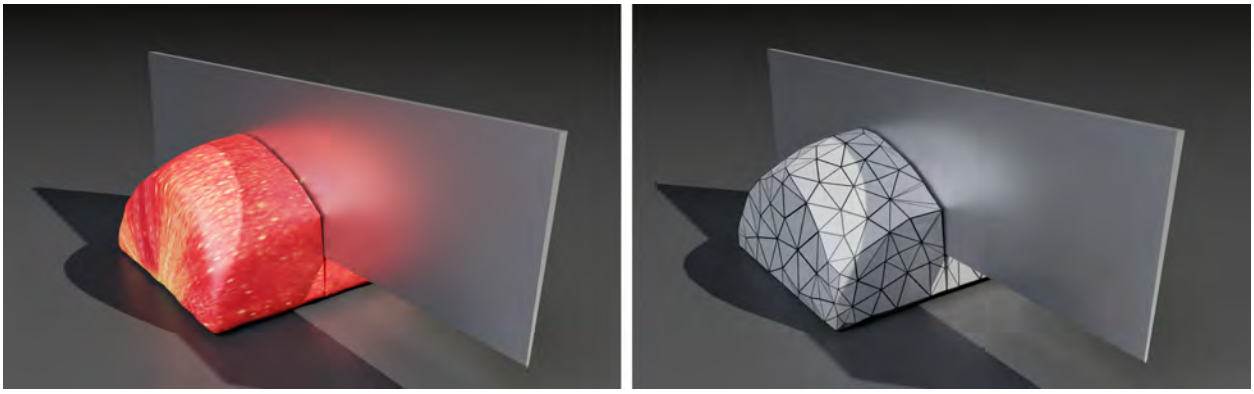


Figure 3.12: Visualization of an apple slice. We use a tetrahedral FEM-based model of the apple generated from scanned real-world data [149].

been applied to learn conditional densities of simulation parameters given trajectories from the simulator and the real system [320, 249, 274, 135, 213, 215, 117].

Parameter inference may also be integrated in a closed-loop system [42, 274, 218, 233], where the currently estimated posterior over simulation parameters guides domain randomization and policy learning. Such approaches iteratively reduce the sim2real gap.

3.2.3 DiSECt: Differentiable Simulator for Cutting

The design of our simulator for cutting biomaterials is motivated by three aspects:

1. The model has to be physically plausible such that the effects of changing physical quantities from continuum and fracture mechanics can be observed realistically.

2. While the cutting process is an inherently discontinuous operation, wherever possible, gradients of the simulation parameters must be efficiently obtainable.

3. The simulator must allow for sufficient degrees of freedom so that fine differences in material properties can be identified at localized places where the knife cuts through a heterogeneous material.

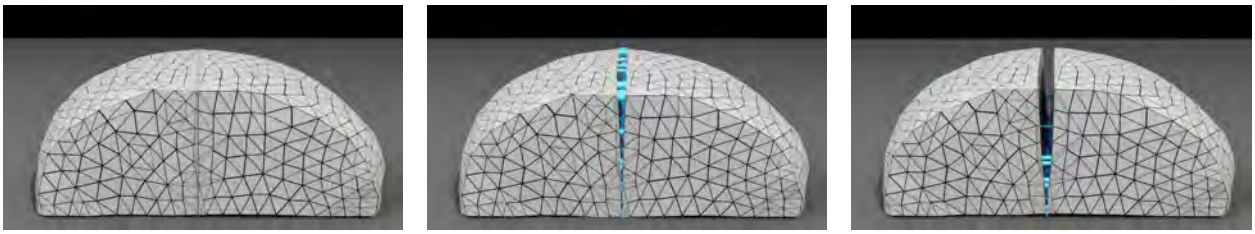


Figure 3.13: To smoothly model damage, we insert springs (shown in blue) between cut elements. We incrementally reduce the spring stiffness based on damage over time, proportional to the contact force applied from the knife. From left at $t = 0\,\mathrm{s}$, $t = 0.8\,\mathrm{s}$, $t = 1.2\,\mathrm{s}$. Here, we have removed the knife from visualization to show the inserted springs clearly.

3.2.3.1 Continuum Mechanics

We implement the Finite Element Method (FEM) to simulate the dynamics of the soft materials used throughout this work. Based on a tetrahedral discretization of the cutting target object, elastic forces are computed that take into account material properties, such as Young's modulus, Poisson's ratio, and density.

We employ a Neo-Hookean constitutive model following the strain-energy density function from [307], which has been designed to model biological tissue and preserve volume when the mesh undergoes large deformations:

$$\Psi = \frac{\mu}{2}(I_C - 3) + \frac{\lambda}{2}(J - \alpha)^2 - \frac{\mu}{2}\log(I_C + 1). \tag{3.6}$$

Here λ, μ are the Lamé parameters[†] and α is a constant. $J = \det(\mathbf{F})$ is the relative volume change, $I_C = \mathrm{tr}(\mathbf{F}^T\mathbf{F})$ is the first invariant of the Cauchy-Green deformation tensor, and $\mathbf{F}$ is the deformation gradient. We give the material properties for our experimental objects in Tab. B.1. Integrating over each tetrahedral element and summing the energy from (3.6) over all elements gives the total elastic potential energy. Forces $\mathbf{f}_{\text{elastic}}$ are derived from the energy gradient analytically and integrated using a semi-implicit

[†]Lamé parameters are a reformulation of Young's modulus and Poisson's ratio; hence we use these terms interchangeably.

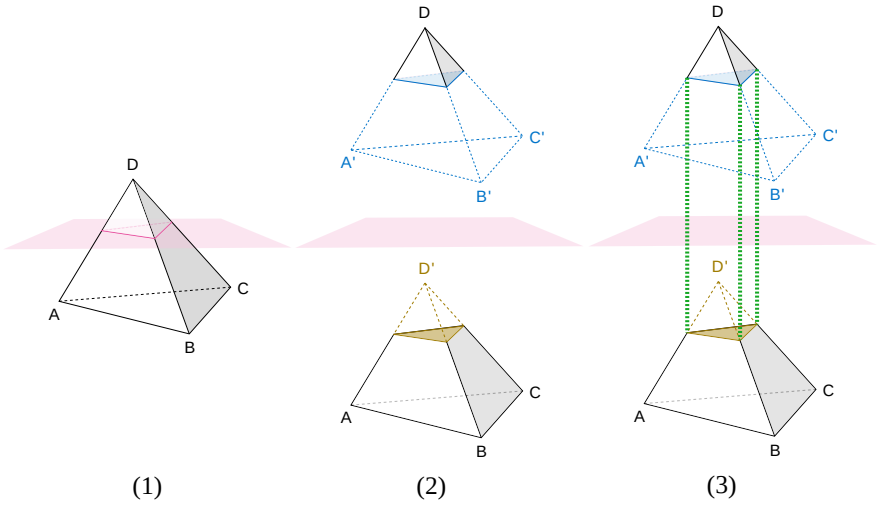


Figure 3.14: Pre-processing of a tetrahedral mesh (visualized by a single tet) prior to being cut along a given cutting surface (red plane). Given the original mesh, (1) intersecting elements are identified, and (2) duplicated such that the intersection geometry is retained in each part of the cut elements. Virtual nodes are inserted where the edges are intersecting the cutting surface. (3) The original and duplicated elements are connected by springs (green) between the virtual nodes.

Euler scheme. In addition to elastic forces, we include a strain-rate dissipation potential to model internal damping [207].

3.2.3.2 Mesh Pre-Processing

Before the simulation begins, the cutting surface for the entire cut is given as a triangle mesh. We implement the Virtual Node Algorithm from Sifakis, Der, and Fedkiw [304], which, as shown in Fig. 3.14, duplicates the mesh elements (tets) that intersect with the cut (subfigure 2), so that the tet above the cutting surface has a portion of material above the cut, and one empty portion below the cut (and vice versa for the reciprocal element). Next, we insert virtual nodes at the intersection points on the edges. These nodes are only represented by their barycentric coordinate $u \in [0, 1]$.

Similar to the two-way coupled "soft particles" described in Sifakis et al. [305], each virtual node's 3D position $\tilde{\mathbf{x}}$ and velocity $\tilde{\mathbf{v}}$ is defined entirely by the coordinates of its two parent vertices, indexed i and j:

$$\tilde{\mathbf{x}} = (1-u)\mathbf{x}_i + u\mathbf{x}_j \qquad \tilde{\mathbf{v}} = (1-u)\mathbf{v}_i + u\mathbf{v}_j.$$

The edge sections along the non-empty portions of the duplicated mesh elements participate in contact dynamics and propagate the resulting contact forces back to their parent nodes, as described in Section 3.2.3.3. Edge sections from the empty portions of the mesh create no collisions and are solely updated from the FEM dynamics of the parent vertices.

In the final step of this mesh preprocessing phase (subfigure 3), springs are inserted that connect the virtual nodes on both sides of the cutting surface which originally belonged to the same edge of the mesh. These springs allow us to simulate damage occurring during the cutting process in an entirely continuous (and thereby differentiable) manner, by weakening their stiffness values as knife contact forces are applied over time.

While our approach in its current form assumes that the entire cutting surface is given before the cutting simulation begins, interactive cutting applications could be accommodated by interweaving this mesh augmentation step with the actual cutting simulation. Nonetheless, the practicality of using such an interactive approach for parameter inference remains to be validated.

3.2.3.3 Contact Dynamics

Following [208], we implement a contact model that represents the knife shape by a signed distance function (SDF) (Fig. 3.15) that interacts with the tetrahedral mesh of the object being cut. To find the closest point between an edge from the mesh and the SDF, we run 20 iterations of the Frank-Wolfe algorithm (Algorithm 7 in appendix), that uses the gradient information from the SDF to find a locally optimal solution for the barycentric coordinate $u \in [0, 1]$ with the smallest distance. Using this coordinate, we can compute the penetration depth and contact normal by querying the SDF and its gradient. Penalizing collisions, the contact normal force is computed as the squared penetration depth in the direction of the normal (zero if no collision). In combination with the relative velocity between the knife and the mesh vertices, friction forces are computed following the continuous friction model from Brown [33, Equation 4.5].

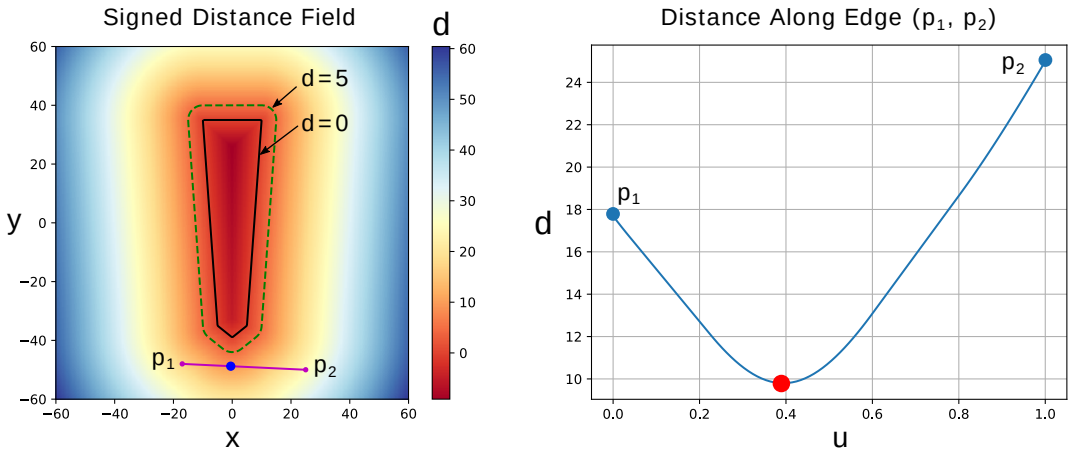


Figure 3.15: 2D slice of the signed distance field (SDF) for the knife shape (not true to scale), where the color indicates distance d. The knife's boundary at $d = 0$ is indicated by a solid black contour line. *Left:* at distances greater than zero the SDF becomes more rounded. *Right:* distance d along the edge (p_1, p_2) with barycentric edge coordinate u varying between 0 (p_1) and 1 (p_2). The closest point found by Algorithm 7 is shown in red. Distances are exemplary and do not represent the actual dimensions used in the simulator (see Appendix B.2).

Analogous to the knife-mesh contact dynamics, we simulate contact forces between the object mesh and the ground that it rests on via the same penalty-based contact model. Here, the forces are computed between mesh vertices, represented by spheres as collision geometry, and the ground represented by a half space. To prevent the object from sliding off the table during the cut, as in [149], we apply boundary conditions that fix mesh vertices in place when they fulfill both the following conditions: (1) touching the ground and (2) being located 1 cm away from the cutting plane.

3.2.3.4 Damage Mechanics

Physically, damage refers to a macroscopic reduction in stiffness or strength of a material caused by the formation and growth of microscopic defects (e.g., voids and microcracks). For fruits and vegetables, which often have limited plastic deformation regimes before failure, damage can be approximated by a reduction in the elastic modulus of the material, or in a discrete mesh-based formulation, in the components of the stiffness matrix.

As follows, our model for damage mechanics leverages the springs that have been introduced in the final step in Section 3.2.3.2. As the knife applies force to the cutting interface, the stiffnesses of the springs that are in contact with the knife, and hence receive knife contact forces, are linearly decreased (see a visualization of this progressive weakening as the knife slides down the cutting interface in Fig. 3.13):

$$k'_e = k_e - \gamma \, \|\mathbf{f}_{\text{knife}}\|, \tag{3.7}$$

where $\mathbf{f}_{\text{knife}}$ is the force the knife applies on the spring (i.e., the contact force that is computed between the knife and the edge), and $\gamma \in [0, 1]$ is a coefficient that controls the "weakness" of the spring (i.e., how easily the material weakens and separates as the knife applies force to it).

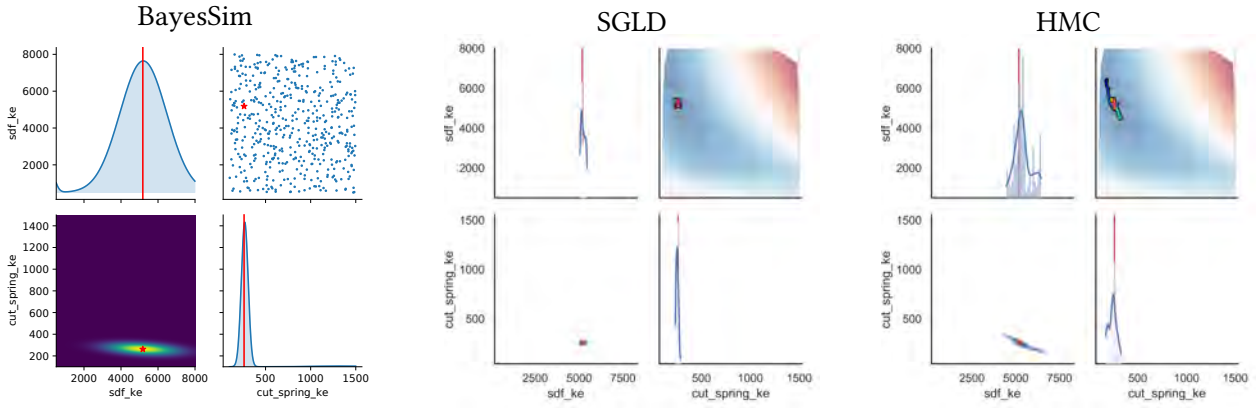


Figure 3.16: Parameter posterior inferred by BayesSim (left), Stochastic Gradient Langevin Dynamics (SGLD) after 90 burn-in iterations (center), and Hamiltonian Monte Carlo (HMC) after 50 burn-in iterations (right) for the two-dimensional parameter inference experiment from Section 3.2.5.1, in which cutting spring stiffness `sdf_ke` and knife contact force stiffness `sdf_ke` need to be estimated. The diagonals of each figure show the marginal densities for the two estimated parameters. The bottom-left sections show a heatmap of the joint distribution. The top-right scatter plot of the BayesSim figure shows the sampled parameters from the training dataset (blue) and the ground-truth (red). The top-right plot sections for SGLD and HMC visualize the loss landscape as a heatmap (where blue means smaller error than red), with the Markov Chain depicted by a colored line. Red lines and stars indicate ground-truth parameter values. SGLD and HMC leverage the gradients of our differentiable simulator and show a much sharper posterior distribution than BayesSim, which uses a dataset of 500 simulated force profiles.

Algorithm 4 Simulation Loop in DiSECt

for each time step i **do**
  Compute gravity and external contact forces $\mathbf{f}_{\text{ext}}$ between mesh and cutting board (half space).
  Compute elastic forces $\mathbf{f}_{\text{elastic}}$ following the constitutive model in Section 3.2.3.1.
  Compute knife contact forces $\mathbf{f}_{\text{knife}}$ as described in Section 3.2.3.3.
  Update cutting spring stiffness $k_e' = k_e - \gamma \left\|\mathbf{f}_{\text{knife}}\right\|$.
  Compute cutting spring forces $\mathbf{f}_{\text{spring}}$.
  Semi-implicit Euler integration of mesh vertices:
    $\mathbf{v}^{t+1} \leftarrow \mathbf{v}^t + \Delta t \mathbf{M}^{-1}(\mathbf{f}_{\text{knife}} + \mathbf{f}_{\text{spring}} + \mathbf{f}_{\text{elastic}} + \mathbf{f}_{\text{ext}})$
    $\mathbf{x}^{t+1} \leftarrow \mathbf{x}^t + \Delta t \mathbf{v}^{t+1}$
  Euler integration of knife velocity (prescribed velocity trajectory).
end for

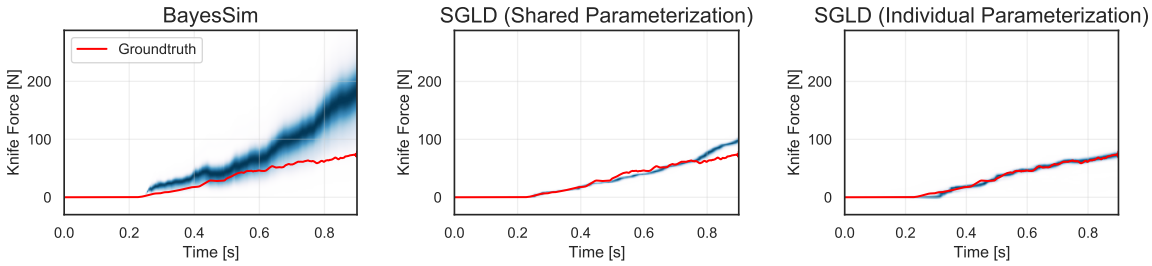


Figure 3.17: Kernel density estimation from 20 knife force trajectories rolled out by sampling from the posterior found by BayesSim (left) and SGLD applied in our differentiable simulator with shared (center) and individual parameterization (right). Shown in red is the ground-truth trajectory from a commercial simulation of cutting an apple with a hemispherical shape. Areas of higher density are shaded in dark blue.

We use a simulation time step of $\Delta t = 10^{-5}$s across all our experiments to accommodate the simulation of and stiff materials (such as apples or potatoes) and centimeter-scale meshes obtained by laser scans of real biomaterials. Our full simulation loop is described in Algorithm 4.

3.2.4 Simulation Parameter Inference

DiSECt supports two modes of parameterizing the simulator: the cutting spring parameters shown in Appendix Tab. B.2 can be shared across all springs, or tuned individually per spring. The shared parameterization is low-dimensional since only one scalar per parameter type needs to be inferred. In contrast, when the cutting springs are individually parameterized, hundreds of variables need to be tuned.

While it is often possible to hand-tune parameters of low-dimensional, phenomenological models, the task of identifying the simulation parameters that can enable close prediction of real-world measurements is daunting, particularly for complex models such as ours with individually tuned spring parameters. To tackle the simulation calibration problem, we leverage automatic differentiation and GPU acceleration to efficiently compute gradients for all the parameters of our simulator. This allows us to use optimization techniques, such as stochastic gradient descent, to directly compute point estimates for the parameters. Moreover, we can leverage modern Bayesian inference methods and estimate posterior distributions for the high-dimensional parameter set given physical observations, such as the force profile of the knife while cutting real foodstuffs. The result is a simulator with the capacity to identify its own uncertainty about the physical world, leading to more robust simulations. We follow a conventional Bayesian approach and define $p(\theta)$ as the prior distribution over simulation parameters (see Appendix Tab. B.2) which, in our experiments, is a uniform distribution, and $p(\phi^r \mid \theta)$ as the likelihood function given by $p(\phi^r \mid \theta) = \exp\{-\|\phi^s_\theta - \phi^r\|_L\}$, where ϕ^r corresponds to real trajectory observations such as knife forces, positions and velocities. ϕ^s_θ is the equivalent simulated trajectory, and $\|\cdot\|_L$ is the L-norm. In our experiments we use the $L1$, which we determined the most effective (see Appendix B.1.2). With both prior and likelihood functions, we can compute the posterior as $p(\theta \mid \phi^r) = \frac{1}{Z} p(\phi^r \mid \theta) p(\theta)$, where Z is a normalizing constant, also known as the marginal likelihood.

In the next sections, we describe three techniques for simulator parameter inference. We start with a gradient-based approach that produces point estimates, followed by stochastic gradient Langevin dynamics (SGLD), a popular gradient-based Markov chain Monte Carlo technique that approximates the posterior as a set of particles. Finally, we describe the likelihood-free inference technique known as BayesSim, which does not make use of gradients and is used as a baseline, demonstrating the value of differentiable simulation.

3.2.4.1 Gradient-Based Optimization

As a baseline for probabilistic parameter inference, we present a solution based on stochastic gradient descent using the popular Adaptive Moment Estimation (Adam) optimizer [164] (see Appendix B.1.1), a first-order method that scales the parameter gradients with respect to their running averages and variances. Unlike the Monte Carlo posterior approximation obtained by SGLD, Adam will find a locally optimal point estimate to the parameters that minimizes the expected loss. We define the loss as $l(\theta) = \log p(\phi^r \mid \theta)$ and compute gradients with respect to the simulation parameters θ that minimize the loss between a real trajectory ϕ^r and simulated trajectories ϕ^s.

3.2.4.2 Stochastic Gradient Langevin Dynamics

A popular alternative to Adam for probabilistic inference is the stochastic gradient Langevin dynamics (SGLD) method [332] (Algorithm 9). SGLD combines the benefits of having access to parameter gradients with well-established sampling-based methods for probabilistic inference to significantly scale the parameter set to high dimensions at a tractable computational cost. The method can be seen as an iterative stochastic gradient optimization approach with the addition of Gaussian noise which is scaled by a preconditioner factor at every iteration. Given a sequence of trajectories generated by our simulator and a sequence of observed trajectories, $\Phi = \{\phi_i^s, \phi_i^r\}_{i=1}^N$, we can write the posterior distribution as

$$p(\theta \mid \Phi) \propto p(\theta) \prod_{i=1}^{N} p(\phi_i^r \mid \theta). \tag{3.8}$$

SGLD takes the energy function of the posterior denoted by $U(\theta) = -\sum_{i=1}^{N} \frac{1}{N} \log p(\Phi \mid \theta) - \log p(\theta)$ and samples from the posterior using the following rule:

$$\theta_{t+1} = \theta_t - \frac{\alpha}{2} A(\theta_t) \nabla U(\theta_t) + \eta_t, \tag{3.9}$$

where $\eta_t \sim \mathcal{N}(0, A(\theta_t)\alpha)$, α is the learning rate, and A is a preconditioner. After an initial burn-in phase necessary for the Markov chain to converge, m samples can be stored to recover an approximate posterior given by $p(\theta \mid \Phi) \approx \frac{1}{m}\sum_{i=1}^{m} \delta_{\theta_i}(\theta)$, where $\delta_{\theta_i}(x)$ is the Dirac delta function which is non-zero whenever $x = \theta_i$.

Critical to SGLD's performance is the choice of an appropriate preconditioner. In this work we follow the extension proposed in Li et al. [185] that computes the preconditioner as an approximation of the Fisher information matrix of the posterior distribution given by A. This approximation can then be sequentially updated using the gradient of the energy function. This is a similar process as the popular RMSProp [316]. Specifically, the preconditioner $A \in \mathbb{R}^{m \times m}$ and momentum $V \in \mathbb{R}^m$ are updated as

$$V(\theta_t) = \beta V(\theta_{t-1}) + (1-\beta)\nabla U(\theta_t) \odot \nabla U(\theta_t), \tag{3.10}$$

$$A(\theta_t) = \operatorname{diag}\left(1 \oslash \left(\epsilon + \sqrt{V(\theta_t)}\right)\right), \tag{3.11}$$

where $\epsilon > 0$ is a small diagonal bias (we choose $\epsilon = 10^{-8}$) added to the preconditioner to prevent it from degenerating, and β (which we set to 0.95) is the exponential decay rate of the preconditioner. $\odot$ and $\oslash$ are element-wise multiplication and division operators.

3.2.4.3 Likelihood-free Inference via BayesSim

BayesSim is a likelihood-free technique to estimate the parameters of a derivative-free simulator that implements a forward dynamics model which is used to generate trajectories for the given simulation parameters. The technique consists of learning a conditional density $q(\theta \mid \phi)$ (which BayesSim represents as a Gaussian mixture model), where θ are the simulation parameters, and ϕ are trajectories or summary statistics of trajectories. For details about our particular BayesSim implementation, please see Appendix B.1.3.

3.2.5 Experiments

To evaluate DiSECt, we first use synthetic data from our own simulator to evaluate probabilistic inference algorithms. Next, we identify the simulation parameters (Appendix Tab. B.2) to closely match an industry-standard, high-fidelity simulation where we have access to the nodal forces and precise motion of the vertices resulting from the knife contact. Finally, we leverage real-world experimental data of measured knife force profiles to evaluate our sim2real transfer.

3.2.5.1 Parameter Inference from Synthetic Data

In our first experiment, we investigate how accurate the estimated posterior is, given synthetic force profiles from known simulation parameters. We create a dataset of 500 knife force trajectories by varying two of the parameters using the shared parameterization (Sec. 3.2.4) from our proposed simulator as training domain. We choose such low dimensionality to ensure that we can obtain enough training data to sample from the uniform prior distribution which is crucial to the performance of likelihood-free methods, such as BayesSim.

The two parameters to be estimated are `sdf_ke` (ranging from 500 to 8000), the stiffness of the contact model at the mesh surface, and `cut_spring_ke` (ranging from 100 to 1500), the stiffness of the cutting springs (cf. Section 3.2.3.4) at the beginning of the simulation.

As shown in Fig. 3.16, BayesSim captures the posterior (see heatmap in the lower-left corner of the left subfigure) around the ground-truth parameters of `sdf_ke` = 5100 and `cut_spring_ke` = 200. SGLD (center), however, yields a significantly sharper density estimate around the true values within 90 burn-in iterations, while Hamiltonian Monte-Carlo (HMC), another gradient-based Bayesian estimation algorithm, finds a slightly wider posterior within approximately 50 iterations.

3.2.5.2 Parameter Inference from High-fidelity Simulator

In our first set of experiments where the training data does not stem from our own model, we simulate cutting trajectories using a commercial, explicit dynamics simulator as a ground-truth source. Details on the simulation setup are in Appendix B.2.2. For each simulation, knife force and nodal motion trajectories were extracted. Each simulation was executed across 4 CPUs and took an average of 1941 min ($> 32\,\mathrm{h}$) to complete. For comparison, our simulator produces a cutting trajectory of 1 s duration (with a $1\times10^{-5}\,\mathrm{s}$ simulation time step) within 30 s on an NVIDIA RTX 2080 GPU, and the gradient of the cost function (likelihood) within 90 s.

We estimate the following parameters θ in this experiment: (1) `sdf_ke`, (2) `sdf_kd`, (3) `sdf_kf`, (4) `sdf_mu`, (5) `cut_spring_ke`, (6) `cut_spring_softness`, and (7) `initial_y` (for an explanation see Tab. B.2 in the appendix).

First, we train BayesSim in an iterative procedure where the currently estimated posterior over the simulation parameters is sampled to roll out a new set of trajectories that are added to the training dataset. Starting from 500 trajectories, we repeatedly sample 20 new parameters and refit BayesSim's mixture density network (MDN) to the updated training dataset. After 100 such iterations (i.e., 2000 additional roll-outs), we obtain the posterior shown in appendix Fig. B.7.

For comparison, we train SGLD in DiSECt for 300 iterations with the same parameterization as used for BayesSim, i.e., the parameters are represented by scalars that are shared across all cutting springs. As shown in Fig. 3.17, trajectories sampled from the estimated posterior of SGLD result in a significantly closer fit (center) to the ground-truth knife force profile compared to BayesSim (left). The average mean absolute error (MAE) of the roll-outs with 4.714 N significantly outperforms BayesSim's average MAE of 26.860 N. The gradient-based estimation method achieves such an outcome with less training data, as it only took 300 trajectory simulations, compared to the 2500 trajectories in total that served as the training dataset for BayesSim. If we allow the parameters to be optimized individually for each cutting spring (resulting

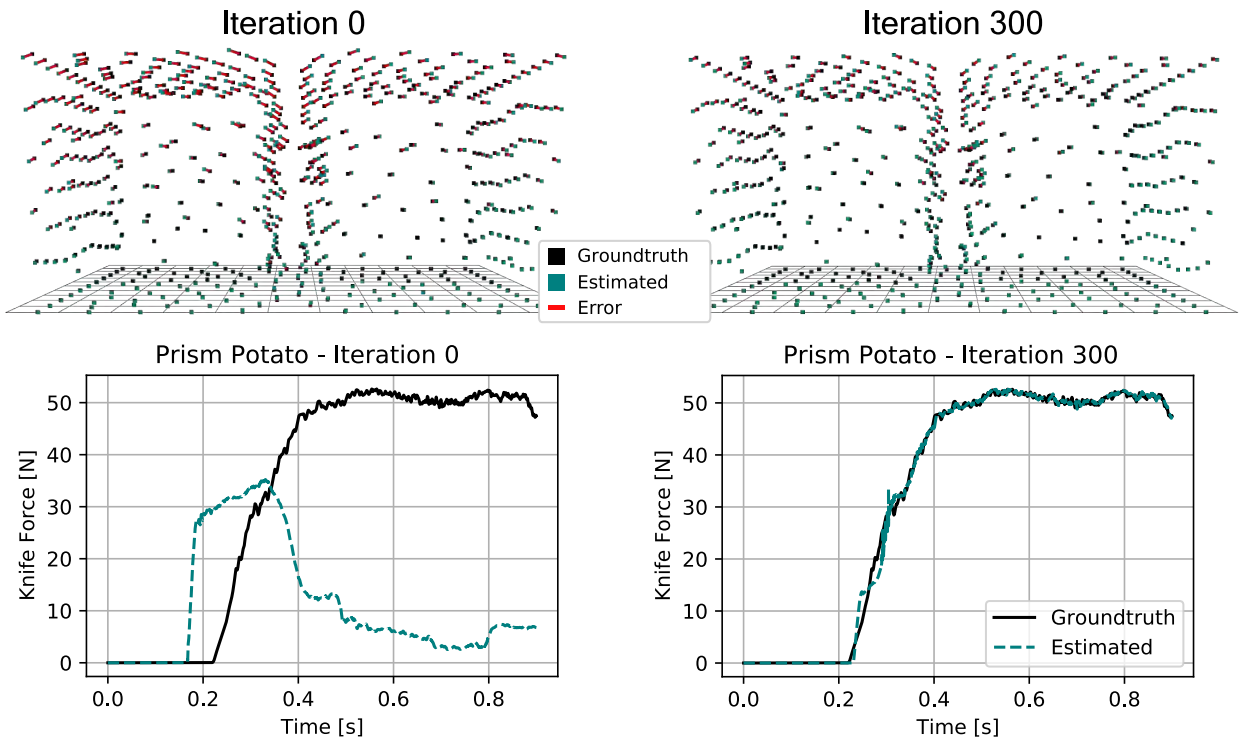


Figure 3.18: Results from simulation parameter optimization given the positions of the vertices (top row) and the knife force (bottom row) with a commercial simulation as ground-truth. *Left:* before optimization, the vertices (top) at the last time step (0.9 s) of the trajectory are visibly distinct between our simulation (blue) and the ground-truth (black), as shown by the red lines indicating the vertex difference. *Right:* after 300 steps with the Adam optimizer, the vertices (top), as well as the knife force profile (bottom), are closer after the parameter inference.

in 1737 parameters in total), the resulting simulation becomes even closer (right), with an average MAE of 3.075 N.

Based on the commercial ground-truth simulation, we collect additional object nodal displacement field trajectories in addition to the knife force profiles, which allows us to leverage another reference signal to calibrate DiSECt. At 18 reference time steps within the trajectory roll-out, we compute the $L2$ error between the positions of the vertices in the ground-truth and our simulator, and add it to the overall cost (which previously only consisted of the $L1$ norm over knife force difference) with a tuned weighting factor. By optimizing the aforementioned parameters individually with the new cost function through Adam, after 300 iterations, we arrive at a simulation that not only closely matches the knife force profile from the ground-truth simulation, but also has a significantly reduced gap in the nodal motions between

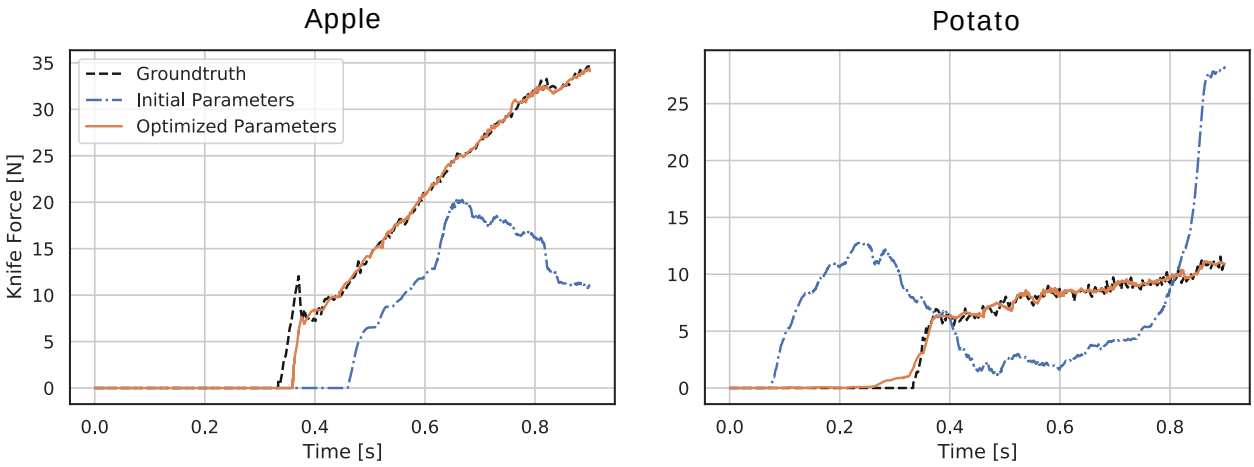


Figure 3.19: Results from optimizing simulation parameters in DiSECt given real-world knife force profiles from vertical cutting of an apple (left) and a potato (right).

the two simulators (see Fig. 3.18). Before the optimization, the mean Euclidean distance between the simulated nodes and the ground-truth nodal positions is 1.554 mm at the last time step ($t = 0.9\,\mathrm{s}$). The low initial error over the nodal motion is explained by the fact that the material properties have been set to already match a potato (Tab. B.1), leaving the cutting-related spring parameters as the only remaining variables to be estimated. After 300 iterations, this error reduced to 1.289 mm, while the knife force profile MAE decreased from 28.366 N to 0.549 N.

3.2.5.3 Parameter Inference from Real World Measurements

In this experiment, we calibrate DiSECt to real-world cutting trajectories. The real-world dataset provided by [149] contains 3D meshes created from laser scans of the actual objects being cut, as well as knife force profiles measured from a force sensor mounted between a robot end-effector and a knife. The knife dimensions are given in Tab. B.2 and explained in Fig. B.3. The triangular surface meshes of the foodstuffs are discretized to tetrahedral meshes via the TetWild [136] meshing library. After optimizing the simulation parameters with Adam for 300 iterations, the simulator closely matches the force profile of a knife cutting an apple (Fig. 3.19 left) with an MAE of 0.253 N, and for a potato cutting action (right) achieves an MAE of

0.379 N. We note that the optimization is stable without requiring restarts from different parameter configurations, and we found even relatively poor parameter initialization (such as the one shown in Figure 3.19) results in highly accurate prediction after the calibration.

3.2.5.4 Generalization

While our simulator is able to calibrate itself very closely to a variety of sources of ground-truth signals – whether these are knife force profiles obtained by a real robot cutting foodstuffs, or motion recordings from the mesh vertices in a commercial simulator that is entirely different from ours (see Appendix B.2.2) – the question of overfitting arises. In the following, we investigate how well the identified simulation parameters transfer to test regimes that differ from the training conditions under which these parameters were optimized.

Generalization to longer simulations As shown in Tab. B.2, DiSECt allows various dynamics parameters to be defined individually for each spring, resulting in hundreds of degrees of freedom. The larger parameterization provides more opportunity to find closely matching solutions, but may be prone to overfitting to the reference trajectory in certain settings. One of the pathological cases occurs when the goal is to predict the knife force trajectory for a duration longer than the time window seen during training from the reference force profile. In the experiment shown in Fig. B.4, we optimize the simulation parameters using Adam on the first 0.4 s section from the reference trajectory. Within that segment, the individual parameterization clearly outperforms the shared parameterization with a MAE of 0.306 N versus 0.384 N. However, when we test the estimated parameters on a simulation with a duration of 0.9 s, the shared parameterization achieves a closer fit with a 4.802 N MAE, compared to the individual tuning with a 5.921 N MAE. Intuitively, as the knife slices downward with constant velocity (at 50 mm s^{-1}), fewer cutting springs are affected by the contact dynamics during a shorter roll-out since the knife does not progress far enough to reach the cutting springs closer to the ground. Hence, their parameters' gradients were zero during the

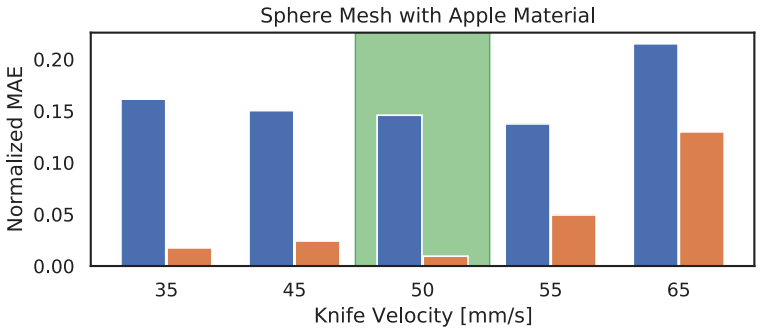


Figure 3.20: Velocity generalization results for a sphere shape with apple material properties given ground-truth simulations with different vertical knife velocities from a commercial simulator. Two versions of DiSECt were calibrated: by sharing the parameters across all cutting springs (blue) and by optimizing each value individually (orange), given a ground-truth trajectory with the knife sliding down at 50 mm s^{-1} speed (highlighted in green). The normalized mean absolute error (MAE) is evaluated against the ground-truth by rolling out the estimated parameters for the given knife velocity.

estimation. Tuning the same kind of parameters uniformly allows all cutting springs to have an improved fit over the initialization, even when their interaction with the knife only becomes apparent at a later time.

Generalization to different knife velocities We investigate how accurately DiSECt predicts knife force profiles given the parameters that were inferred from a cutting trajectory with a knife downward velocity of 50 mm s^{-1}. At test time, we change the downward velocity to 35, 45, 55, and 65 mm s^{-1}. As shown in Fig. 3.20 (and Fig. B.5 in the appendix), the individual parameterization significantly outperforms the shared parameterization in normalized mean absolute error (NMAE), i.e., the MAE between the ground-truth and estimated trajectory divided by the mean force of the ground-truth trajectory, achieving a four-fold more accurate result compared to the shared parameterization in many cases (see example knife force profile in Fig. 3.21).

Generalization to different geometries Cutting the same type of biomaterial can lead to drastically different knife force profiles, even when all the variables that influence the motion of the knife remain the same [149]. This can be caused by different geometries even within the same object class, as no two fruits or vegetables of the same type are identical. Since the mesh topologies can differ significantly between the

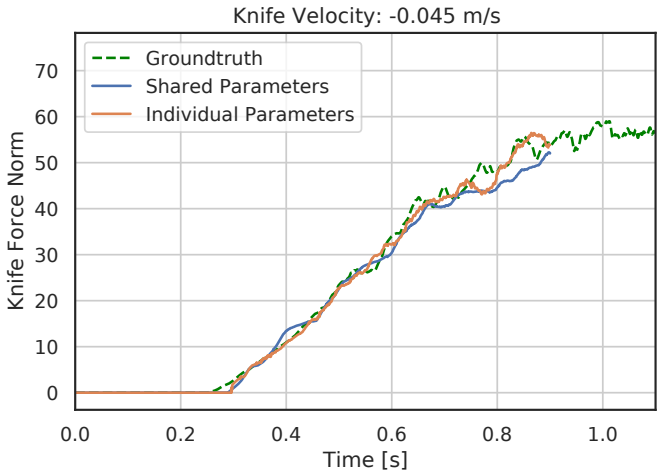


Figure 3.21: Knife force profiles for cutting a cylindrical mesh with cucumber material properties by simulating the parameters with $45\mathrm{mm\,s^{-1}}$ knife velocity downwards. The simulation parameters in the shared and individual parameterization have been inferred from a ground-truth trajectory with $50\mathrm{mm\,s^{-1}}$ knife velocity from a commercial solver (see Sec. 3.2.5.4).

different geometric shapes that a foodstuff may have, a direct mapping between the virtual nodes (respective cutting springs) is not possible, which would allow the transfer of the individual parameters. Instead, we propose a weighted mapping of a combination of cutting spring parameters from the source mesh that are in proximity to the cutting springs of the target mesh. We developed an optimal transport [258] method that receives as inputs the cutting spring coordinates in 2D (obtained by the mesh preprocessing step from Section 3.2.3.2) at the cutting interface (shown in Fig. 3.23) from a source domain, and a different set of 2D coordinates for the target domain. By minimizing the Earth Mover's Distance (EMD) [284] between the cutting spring vertices of the two meshes[‡], we find a weight matrix that allows us to compute spring parameters for the target domain as a weighted combination of the parameters from the source mesh. For more details, see Sec. B.1.4. Similar to the velocity generalization experiments, we observe a significantly improved NMAE performance (Tab. 3.1) in most cases when the cutting spring parameters are tuned individually ("NMAE OT") in contrast to duplicating average of each spring parameter ("NMAE Avg") from the source domain across all locations in the target domain.

[‡]Our implementation uses the Python Optimal Transport library [82]

Material	Source Mesh	Target Mesh	NMAE OT	NMAE Avg
Potato	Real Potato 1	Real Potato 2	0.948	5.635
Potato	Real Potato 2	Real Potato 1	1.360	6.981
Apple	Real Apple 2	Real Apple 3	6.844	1.330
Apple	Real Apple 3	Real Apple 2	3.857	19.749
Potato	Cylinder	Prism	3.470	12.001
Potato	Prism	Cylinder	0.933	1.983
Potato	Prism	Sphere	7.867	15.841
Potato	Sphere	Prism	35.347	16.812
Apple	Cylinder	Sphere	4.261	14.457
Apple	Sphere	Cylinder	1.839	1.602
Apple	Prism	Sphere	0.920	4.531
Apple	Sphere	Prism	13.883	45.983
Cucumber	Cylinder	Sphere	66.672	71.102
Cucumber	Sphere	Cylinder	4.239	0.484
Cucumber	Cylinder	Prism	58.138	64.407
Cucumber	Prism	Cylinder	1.046	1.855

Table 3.1: Mesh generalization results when the parameters from the source domain are transferred to the target domain via Optimal Transport (OT), and by averaging the parameters across all cutting springs. The numbers show the normalized mean absolute error (NMAE), i.e., the MAE divided by the mean of the respective ground-truth knife force profile, to make the results comparable across different material properties and geometries.

Overall, our generalization experiments have shown that, although we optimize hundreds of parameters involved in the cutting dynamics at highly localized places, such representation still generalizes between various conditions. The successful transport of these parameters between two topologically different meshes based on their spatial correspondences indicates that our simulation parameters implicitly encode material properties that generalize across mesh topologies.

3.2.5.5 Controlling Knife Velocity

In real-world applications, automated cutting of foodstuffs may need to meet multiple competing objectives. In particular, force may be minimized to prevent peripheral damage to the object, or ensure human safety, while the velocity may be maximized to reduce the required time. For cutting of food and biomaterials, humans have intuitively developed the strategy of minimizing cutting force by pressing the knife vertically and simultaneously slicing horizontally (i.e., a sawing motion) [13, 58, 57]. Such a cutting action

can be intuitively understood as follows: by definition, the work applied by the knife to the object is equal to the cutting force integrated over displacement, and by conservation of energy, also equal to the fracture energy required to introduce a cut in the vertical plane. By simultaneously slicing horizontally, the distance traveled by the knife will be greater, reducing the cutting force for the required fracture energy.

3.2.5.6 Knife Motion Trajectory Optimization

We represent the knife velocity trajectory by k equidistant keyframes in time. Three parameters are to be optimized per keyframe i (refer to Fig. B.3 for the coordinate frame orientation w.r.t. the knife):

- a_i: the amplitude of the lateral (along z axis) sinusoidal velocity
- b_i: the frequency of the lateral sinusoidal velocity
- c_i: the vertical (along y axis) velocity

To allow for a smooth interpolation between the keyframes, and propagation of gradients from all trajectory parameters at every time step, we weight the contribution of all keyframe parameters on the entire trajectory via the radial basis function (RBF) kernel. The kernel uses the squared norm of the difference between the current time t difference and the predefined keyframe times to compute the weight contributions $\mathbf{w} \in \mathbb{R}^k$ of the keyframe parameters (see Fig. 3.22):

$$\mathbf{w}(t) = \exp\left(-\frac{\|t - w\|^2}{2\sigma^2}\right) \tag{3.12}$$

In effect, a nonzero contribution on the trajectory is maintained from all keyframes at all times, which eases gradient-based optimization.

The kernel width σ controls how smoothed out the contributions of the keyframe parameters become. We found $\sigma = \sqrt{0.03}$ to be an appropriate setting (see illustration in Fig. 3.22), given that the duration of the cutting action we optimize for is 0.9 s using $k = 5$ keyframes.

To compute the knife's horizontal (sideways) and vertical velocities $\dot{z}_{\text{knife}}(t)$ and $\dot{y}_{\text{knife}}(t)$ at time $t = [0..T]$, the keyframe parameters in vector form $\mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathbb{R}^k$ are combined with the time-dependent weighting contribution from Eq. 3.12:

$$\dot{z}_{\text{knife}}(t) = \mathbf{a} \cdot \mathbf{w}(t) \cos(\mathbf{b} \cdot \mathbf{w}(t)t) \tag{3.13}$$

$$\dot{y}_{\text{knife}}(t) = \mathbf{c} \cdot \mathbf{w}(t) \tag{3.14}$$

We minimize the mean knife force plus the vertical knife velocity integrated over the entire length of the trajectory. Thus, we penalize high actuation effort by the robot controlling the knife while maintaining a fast progression of the cutting process:

$$\underset{\mathbf{u}=[\mathbf{a},\mathbf{b},\mathbf{c}]}{\text{minimize}} \qquad \mathcal{L} = \frac{1}{T} \int f(t, \mathbf{a}, \mathbf{b}, \mathbf{c}) + \dot{y}_{\text{knife}}(t)\, dt \tag{3.15}$$

$$\text{s.t.} \qquad y_{\text{knife}}(T) = h_{\text{end}} \tag{3.16}$$

$$|z_{\text{knife}}(t)| \leq \frac{1}{2} l_{\text{knife}}, \tag{3.17}$$

where $k = 5$ is the number of keyframes, $T = 0.9\,\text{s}$ is the time at the end of the trajectory, f is the simulation step that returns the knife force norm $\|\mathbf{f}_{\text{knife}}\|$ at time step t given the trajectory parameter vectors $\mathbf{a}, \mathbf{b}, \mathbf{c}$, and $l_{\text{knife}} = 15\,\text{cm}$ is the blade length of the knife. We impose the hard constraint in Eq. 3.17 to ensure that the knife does not move too far along z where the blade of the knife ends (which would trivially minimize the knife force, although we assume the knife blade has infinite length in this experiment setup).

Constrained optimization problems are typically solved by converting them to unconstrained optimization problems through the introduction of Lagrange multipliers. However, the critical points to such Lagrangians often tend to be saddle points, which gradient-descent-style algorithms, such as Adam, will

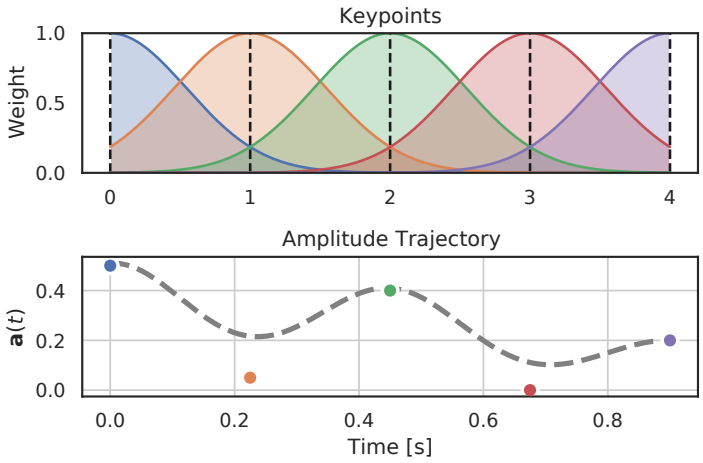


Figure 3.22: Visualization of five keypoints (top) evenly distributed in time with exemplary amplitude values $\mathbf{a}[i]$ per keyframe i (color-matching dots in lower plot). The resulting continuous trajectory $\mathbf{a}(t)$ resulting from weighting the keyframes via the RBF kernel (Eq. 3.12) is shown as the dashed line at the bottom.

not converge to [260]. To make the unconstrained objective amenable to gradient descent, following the modified differential method of multipliers (MDMM) [260], we introduce a penalty term for $\mathbf{u} = [\mathbf{a}, \mathbf{b}, \mathbf{c}]$:

$$E_{\text{penalty}} = \frac{c}{2}(g(\mathbf{u}))^2.$$

This term acts as an attractor to the energy function that we are optimizing for (where c acts as a damping factor), where $g(\mathbf{u})$ is an equality constraint.

To include the inequality constraint in Eq. 3.17, we convert it to an equality constraint $g(\mathbf{x})$ by introducing slack variable $\gamma \in \mathbb{R}$ that becomes part of $\mathbf{u}$:

$$g(\mathbf{u}) = \frac{1}{2}l_{\text{knife}} - |z_{\text{knife}}(t)| - \gamma^2. \tag{3.18}$$

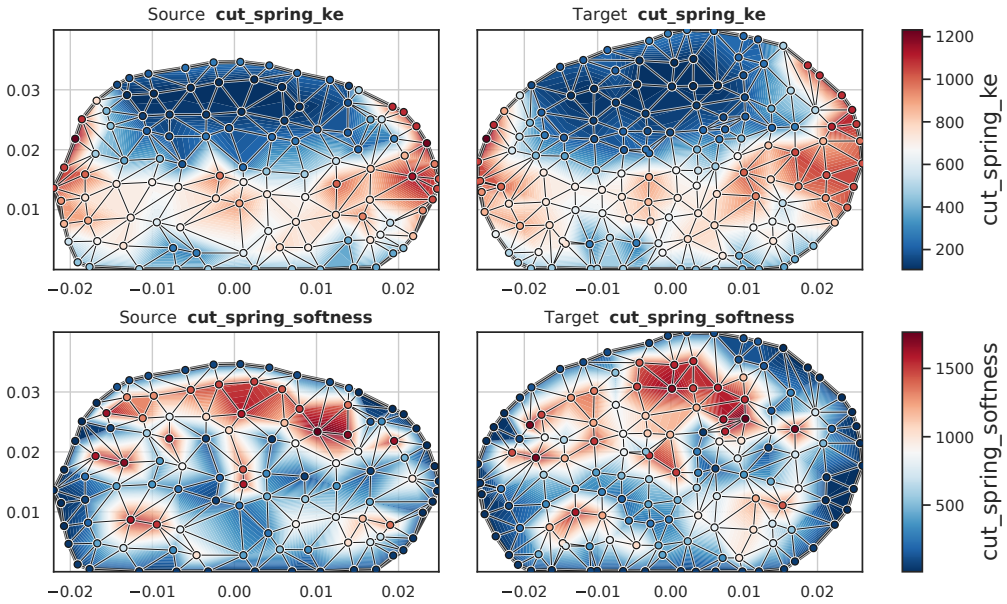


Figure 3.23: Transfer of cutting parameters between two different potato meshes from a real-world cutting dataset. For the model in the left column, the cutting spring parameters have been optimized individually for each cutting spring given a single trajectory of the knife force (only two of the parameter types are shown in both rows). These parameters have been transferred to the mesh on the right column via Optimal Transport with the Earth Mover's Distance (EMD) objective (more details for this mesh transfer example are shown in Fig. B.2).

The update rule for the trajectory parameters $\mathbf{u}$ is then

$$\mathbf{u}' = \mathbf{u} - \frac{\partial \mathcal{L}}{\partial \mathbf{u}} - \lambda \frac{\partial g}{\partial \mathbf{u}} - cg(\mathbf{u}) \frac{\partial g}{\partial \mathbf{u}} \tag{3.19}$$

$$\lambda' = \lambda + g(\mathbf{u}). \tag{3.20}$$

Using gradient-based trajectory optimization, we observe that such an intuitive cutting strategy emerges. We define the cost function in Eq. 3.15 to penalize the mean knife force and inverse velocity, and we parameterize the knife trajectory via keyframes in time that define the downward velocity and sinusoidal time-varying horizontal velocity (a complete description is given in Section 3.2.5.6 of the appendix).

When optimizing Eq. 3.15 through Adam without constraints on the sideways knife position, the resulting motion after 50 iterations (orange line in right subplot of Fig. 3.24) consists of an initial pressing phase up to the point of contact with the cucumber, after which the knife continuously moves sideways

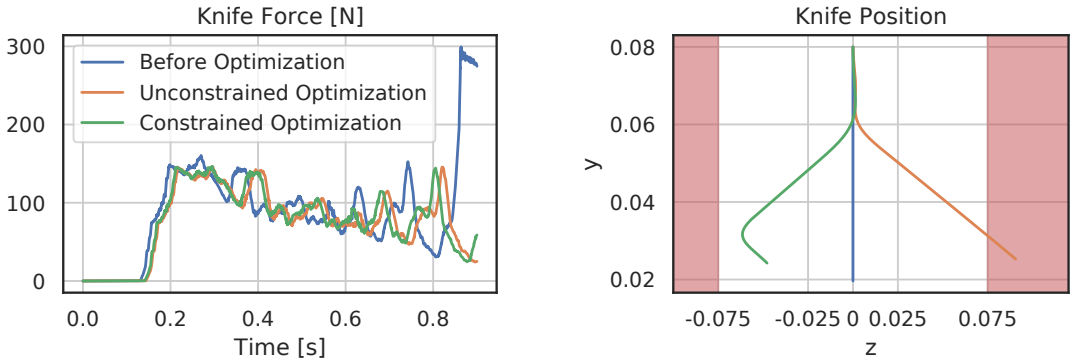


Figure 3.24: Results from the trajectory optimization experiment (Section 3.2.5.5) on the cylinder mesh with cucumber material properties where the mean knife force is penalized and the overall downward velocity maximized. *Left:* knife force profiles before the trajectory optimization (blue), after unconstrained (orange) and constrained (green) optimization. *Right:* resulting knife motions (starting from $y = 8\,\mathrm{cm}$ moving downwards), with constraints on the lateral motion shaded in red.

Figure 3.25: Watertight 3D meshes obtained from 3D scans of a real apple, a peeled banana, and a cucumber, as part of the real-robot experiments from Sec. 3.2.6. The textured meshes have been visualized in Blender via the Cycles renderer. For simulation, these meshes are downsampled and tetrahedralized.

without sawing. To limit the sideways motion to remain within the bounds of the blade length, we add an inequality constraint in Eq. 3.17. By performing constrained optimization with the modified differential method of multipliers (MDMM) (see Section 3.2.5.6), we see that the knife moves within the bounds of the 15 cm blade length (green line on the right in Fig. 3.24), while incurring only slightly more mean knife force (76.726 N) compared to the solution from the earlier unconstrained optimization (76.498 N). For comparison, the mean knife force was 89.698 N before the trajectory optimization.

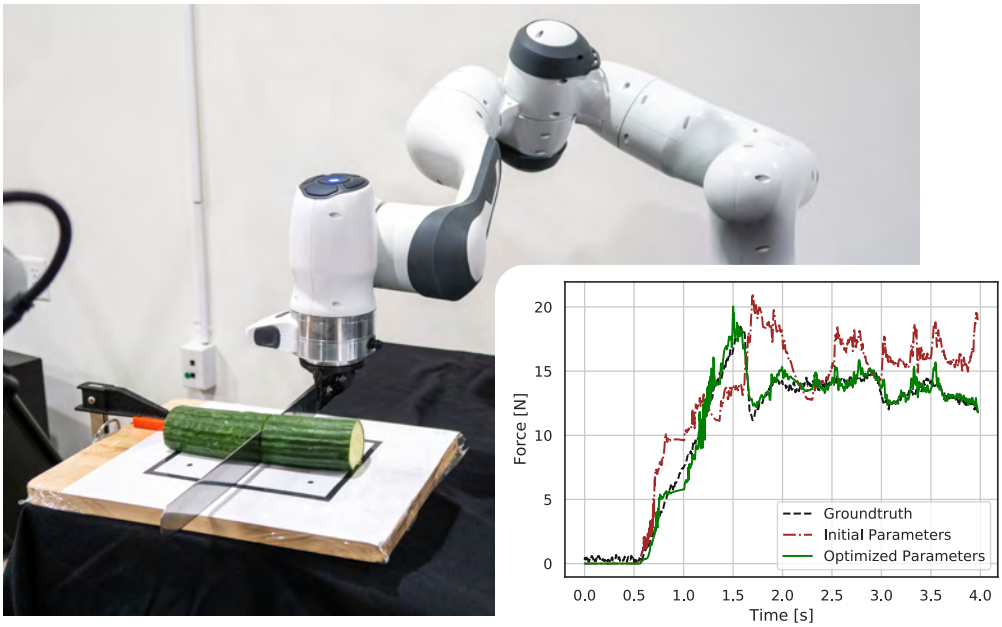


Figure 3.26: *Left:* Experimental setup of our real-robot experiments where a Panda robot arm equipped with the slicing knife has cut vertically through a piece of cucumber. *Right:* Force profile obtained from cutting this cucumber vertically (black), compared against the simulated force profile from before (red) and after (green) optimization of the simulation parameters.

3.2.6 Real-robot Optimized Cutting Motion

In our final set of experiments, we validate the capabilities of our simulator to infer the simulation parameters and optimize the knife motion trajectory on a real robot. We designed and 3D-printed a knife fixture for a thin slicing knife with a blade length of 25 cm that attaches as end-effector to a Franka Emika Panda robot arm (see Fig. 3.26). Before slicing, we obtain highly detailed meshes of all the fruits to be cut via the turn-table-based EinScan SE 3D scanner (see Fig. 3.25). We generate watertight surface meshes and downsample them before discretization with TetMeshWild [136] to obtain adequate tetrahedral meshes used in our simulation. While force-torque sensors attached to the end-effector, such as in the experimental setup from [149], may provide force measurements of higher accuracy, we found the linear force calculated from the joint forces measured by the torque sensors of the Panda arm very comparable in quality to the previously used dataset of real force profiles in Sec. 3.2.5.

3.2.6.1 Parameter Inference

Starting with a piece of cucumber, we command the robot arm via Cartesian velocity control to cut in a straight line vertically with a velocity of $4\,\mathrm{cm\,s^{-1}}$ downwards starting from a height of 11 mm of the blade above the cutting board (see Fig. 3.26). We continuously calculate the linear forces acting on the knife from the measurements of the torque sensors of the robot arm given the kinematic and inertial properties of the knife and its fixture computed by the Autodesk Fusion 360 CAD software. As before, we take the norm of the linear knife force at each time step as our ground-truth signal. Given such force profile, we estimate the simulation parameters leading to a significantly reduced reality gap to the simulated knife force (the mean force error at each time step declined from over 14 N to less than 1 N).

Within 200 iterations of the Adam optimizer, we find simulation parameters that yield a simulated force profile which closely matched the real measurements (see the force plot on the right of Fig. 3.26). We repeat the same cutting motions on an apple and a peeled banana, which are fruits that are on the extreme ends of stiffness and toughness within the foodstuffs we investigated in this work. As shown in Figures 3.27 and 3.28 for the apple and the banana, respectively, the optimized simulation parameters achieve a significantly lower reality gap in the generated force profiles than at the beginning of the simulation (the force profile given the initial parameters is plotted in red).

We quantify the accuracy in the simulated force profiles via the normalized mean absolute error metric (NMAE) again, where the MAE between the simulated and real force profile is divided by the mean of the real force profile. As reported in the first two rows of Tab. 3.2, the NMAE shrunk significantly between the initial setting of the simulation parameters and the final result of the optimization.

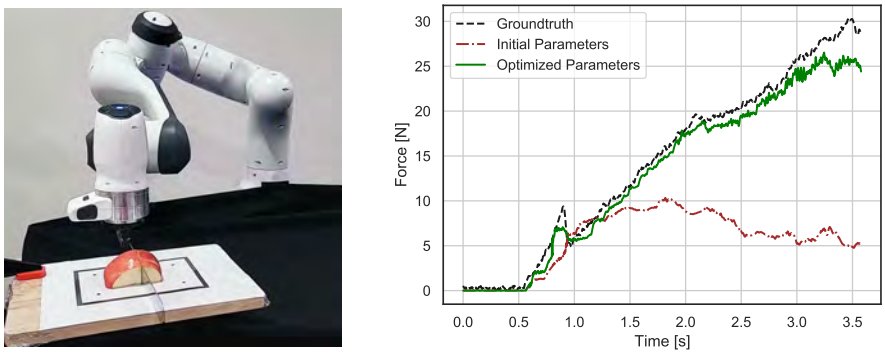


Figure 3.27: Vertical cutting of an apple.

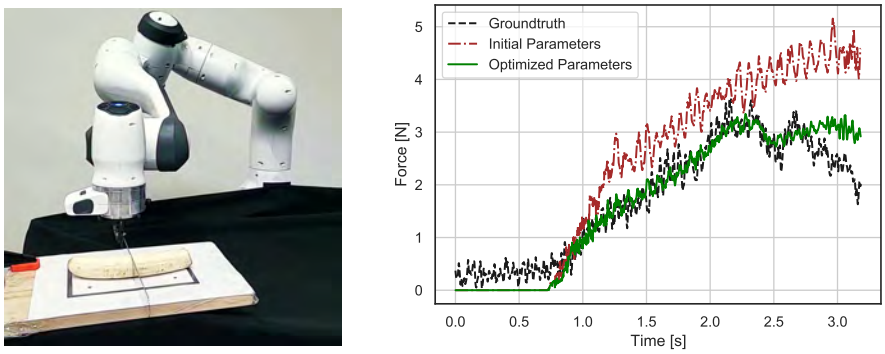


Figure 3.28: Vertical cutting of a peeled banana.

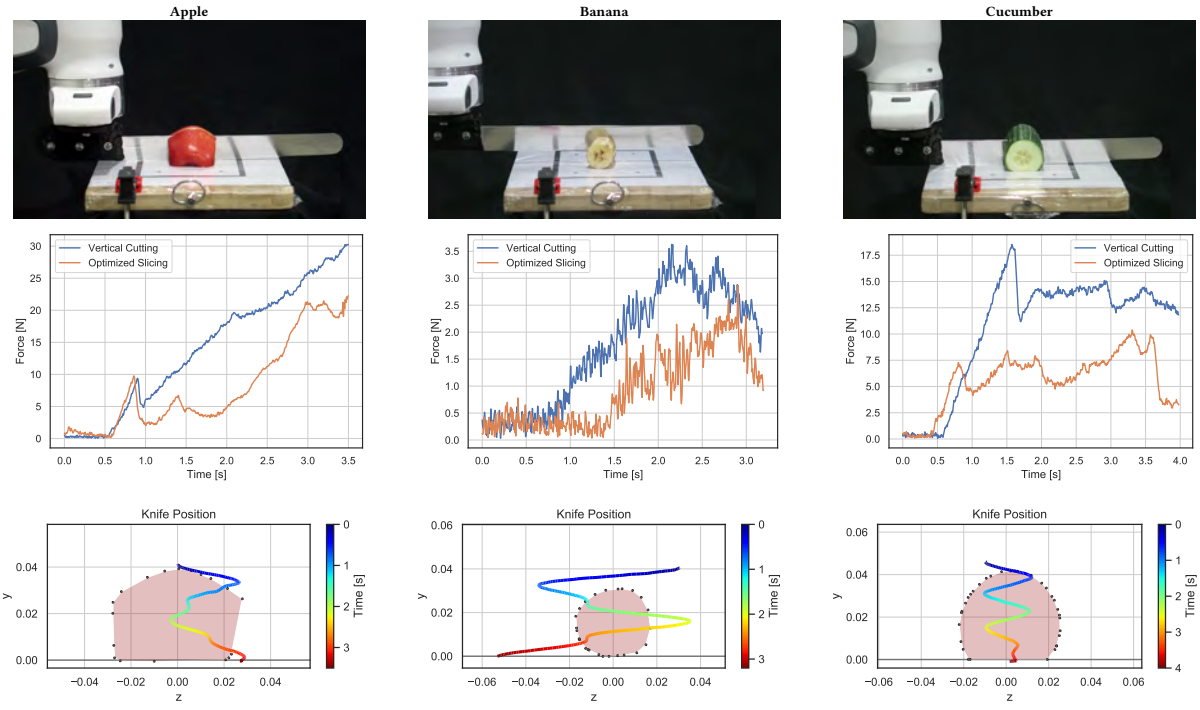


Figure 3.29: Real-robot executions of the optimized slicing motions for an apple (left column), a peeled banana (center column), and a cucumber (right column). *First row:* side view of the experimental setup. *Second row:* comparison of measured knife force profiles from a vertical cut (blue) and the optimized slicing motion (orange). *Third row:* trajectory of the knife following the optimized slicing motion. The colorful line indicates the position of the blade center point over time, from the first (dark blue) to the last time step (dark red) of the motion. The shaded polygon behind the trajectory line indicates the silhouette of the mesh being cut (orthographic projection of the convex hull of the fruit's shape).

3.2.6.2 Trajectory Optimization

Having optimized the simulation parameters from measurements obtained through a vertical cut, we now seek to find cutting motions that require less effort from the robot and thereby exert less force on the material which potentially damages the fruit. To this end, we select a half of the mesh that we cut previously as our new cutting target where we aim to optimize a lateral slicing motion, similar to our setup from Section 3.2.5.5. Since the mesh topology has changed between the previous vertical cut and the new part of the fruit that we want to slice, we transport the previously optimized simulation parameters to the new cutting surface via the optimal transport technique we developed in Sec. B.1.4.

Given the simulation setup and transported cutting spring parameters, we are now ready to optimize the lateral slicing motion of the knife. We follow our formulation from Eq. 3.15 where we parameterize the motion through five way points that encode the slicing amplitude and vertical velocity. At the same time, we again enforce the constraints to (1) ensure the knife touches the cutting board at the end of the cut, and (2) that the lateral motion of the knife does not exceed the blade length. We execute the optimized slicing motions on the real robot, as shown in Fig. 3.29, for the apple, banana, and cucumber. The fruits are fixed to the cutting board via water-resistant super glue to ensure that they remain in place without the need of an external fixture. Analogously, in simulation, we also add boundary conditions to the lowest mesh vertices that touch the cutting board to keep them at a fixed location.

With an initialization of the slicing amplitude to zero, which corresponds to a vertical cut, the optimized motions exhibit a pronounced slicing trajectory (see bottom row in Fig. 3.29). Particularly for the banana (middle column), the knife moves laterally by more than ten centimeters, while the lateral motions on the apple and cucumber remain smaller. In the measured forces, we see a clear benefit of the obtained slicing motions. As shown in the second row of Fig. 3.29, the force profile of the optimized knife trajectory (orange) is considerable lower than its counterpart of the vertical cutting motion (blue). We quantify the reduction in exerted knife force in the bottom two rows in Tab. 3.2. The mean of the knife force over the

	Apple	Banana	Cucumber
Simulation accuracy *before* parameter inference			
Knife force NMAE	0.606	0.487	0.231
Simulation accuracy *after* parameter inference			
Knife force NMAE	0.083	0.160	0.050
Trajectory optimization performance			
Mean force reduction	42.7 %	46.5 %	46.7 %
Max force reduction	26.5 %	20.9 %	43.9 %

Table 3.2: Summary of results in parameter inference and trajectory optimization from our real-robot experiments in Sec. 3.2.6.

entire length of the trajectory is consistently reduced by more than 40%, while the peak force (maximum of force profile) has shrunk by 20% (banana) to more than 40% (cucumber). Since the cutting of a peeled banana requires significantly less force than the other fruits, it is expected that the maximal force does not reduce by as much compared to a stiffer material, such as a cucumber or an apple.

3.2.7 Conclusions

We presented the first differentiable simulator for the robotic cutting of soft materials. Differentiability was achieved through a continuous contact formulation, the insertion of virtual nodes along a cutting plane, and a continuous damage model based on the progressive weakening of springs. The advantages of differentiability were shown through a systematic comparison of multiple gradient-based and derivative-free methods for optimizing the simulation parameters; leveraging gradients from the simulator enabled highly efficient estimation of posteriors over hundreds of parameters. The calibrated simulator was able to reproduce ground-truth data from a state-of-the-art commercial simulator in a fraction of the time, as well as closely match data from a real-world cutting dataset. Simulator predictions generalized across cutting velocities, object instances, and object geometries. A control experiment was performed in which human pressing-and-slicing behavior emerged from sample-efficient constrained optimization applied to the differentiable simulator, reducing the mean knife force by 15% relative to a vertical cut. Finally, we applied

the capabilities of parameter inference, parameter transport, and trajectory optimization to a real-robot cutting scenario where the sim2real gap was closed and the measured knife forces reduced significantly.

In future work, we plan to make multiple extensions. First, we will extend the material model for the soft material to explicitly capture nonlinearity and isotropy, as commonly observed in biomaterials; explicitly specifying such behaviors will facilitate optimization of simulator parameters. In addition, we will extend our modeling approach to accommodate more complex cutting actions, such as carving, in which the knife may arbitrarily rotate and follow more diverse trajectories than sawing motions; this approach will be used in additional control experiments, with the resulting trajectories compared again to human actions. Ultimately, we envision the use of differentiable cutting simulators in applications as challenging as robotic surgery, where tissue parameters can accurately and efficiently be estimated on initial contact with the cutting instrument, and the calibrated simulator can be used for faster-than-real-time roll-outs in a model-predictive framework for online surgical planning.

Acknowledgments

We thank Yan-Bin Jia and Prajjwal Jamdagni for providing the dataset of real-world cutting trajectories and meshes that we used in our experiments from Sec. 3.2.5. We are grateful to Krishna Mellachervu for outstanding technical support with commercial solvers. We thank David Millard for helping to set up the robot infrastructure and guiding us in the fabrication of the knife fixture. Finally, we thank Mike Skolones for his mentorship.

Chapter 4

Closing the Reality Gap in Simulators

In the previous chapter we have explored how the reality gap in simulators can be closed through gradient-based optimization of the parameters of a differentiable physics engine. We now focus in detail on the methods that improve the realism of such simulators.

In Sec. 4.1, we develop a particle-based Bayesian inference algorithm [117] that is tailored specifically to finding posterior distributions of simulation parameters given trajectories from a real system. Such probabilistic system identification approach leverages graphical processing units (GPU) to evaluate the gradient of the likelihood function of hundreds of particles in parallel, which enables it to find multimodal parameter distributions.

Besides finding the right parameters for the given simulator, an important source of prediction error lies in the expressiveness of the analytical models the simulator implements. As we saw in Sec. 3.1, by introducing a learned quadratic function into the sensor model to capture the behavior of the photodiode in a LiDAR sensor, such small, highly localized function approximators give the opportunity to overcome deficits in the analytical simulation models. In Sec. 4.2, we develop this approach further and introduce NeuralSim [122], a differentiable framework that augments a rigid-body physics engine by neural networks to learn arbitrary dynamical effects from data.

Finally, we remove several limitations of our previous approaches where state measurements were assumed to be available, as well as the kinematic structure of the mechanism we aim to identify. In Sec. 4.3, we present a pipeline that can learn simulators from depth or RGB videos, while reconstructing the articulations of rigid-body mechanisms [120].

4.1 Probabilistic Inference of Simulation Parameters via Parallel Differentiable Simulation

Reproducing real world dynamics in simulation is critical for the development of new control and perception methods. This task typically involves the estimation of simulation parameter distributions from observed rollouts through an inverse inference problem characterized by multi-modality and skewed distributions. We address this challenging problem through a novel Bayesian inference approach that approximates a posterior distribution over simulation parameters given real sensor measurements. By extending the commonly used Gaussian likelihood model for trajectories via the multiple-shooting formulation, our gradient-based particle inference algorithm, Stein Variational Gradient Descent, is able to identify highly nonlinear, underactuated systems. We leverage GPU code generation and differentiable simulation to evaluate the likelihood and its gradient for many particles in parallel. Our algorithm infers nonparametric distributions over simulation parameters more accurately than comparable baselines and handles constraints over parameters efficiently through gradient-based optimization. We evaluate estimation performance on several physical experiments. On an underactuated mechanism where a 7-DOF robot arm excites an object with an unknown mass configuration, we demonstrate how the inference technique can identify symmetries between the parameters and provide highly accurate predictions.

Figure 4.1: Panda robot arm shaking a box with two weights in it at random locations in our parallel differentiable simulator (left), physical robot experiment (center), and the inferred particle distribution using our proposed method over the 2D positions of the two weights inside the box (right).

4.1.1 Introduction

Simulators for robotic systems allow for rapid prototyping and development of algorithms and systems [167], as well as the ability to quickly and cheaply generate training data for reinforcement learning agents and other control algorithms [6]. In order for these models to be useful, the simulator must accurately predict the outcomes of real-world interactions. This is accomplished through both accurately modeling the dynamics of the environment as well as correctly identifying the parameters of such models. In this work, we focus on the latter problem of parameter inference.

Optimization-based approaches have been applied in the past to find simulation parameters that best match the measured trajectories [42, 274]. However, in many systems we encounter in the real world, the dynamics are highly nonlinear, resulting in optimization landscapes fraught with poor local optima where such algorithms get stuck. Global optimization approaches, such as population-based methods, have been applied [122] but are sample-inefficient and cannot quantify uncertainty over the predicted parameters.

In this work we follow a probabilistic inference approach and estimate belief distributions over the the most likely simulation parameters given the noisy trajectories of observations from the real system. The relationship between the trajectories and the underlying simulation parameters can be highly nonlinear, hampering commonly used inference algorithms. To tackle this, we introduce a multiple-shooting formulation to the parameter estimation process which drastically improves convergence to high-quality solutions. Leveraging GPU-based parallelism of a differentiable simulator allows us to efficiently compute likelihoods and evaluate its gradients over many particles simultaneously. Based on Stein Variational Gradient Descent (SVGD), our gradient-based nonparametric inference method allows us to optimize parameters while respecting constraints on parameter limits and continuity between shooting windows. Through various experiments we demonstrate the improved accuracy and convergence of our approach.

Our contributions are as follows: first, we reformulate the commonly used Gaussian likelihood function through the multiple-shooting strategy to allow for the tractable estimation of simulation parameters

from long noisy trajectories. Second, we propose a constrained optimization algorithm for nonparametric variational inference with constraints on parameter limits and shooting defects. Third, we leverage a fully differentiable simulator and GPU parallelism to automatically calculate gradients for many particles in parallel. Finally, we validate our system on a simulation parameter estimation problem from real-world data and show that our calculated posteriors are more accurate than comparable algorithms, as well as likelihood-free methods.

4.1.2 Related Work

System identification methods for robotics use a dynamics model with often linearly dependent parameters in classical time or frequency domain [325], and solve for these parameters via least-squares methods [172]. Such estimation approaches have been applied, for example, to the identification of inertial parameters of robot arms [160, 10, 37, 325] with time-dependent gear friction [101], or parameters of contact models [326, 74]. Parameter estimation has been studied to determine a minimum set of identifiable inertial parameters [89] and finding exciting trajectories which maximize identifiability [7, 90]. More recently, least-squares approaches have been applied to estimate parameters of nonlinear models, such as the constitutive equations of material models [210, 109, 241], and contact models [168, 181]. In this work, we do not assume a particular type of system to identify, but propose a method for general-purpose differentiable simulators that may combine multiple models whose parameters can influence the dynamics in highly nonlinear ways.

Bayesian methods seek to infer probability distributions over simulation parameters, and have been applied to infer the parameters of dynamical systems in robotic tasks [330, 238, 312] and complex dynamical systems [244]. Our approach is a Bayesian inference algorithm which allows us to include priors to find posterior distributions over simulation parameters. The advantages of Bayesian inference approaches

have been shown to be useful in the areas of uncertainty quantification [244, 257], system noise quantification [317] and model parameter inference [265, 55].

Our method is designed for differentiable simulators which have been developed recently for various areas of modeling, such as articulated rigid body dynamics [14, 137, 91, 266, 122, 181], deformables [139, 137, 239, 121, 140] and cloth simulation [190, 137, 266], as well as sensor simulation [243, 119]. Certain physical simulations (e.g. fracture mechanics) may not be analytically differentiable, so that surrogate gradients may be necessary [110].

Without assuming access to the system equations, likelihood-free inference approaches, such as approximate Bayesian computation (ABC), have been applied to the inference of complex phenomena [320, 249, 274, 135, 214, 215]. While such approaches do not rely on a simulator to compute the likelihood, our experiments in Sec. A.2.4 indicate that the approximated posteriors inferred by likelihood-free methods are less accurate for high-dimensional parameter distributions while requiring significantly more simulation roll-outs as training data.

Domain adaptation techniques have been proposed that close the loop between parameter estimation from real observation and improving policies learned from simulators [42, 274, 219, 64]. Achieving an accurate simulation is typically not the final objective of these methods. Instead, the performance of the learned policy derived from the calibrated simulator is evaluated which does not automatically imply that the simulation is accurate [177]. In this work, we focus solely on the calibration of the simulator where its parameters need to be inferred.

4.1.3 Formulation

In this work we address the parameter estimation problem via the Bayesian inference methodology. The posterior $p(\theta|D_{\mathcal{X}})$ over simulation parameters $\theta \in \mathbb{R}^M$ and a set of trajectories $D_{\mathcal{X}}$ is calculated using Bayes' rule:

$$p(\theta|D_{\mathcal{X}}) \propto p(D_{\mathcal{X}}|\theta)p(\theta),$$

where $p(D_{\mathcal{X}}|\theta)$ is the likelihood distribution and $p(\theta)$ is the prior distribution over the simulation parameters. We aim to approximate the distribution over true parameters $p(\theta^{\text{real}})$, which in general may be intractable to compute. These true parameters θ^{real} generate a set of trajectories $D_{\mathcal{X}}^{\text{real}}$ which may contain some observation noise. We assume that these parameters are unchanging during the trajectory and represent physical parameters, such as friction coefficients or link masses.

We assume that each trajectory is a Hidden Markov Model (HMM) [286] which has some known initial state, $\mathbf{s}_0$, and some hidden states $\mathbf{s}_t$, $t \in [1..T]$. These hidden states induce an observation model $p_{\text{obs}}(\mathbf{x}_t|\mathbf{s}_t)$. In the simulator, we map simulation states to observations via a deterministic observation function $f_{\text{obs}} : \mathbf{s} \mapsto \mathbf{x}$.

The transition probability $p(\mathbf{s}_t|\mathbf{s}_{t-1}, \theta)$ of the HMM cannot be directly observed but, in the case of a physics engine, can be approximated by sampling from a distribution of simulation parameters and applying a deterministic simulation step function $f_{\text{step}} : (\mathbf{s}, t, \theta) \mapsto \mathbf{s}$. In Sec. A.2.1, we describe our implementation of f_{step}, the discrete dynamics simulation function. The function f_{sim} rolls out multiple steps via f_{step} to produce a trajectory of T states given a parameter vector θ and an initial state $\mathbf{s}_0$: $f_{\text{sim}}(\theta, \mathbf{s}_0) = [\mathbf{s}]_{t=1}^{T}$. To compute measurements from such state vectors, we use the observation function f_{obs}: $\mathcal{X} = f_{\text{obs}}([\mathbf{s}]_{t=1}^{T})$. Finally, we obtain a set of simulated trajectories $D_{\mathcal{X}}^{\text{sim}} = [f_{\text{obs}}(f_{\text{sim}}(\theta, \mathbf{s}_0^{\text{real}}))]$ for each initial state $\mathbf{s}_0^{\text{real}}$ from the trajectories in $D_{\mathcal{X}}^{\text{real}}$. The initial state $\mathbf{s}_0^{\text{real}}$ from an observed trajectory may be acquired via state

estimation techniques, e.g. methods that use inverse kinematics to infer joint positions from tracking measurements.

We aim to minimize the Kullback-Leibler (KL) divergence between the trajectories generated from forward simulating our learned parameter particles and the ground-truth trajectories , while taking into account the priors over simulation parameters:

$$d_{\mathrm{KL}}\left[p(D_{\mathcal{X}}^{\mathrm{sim}}|\theta^{\mathrm{sim}})p(\theta^{\mathrm{sim}}) \parallel p(D_{\mathcal{X}}^{\mathrm{real}}|\theta^{\mathrm{real}})p(\theta^{\mathrm{real}})\right].$$

We choose the *exclusive* KL divergence instead of the opposite direction since it was shown for our choice of particle-based inference algorithm in Liu [195] that the particles exactly approximate the target measure when an infinitely dimensional functional optimization process minimizes this form of KL divergence.

4.1.4 Approach

4.1.4.1 Stein Variational Gradient Descent

A common challenge in statistics and machine learning is the approximation of intractable posterior distributions. In the domain of robotics simulators, the inverse problem of inferring high-dimensional simulation parameters from trajectory observations is often nonlinear and non-unique as there are potentially a number of parameters values that equally well produce simulated roll-outs similar to the real dynamical behavior of the system. This often results in non-Gaussian, multi-modal parameter estimation problems. Markov Chain Monte-Carlo (MCMC) methods are known to be able to find the true distribution, but require an enormous amount of samples to converge, which is exacerbated in high-dimensional parameter spaces. Variational inference approaches, on the other hand, approximate the target distribution by a simpler, tractable distribution, which often does not capture the full posterior over parameters accurately [29].

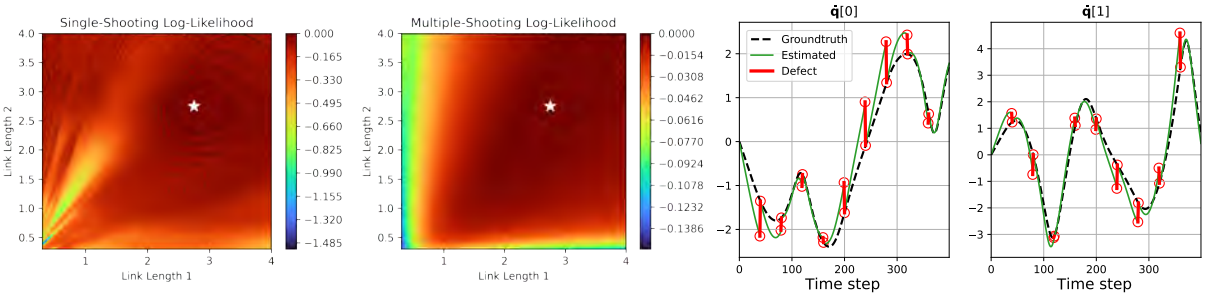
We present a solution based on the Stein Variational Gradient Descent (SVGD) algorithm [196] that approximates the posterior distribution $p(\theta|D_{\mathcal{X}}) = \frac{p(D_{\mathcal{X}}|\theta)p(\theta)}{\int p(D_{\mathcal{X}}|\theta)p(\theta)d\theta}$ by a set of particles $q(\theta|D_{\mathcal{X}}) = \frac{1}{N}\sum_{i=1}^{N}\delta(\theta_i - \theta)$ where $\delta(\cdot)$ is the Dirac delta function, and makes use of differentiable likelihood and prior functions to be efficient. SVGD avoids the computation of the intractable marginal likelihood in the denominator by only requiring the computation of $\nabla_\theta \log p(\theta|D_{\mathcal{X}}) = \frac{\nabla_\theta p(\theta|D_{\mathcal{X}})}{p(\theta|D_{\mathcal{X}})}$ which is independent of the normalization constant. The particles are adjusted according to the steepest descent direction to reduce the KL divergence in a reproducing kernel Hilbert space (RKHS) between the current set of particles representing $q(\theta|D_{\mathcal{X}})$ and the target $p(\theta|D_{\mathcal{X}})$.

As derived in [196], the set of particles $\{\theta_i\}_{i=1}^{N}$ is updated by the following function:

$$\begin{aligned} \theta_i &\leftarrow \theta_i + \epsilon\phi(\theta_i), \\ \phi(\cdot) &= \frac{1}{N}\sum_{j=1}^{N}\left[k(\theta_j,\theta)\nabla_{\theta_j}\log p(D_{\mathcal{X}}|\theta_j)p(\theta_j) + \nabla_{\theta_j}k(\theta_j,\theta)\right], \end{aligned} \tag{4.1}$$

where $k(\cdot,\cdot) : \mathbb{R}^M \times \mathbb{R}^M \to \mathbb{R}$ is a positive definite kernel and ϵ is the step size. In this work, we use the radial basis function kernel, which is a common choice for SVGD [196] due to its smoothness and infinite differentiability. To tune the kernel bandwidth, we adopt the median heuristic, which has been shown to provide robust performance on large learning problems [88].

SVGD requires that the likelihood function be differentiable in order to calculate Eq. 4.1, which in turn requires that f_{sim} and f_{obs} be differentiable. To enable this, we use a fully-differentiable simulator, observation function and likelihood function, and leverage automatic differentiation with code generation to generate CUDA kernel code [293]. Because of this CUDA code generation, we are able to calculate $\nabla_{\theta_j}\log p(D_{\mathcal{X}}|\theta)p(\theta)$ for each particle in parallel on the GPU.



(a) Single Shooting (b) Multiple Shooting (c) Shooting Windows and Defects

Figure 4.2: The two heatmaps plots on the left show the landscape of the log-likelihood function for an inference problem where the two link lengths of a double pendulum are estimated (ground-truth parameters indicated by a white star). In (a), the likelihood is evaluated over a 400-step trajectory; in (b), the trajectory is split into 10 shooting windows and the likelihood is computed via Eq. 4.6. In (c), the shooting intervals and defects are visualized for an exemplary parameter guess.

4.1.4.2 Likelihood Model for Trajectories

We define $p(\mathcal{X}^{\text{sim}}|\mathcal{X}^{\text{real}})$ as the probability of an individual trajectory $\mathcal{X}^{\text{sim}}$ simulated from parameters θ with respect to a matched ground-truth trajectory $\mathcal{X}^{\text{real}}$. Following the HMM assumption from Sec. 4.1.3, we treat $p(\mathcal{X}^{\text{sim}}|\mathcal{X}^{\text{real}})$ as the product of probabilities over observations, shown in Eq. 4.2. This assumption is justified because the next state $\mathbf{s}_{t+1}$ is fully determined by the position $\mathbf{q}_t$ and velocity $\dot{\mathbf{q}}_t$ of the current state $\mathbf{s}_t$ in articulated rigid body dynamics (see Sec. A.2.1). To directly compare observations we use a Gaussian likelihood:

$$\begin{aligned} p(\mathcal{X}^{\text{real}}|\mathcal{X}^{\text{sim}}) &= \prod_{t \in T} p(\mathbf{x}_t^{\text{real}}, \mathbf{x}_t^{\text{sim}}) \\ &= \prod_{t \in T} \mathcal{N}(\mathbf{x}_t^{\text{real}}|\mathbf{x}_t^{\text{sim}}, \sigma_{\text{obs}}^2). \end{aligned} \tag{4.2}$$

This likelihood model for observations is used to compute the likelihood for a trajectory

$$\begin{aligned} p_{ss}(\mathcal{X}^{\text{real}}|\theta) &= p(\mathcal{X}^{\text{real}}|f_{\text{obs}}(f_{\text{sim}}(\theta, \mathbf{s}_0^{\text{real}}))) \\ &= p(\mathcal{X}^{\text{real}}|\mathcal{X}^{\text{sim}}), \end{aligned} \tag{4.3}$$

where $\mathbf{s}_0^{\text{real}}$ is (an estimate of) the first state of $\mathcal{X}^{\text{real}}$. This formulation is known as a *single-shooting* estimation problem, where a trajectory is compared against another only by varying the initial conditions of the modeled system. To evaluate the likelihood for a collection of ground-truth trajectories, $D_{\mathcal{X}}^{\text{real}}$, we use an equally weighted Gaussian Mixture Model likelihood:

$$p_{\text{obs}}(D_{\mathcal{X}}^{\text{real}}|\theta) = \sum_{\mathcal{X}^{\text{real}} \in D_{\mathcal{X}}^{\text{real}}} p_{ss}(\mathcal{X}^{\text{real}}|\theta). \tag{4.4}$$

4.1.4.3 Multiple Shooting

Estimating parameters from long trajectories can prove difficult in the face of observation noise, and for systems in which small changes in initial conditions produce large changes in trajectories, potentially resulting in poor local optima [15]. We adopt the *multiple-shooting* method which significantly improves the convergence and stability of the estimation process. Multiple shooting has been applied in system identification problems [30] and biochemistry [253].

Multiple shooting divides up the trajectory into n_s shooting windows over which the likelihood is computed. To be able to simulate such shooting windows, we require their start states $\mathbf{s}^s$ to be available for f_{sim} to generate a shooting window trajectory. Since we only assume access to the first true state $\mathbf{s}_0^{\text{real}}$ from the real trajectory $\mathcal{X}^{\text{real}}$, we augment the parameter vector θ by the start states of the shooting windows, which we refer to as *shooting variables* $\mathbf{s}_t^s$ (for $t = h, 2h, \ldots, n_s \cdot h$). We define an augmented parameter vector as $\bar{\theta} = \begin{bmatrix} \theta & \mathbf{s}_h^s & \ldots & \mathbf{s}_{n_s \cdot h}^s \end{bmatrix}$. A shooting window of h time steps starting from time t is then simulated via $\mathcal{X}_t = f_{\text{obs}}(f_{\text{sim}}(\theta, \mathbf{s}_t^s)[0\!:\!h])$, where $[i\!:\!j]$ denotes a selection operation of the sub-array between the indices i and j. Analogous to Eq. 4.2, we evaluate the likelihood $p(\mathcal{X}^{\text{real}}[t\!:\!t\!+\!h] \mid \mathcal{X}_t^{\text{sim}})$ for a single shooting window as a product of state-wise likelihoods.

To impose continuity between the shooting windows, *defect constraints* are imposed as a Gaussian likelihood term between the last simulated state $\mathbf{s}_t$ from the previous shooting window at time t and the shooting variable $\mathbf{s}_t^s$ at time t:

$$p_{\text{def}}(\mathbf{s}_t^s, \mathbf{s}_t) = \mathcal{N}(\mathbf{s}_t^s | \mathbf{s}_t, \sigma_{\text{def}}^2) \qquad t \in [h, 2h, \dots] \tag{4.5}$$

where σ_{def}^2 is a small variance so that the MCMC samplers adhere to the constraint. Including the defect likelihood allows the extension of the likelihood defined in Eq. 4.3 to a multiple-shooting scenario:

$$\begin{aligned} p_{ms}(\mathcal{X}^{\text{real}} | \bar{\theta}) = \prod_{t \in H} p_{\text{def}}(\mathbf{s}_t^s, \mathbf{s}_t)\; p(\mathcal{X}^{\text{real}}[t\!:\!t+h] \mid \mathcal{X}_t^{\text{sim}}), \\ \mathbf{s}_0^s = \mathbf{s}_0^{\text{real}} \qquad H = [0, h, 2h, \dots] \end{aligned} \tag{4.6}$$

As for the single-shooting case, Eq. 4.4 with p_{ms} as the trajectory-wise likelihood function gives the likelihood p_{obs} for a set of trajectories.

In Fig. 4.2, we provide a parameter estimation example where the two link lengths of a double pendulum must be inferred. The single-shooting likelihood from Eq. 4.3 (shown in Fig. 4.2a) exhibits a rugged landscape where many estimation algorithms will require numerous samples to escape from poor local optima, or a much finer choice of parameter prior to limit the search space. The multiple-shooting likelihood (shown in Fig. 4.2) is significantly smoother and therefore easier to optimize.

4.1.4.4 Parameter Limits as a Uniform Prior

Simulators may not be able to handle simulating trajectories from any given parameter and it is often useful to enforce some limits on the parameters. To model this, we define a uniform prior distribution on the parameter settings $p_{\text{lim}}(\theta) = \prod_{i=1}^{M} U(\theta_i | \theta_{\text{min}_i}, \theta_{\text{max}_i})$, where $\theta_{\text{min}_i}, \theta_{\text{max}_i}$ denote the upper and lower limits of parameter dimension i.

4.1.4.5 Constrained Optimization for SVGD

Directly optimizing SVGD for the unnormalized posterior on long trajectories, with a uniform prior, can be difficult for gradient based optimizers. The uniform prior has discontinuities at the extremities, effectively imposing constraints, which produce non-differentiable regions in the parameter space domain. We propose an alternative solution to deal with this problem and treat SVGD as a constrained optimization on p_{obs} with p_{def} and p_{lim} as constraints.

A popular method of constrained optimization is the Modified Differential Multiplier Method (MDMM) [261] which augments the cost function to penalize constraint violations. In contrast to the basic penalty method, MDMM uses Lagrange multipliers in place of constant penalty coefficients that are updated automatically during the optimization:

$$\begin{aligned} \text{maximize} \quad & \log p_{\text{obs}}(D_{\mathcal{X}} = D_{\mathcal{X}}^{\text{real}} \mid \theta) \\ \text{s.t.} \quad & g(\bar{\theta}) = 0. \end{aligned} \tag{4.7}$$

MDMM formulates this setup into an unconstrained minimization problem by introducing Lagrange multipliers λ (initialized with zero):

$$\mathcal{L}_c(\bar{\theta}, \lambda) = -\log p_{\text{obs}}(D_{\mathcal{X}} = D_{\mathcal{X}}^{\text{real}} | \theta) + \lambda g(\bar{\theta}) + \frac{c}{2}[g(\bar{\theta})]^2, \tag{4.8}$$

where $c > 0$ is a constant damping factor that improves convergence in gradient descent algorithms. To accommodate these Lagrange multipliers per-particle we again extend the parameter set ($\bar{\theta}$) introduced

in Section 4.1.4.3 to store the multiple shooting variables and Lagrange multipliers, $\bar{\theta} = (\theta, \mathbf{s}^s, \lambda_{\text{def}}, \lambda_{\text{lim}})$. The following update equations for θ and λ are used to minimize Eq. 4.8:

$$\dot{\theta} = \frac{\partial \log p_{\text{obs}}(D_{\mathcal{X}}^{\text{real}}|\theta)}{\partial \theta} - \lambda \frac{\partial g(\bar{\theta})}{\partial \theta} - cg(\bar{\theta}) \frac{\partial g(\bar{\theta})}{\partial \theta}$$

$$\dot{\lambda} = g(\bar{\theta})$$

We include parameter limit priors from Section 4.1.4.4 as the following equality constraints (where $\text{clamp}(x, a, b)$ clips the value x to the interval $[a, b]$): $g_{\text{lim}}(\theta) = \text{clamp}(\theta, \theta_{\text{min}}, \theta_{\text{max}}) - \theta$. Other constraints are the defect constraints from Eq. 4.5 which are included as equality constraints as well:

$$g_{\text{def}}(\theta, \mathbf{s}_t^s) = \log p_{\text{def}}(\mathbf{s}_t^s, \mathbf{s}_t) = \|\mathbf{s}_t^s - \mathbf{s}_t\|^2 / \sigma_{\text{def}}^2.$$

The overall procedure of our Constrained SVGD (CSVGD) algorithm is given in Algorithm 1.

Algorithm 5 Constrained SVGD

Require: differentiable simulator $f_{\text{sim}} : (\theta, \mathbf{s}_0) \mapsto [\mathbf{s}]$, observation function $f_{\text{obs}} : \mathbf{s} \mapsto \mathbf{x}$, start states $\mathbf{s}_0^i$ for each ground-truth trajectory $\mathcal{X}_i^{\text{real}} \in D_{\mathcal{X}}^{\text{real}}$, learning rate scheduler (e.g. Adam), kernel choice (e.g. RBF)

for $i = 1 \dots \text{max\_iterations}$ **do**

Roll out simulated observations $D_{\mathcal{X}}^{\text{sim}} = [f_{\text{obs}}(f_{\text{sim}}(\theta, \mathbf{s}_0^{\text{real}})) \; \forall \mathcal{X}^{\text{real}} \in D_{\mathcal{X}}^{\text{real}}]$

Compute $\phi(\theta)$ via Eq. 4.1 and $\log p_{\text{obs}}(D_{\mathcal{X}} = D_{\mathcal{X}}^{\text{real}} \mid \theta)$

Update θ via $\dot{\theta} = \phi(\theta) - \lambda_{\text{lim}} \frac{\partial g_{\text{lim}}}{\partial \theta} - cg_{\text{lim}} \frac{\partial g_{\text{lim}}}{\partial \theta} - \lambda_{\text{def}} \frac{\partial g_{\text{def}}}{\partial \theta} - cg_{\text{def}} \frac{\partial g_{\text{def}}}{\partial \theta}$

Update $\lambda_{\text{lim}}, \lambda_{\text{def}}, \dot{\mathbf{s}^s}_t$ via $\dot{\lambda}_{\text{lim}} = g_{\text{lim}}(\theta), \dot{\lambda}_{\text{def}} = g_{\text{def}}(\theta), \dot{\mathbf{s}^s}_t = g_{\text{def}}(\theta, \mathbf{s}_t^s)$ for $t \in [h, 2h, \dots]$

end for

return θ

4.1.4.6 Performance Metrics for Particle Distributions

In most cases we do not know the underlying ground-truth parameter distribution $p(\theta^{\text{real}})$, and only have access to a finite set of ground-truth trajectories. Therefore, we measure the discrepancy between trajectories rolled out from the estimated parameter distribution, $D_{\mathcal{X}}^{\text{sim}}$, and the reference trajectories, $D_{\mathcal{X}}^{\text{real}}$.

One measure, the KL divergence, is the expected value of the log likelihood ratio between two distributions. Although the KL divergence cannot be calculated from samples of continuous distributions, methods have been developed to estimate it from particle distributions using the k-nearest neighbors distance [328]. The KL divergence is non-symmetric and this estimate can be poor in two situations. The first is estimating $d_{\mathrm{KL}}(D_{\mathcal{X}}^{\mathrm{sim}} \parallel D_{\mathcal{X}}^{\mathrm{real}})$ when the particles are all very close to one trajectory, causing a low estimated divergence, yet the posterior is of poor quality. The opposite can happen when the particles in the posterior are overly spread out when estimating $d_{\mathrm{KL}}(D_{\mathcal{X}}^{\mathrm{real}} \parallel D_{\mathcal{X}}^{\mathrm{sim}})$.

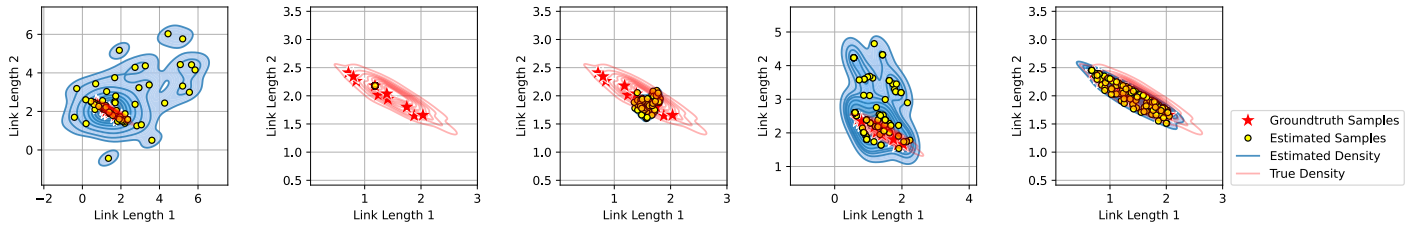
We additionally measure the maximum mean discrepancy (MMD) [96], which is a metric used to determine if two sets of samples are drawn from the same distribution by calculating the square of the distance between the embedding of the two distributions in a RKHS.

4.1.5 Experiments

We compare our method against commonly used parameter estimation baselines. As an algorithm comparable to our particle-based approach, we use the Cross Entropy Method (CEM) [283]. In addition, we evaluate the Markov chain Monte-Carlo techniques Emcee [83], Stochastic Gradient Langevin Dynamics (SGLD) [332], and the No-U-Turn-Sampler (NUTS) [133], which is an adaptive variant of the gradient-based Hamiltonian MC algorithm. For Emcee, we use a parallel sampler that implements the "stretch move" ensemble method [95]. For BayesSim, we report results from the best performing instantiation using a mixture density random Fourier features model (MDRFF) with Matérn kernel in Tab. 4.1. We present further details and extended results from our experiments in the appendix.

4.1.5.1 Parameter Estimation Accuracy

We create a synthetic dataset of 10 trajectories from a double pendulum which we simulate by varying its two link lengths. These two uniquely identifiable [74] parameters are drawn from a Gaussian distribution



(a) Emcee (b) CEM (c) SGLD (d) NUTS (e) SVGD

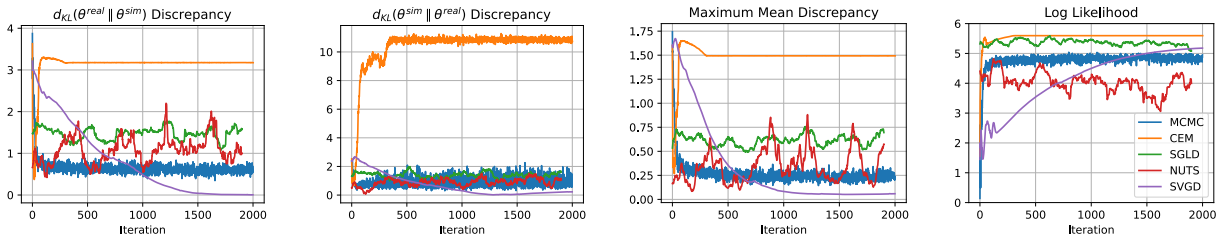
Figure 4.3: Estimated posterior distributions from synthetic data generated from a known multivariate Gaussian distribution (Section 4.1.5.1). The 100 last samples were drawn from the Markov chains sampled by Emcee, SGLD, and NUTS, while CEM and SVGD used 100 particles.

Experiment	Metric	Emcee	CEM	SGLD	NUTS	SVGD	BayesSim	CSVGD (Ours)
Double Pendulum	$d_{\mathrm{KL}}(D_{\mathcal{X}}^{\mathrm{real}} \parallel D_{\mathcal{X}}^{\mathrm{sim}})$	8542.2466	8911.1798	8788.0962	9196.7461	8803.5683	8818.1830	**5204.5336**
	$d_{\mathrm{KL}}(D_{\mathcal{X}}^{\mathrm{sim}} \parallel D_{\mathcal{X}}^{\mathrm{real}})$	4060.6312	8549.5927	7876.0310	6432.2131	10283.6659	3794.9873	**2773.1751**
	MMD	1.1365	0.9687	2.1220	0.5371	0.7177	0.6110	**0.0366**
Panda Arm	$\log p_{\mathrm{obs}}(D_{\mathcal{X}}^{\mathrm{real}} \parallel D_{\mathcal{X}}^{\mathrm{sim}})$	-16.1185	-17.3331	-17.3869	-17.9809	-17.7611	-17.6395	**-15.1671**

Table 4.1: Consistency metrics of the posterior distributions approximated by the different estimation algorithms. Each metric is calculated across simulated and real trajectories. Lower is better on all metrics except $\log p_{\mathrm{obs}}(D_{\mathcal{X}}^{\mathrm{real}} \parallel D_{\mathcal{X}}^{\mathrm{sim}})$.

with a mean of $(1.5\,\mathrm{m}, 2\,\mathrm{m})$ and a full covariance matrix (density visualized by the red contour lines in Fig. 4.3). We show the evolution of the consistency metrics in Fig. 4.4. Trajectories generated by evaluating the posteriors of the compared methods are compared against 50 test trajectories rolled out from the ground-truth parameter distribution.

We find that SVGD produces a density very close to the density that matches the principal axes of the ground-truth posterior, and outperforms the other methods in all metrics except log likelihood. CEM collapses to a single high-probability estimate but does not accurately represent the full posterior, as can be seen in Fig. 4.4d being maximal quickly but poor performance on the other metrics which measure the spread of the posterior. Emcee and NUTS represent the spread of the posterior but do not sharply capture the high-likelihood areas, shown by their good performance in Fig. 4.4b, Fig. 4.4b and Fig. 4.4c. SGLD captures a small amount of the posterior around the high likelihood points but does not fully approximate the posterior.



(a) (b) (c) (d)

Figure 4.4: Accuracy metrics for the estimated parameter posteriors shown in Fig. 4.3. The estimations were done on the synthetic dataset of a multivariate Gaussian over parameters (Section 4.1.5.1). Fig. 4.4d shows the single-shooting likelihood using the equation described in Eq. 4.3.

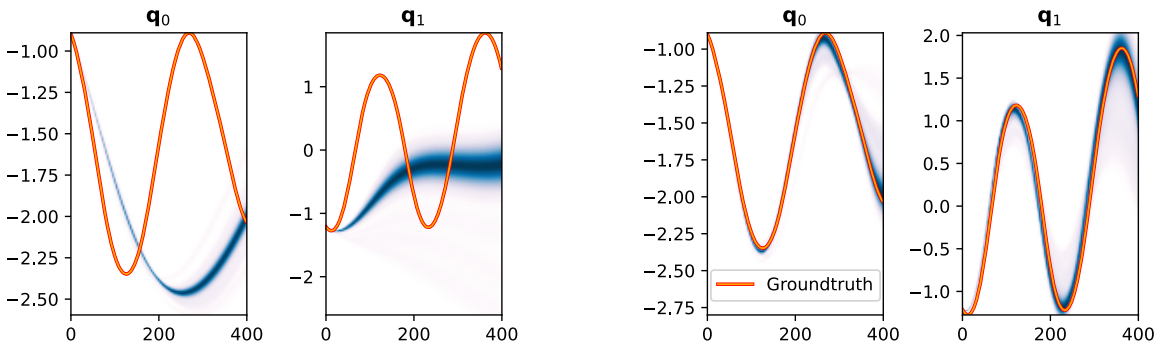
4.1.5.2 Identify Real-world Double Pendulum

We leverage the dataset from [9] containing trajectories from a physical double pendulum. While in our previous synthetic data experiment the parameter space was reduced to only the two link lengths, we now define 11 parameters to estimate. The parameters for each link are the mass, inertia I_{xx}, the center of mass in the x and y direction, and the joint friction. We also estimate the length of the second link. Note that parameters, such as the length of the first link, are not explicitly included since they are captured by the remaining parameters, which we validated through sensitivity analysis. Like before, the state space is completely measurable, i.e. $\mathbf{s} \approx \mathbf{x}$, except for observation noise.

In this experiment we find that CSVGD outperforms all other methods in KL divergence (both ways) as well as MMD, shown in Tab. 4.1. We believe this is because of the complex relationship between the parameters of each link and the resulting observations. This introduces many local minima which are hard to escape from (see Fig. 4.5). The multiple-shooting likelihood improves the convergence significantly by simplifying the optimization landscape.

4.1.5.3 Identify Inertia of an Articulated Rigid Object

In our final experiment, we investigate a more complicated physical system which is underactuated. Through an uncontrollable universal joint, we attach an acrylic box to the end-effector of a physical 7-DOF Franka Emika Panda robot arm. We fix two 500 g weights at the bottom inside the box, and prescribe a trajectory



(a) SVGD (b) CSVGD

Figure 4.5: Trajectory density plots (only joint positions $\mathbf{q}$ are shown) obtained by simulating the particle distribution found by SVGD with the single-shooting likelihood (Fig. 4.5a) and the multiple-shooting likelihood (Fig. 4.5b) on the real-world double pendulum (Section 4.1.5.2).

which the robot executes through a PD controller. By tracking the motion of the box via a VICON motion capture system, we aim to identify the 2D locations of the two weights (see Fig. 4.1). The system only has access to the proprioceptive measurements from the robot arm (seven joint positions and velocities), as well as the 3D pose of the box at each time step. In the first phase, we identify the inertia properties of the empty box, as well as the joint friction parameters of the universal joint from real-robot trajectories of shaking an empty box. Given our best estimate, in the actual parameter estimation setup we infer the two planar positions of the weights.

The particles from SVGD and CSVGD quickly converge in a way that the two weights are aligned opposed to each other. If the weights were not at locations symmetrical about the center, the box would tilt and yield a large discrepancy to the real observations. MCMC, on the other hand, even after more than ten times the number of iterations, only rarely approaches configurations in which the box remains balanced. In Fig. 4.1 (right) we visualize the posterior over weight locations found by CSVGD (blue shade). The found symmetries are clearly visible when we draw lines (orange) between the inferred positions of both weights (yellow, green), while the true weight locations (red) are contained in the approximated distribution. As can be seen in Tab. 4.1, the log likelihood is maximized by CSVGD. The results indicate

the ability of our method to accurately model difficult posteriors over complex trajectories because of the symmetries underlying the simulation parameters.

4.1.6 Conclusion

We have presented Constrained Stein Variational Gradient Descent (CSVGD), a new method for estimating the distribution over simulation parameters that leverages Bayesian inference and parallel, differentiable simulators. By segmenting the trajectory into multiple shooting windows via hard defect constraints, and effectively using the likelihood gradient, CSVGD produces more accurate posteriors and exhibits improved convergence over previous estimation algorithms.

In future work, we plan to leverage the probabilistic predictions from our simulator for uncertainty-aware control applications. Similar to Lambert et al. [176], the particle-based uncertainty information can be leveraged by a model-predictive controller that takes into account the multi-modality of future outcomes.

4.2 NeuralSim: Augmenting Differentiable Simulators with Neural Networks

Differentiable simulators provide an avenue for closing the sim-to-real gap by enabling the use of efficient, gradient-based optimization algorithms to find the simulation parameters that best fit the observed sensor readings. Nonetheless, these analytical models can only predict the dynamical behavior of systems for which they have been designed. In this work, we study the augmentation of a novel differentiable rigid-body physics engine via neural networks that is able to learn nonlinear relationships between dynamic quantities and can thus model effects not accounted for in traditional simulators. Such augmentations require less data to train and generalize better compared to entirely data-driven models. Through extensive experiments, we demonstrate the ability of our hybrid simulator to learn complex dynamics involving frictional contacts from real data, as well as match known models of viscous friction, and present an approach for automatically discovering useful augmentations. We show that, besides benefiting dynamics modeling, inserting neural networks can accelerate model-based control architectures. We observe a ten-fold speed-up when replacing the QP solver inside a model-predictive gait controller for quadruped robots with a neural network, allowing us to significantly improve control delays as we demonstrate in real-hardware experiments.

4.2.1 Introduction

Physics engines enable the accurate prediction of dynamical behavior of systems for which an analytical model has been implemented. Given such models, Bayesian estimation approaches [274] can be used to find the simulation parameters that best fit the real-world observations of the given system. While the estimation of simulation parameters for traditional simulators via black-box optimization or likelihood-free inference algorithms often requires a large number of training data and model evaluations, differentiable

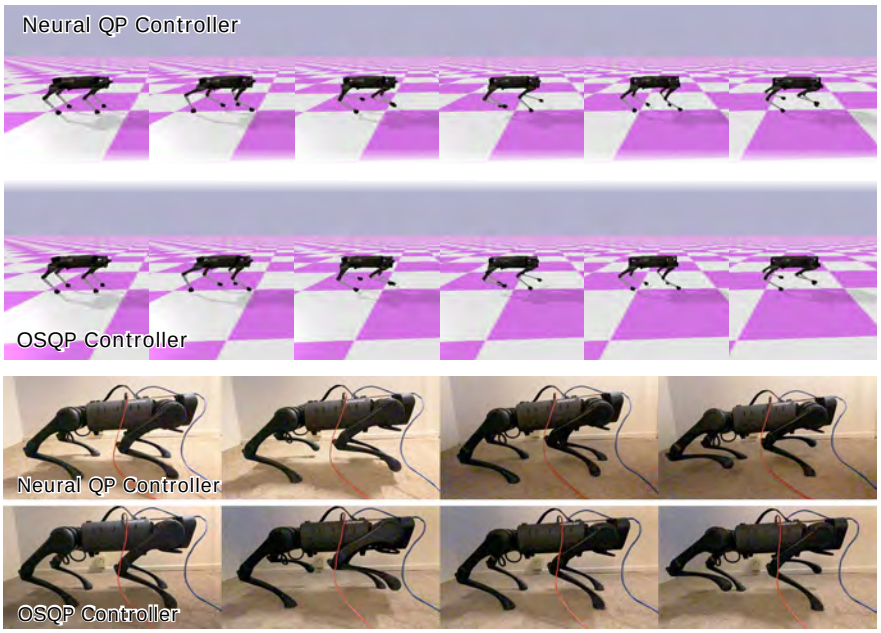


Figure 4.6: *Top:* Snapshots from a few seconds of the locomotion trajectories generated by the QP-based model-predictive controller (first row) and the neural network controller imitating the QP solver (second row) in our differentiable simulator running the Laikago quadruped. *Bottom:* model-predictive control using the neural network (third row) and OSQP (fourth row) running on a real Unitree A1 robot.

	Analytical	**Data-driven**	**End-to-end ∇**	**Hybrid**
Physics engine [54, 319, 179]	✓			
Residual physics [144, 3, 353, 94]	✓	✓		✓
Learned physics [291, 18, 189, 154]		✓	✓	
Differentiable simulators [92, 137, 40, 14, 91, 266, 239]	✓		✓	
Our approach	✓	✓	✓	✓

Table 4.2: Comparison of dynamics modeling approaches (selected works) along the axes of analytical and data-driven modeling, end-to-end differentiability, and hybrid approaches.

simulators allow the use of gradient-based optimizers that can significantly improve the speed of parameter inference approaches [239].

Nonetheless, a sim-to-real gap remains for most systems we use in the real world. For example, in typical simulations used in robotics, the robot is assumed to be an articulated mechanism with perfectly rigid links, even though they actually bend under heavy loads. Motions are often optimized without accounting for air resistance. In this work, we focus on overcoming such errors due to unmodeled effects by implementing a differentiable rigid-body simulator that is enriched by fine-grained data-driven models.

Instead of learning the error correction between the simulated and the measured states, as is done in residual physics models, our augmentations are embedded inside the physics engine. They depend on physically meaningful quantities and actively influence the dynamics by modulating forces and other state variables in the simulation. Thanks to the end-to-end differentiability of our simulator, we can efficiently train such neural networks via gradient-based optimization.

Our experiments demonstrate that the hybrid simulation approach, where data-driven models augment an analytical physics engine at a fine-grained level, outperforms deep learning baselines in training efficiency and generalizability. The hybrid simulator is able to learn a model for the drag forces for a swimmer mechanism in a viscous medium with orders of magnitude less data compared to a sequence learning model. When unrolled beyond the time steps of the training regime, the augmented simulator significantly outperforms the baselines in prediction accuracy. Such capability is further beneficial to closing the sim-to-real gap. We present results for planar pushing where the contact friction model is enriched by neural networks to closely match the observed motion of an object being pushed in the real world. Leveraging techniques from sparse regression, we can identify which inputs to the data-driven models are the most relevant, giving the opportunity to discover places where to insert such neural augmentations.

Besides reducing the modeling error or learning entirely unmodeled effects, introducing neural networks to the computation pipeline can have computational benefits. In real-robot experiments on a Unitree A1 quadruped we show that when the quadratic programming (QP) solver of a model-predictive gait controller is replaced by a neural network, the inference time can be reduced by an order of magnitude.

We release our novel, open-source physics engine, Tiny Differentiable Simulator (TDS)[*], which implements articulated rigid-body dynamics and contact models, and supports various methods of automatic differentiation to compute gradients w.r.t. any quantity inside the engine.

[*]Open-source release of TDS: https://github.com/google-research/tiny-differentiable-simulator

4.2.2 Related Work

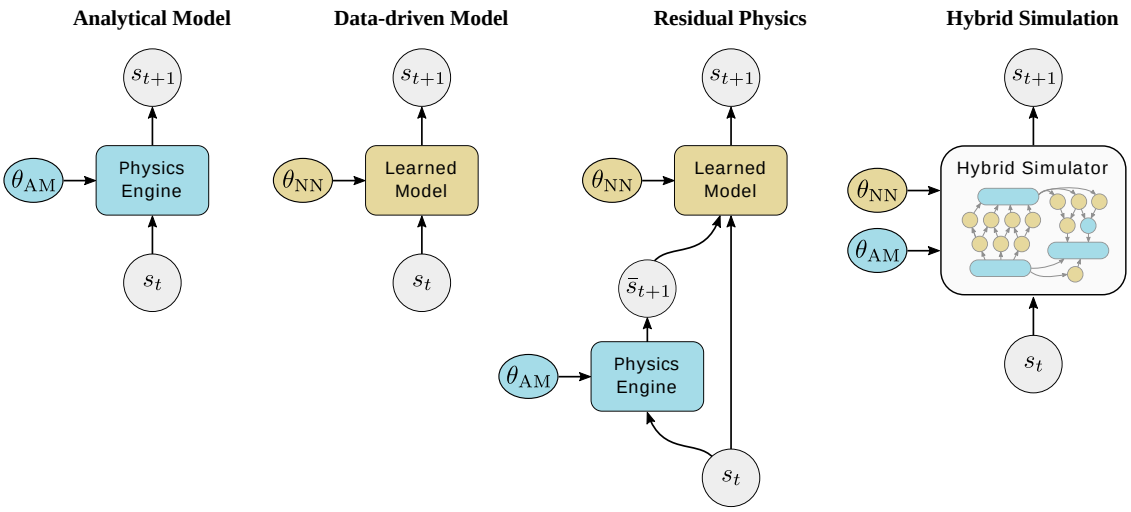


Figure 4.7: Comparison of various model architectures (cf. Anurag et al. [3]).

Various methods have been proposed to learn system dynamics from time series data of real systems (see Tab. 4.2 and Fig. 4.7). Early works include Locally Weighted Regression [295] and Forward Models [229]. Modern "intuitive physics" models often use deep graph neural networks to discover constraints between particles or bodies ([18, 348, 114, 268, 234, 46, 189, 291]). Physics-based machine learning approaches introduce a physics-informed inductive bias to the problem of learning models for dynamical systems from data [45, 202, 269, 205, 97, 311]. We propose a general-purpose hybrid simulation approach that combines analytical models of dynamical systems with data-driven residual models that learn parts of the dynamics unaccounted for by the analytical simulation models.

Originating from traditional physics engines (examples include Coumans and Bai [54], Todorov, Erez, and Tassa [319], and Lee et al. [179]), differentiable simulators have been introduced that leverage automatic, symbolic or implicit differentiation to calculate parameter gradients through the analytical Newton-Euler equations and contact models for rigid-body dynamics [92, 40, 14, 169, 124, 123, 91, 181, 206]. Simulations for other physical processes, such as light propagation [243, 119] and continuum mechanics [139, 190, 137, 266, 239] have been made differentiable as well.

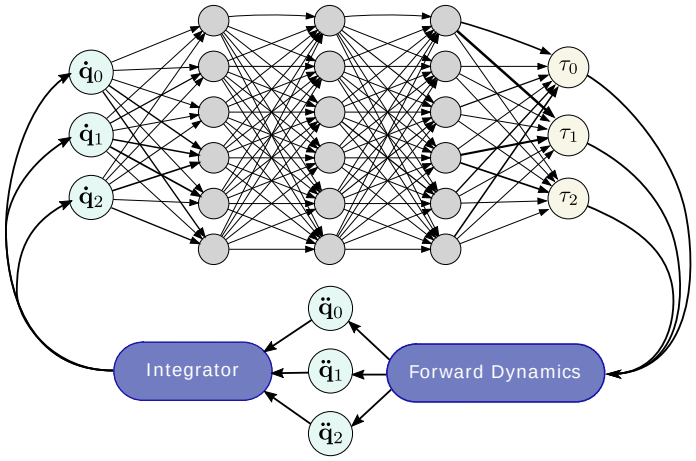


Figure 4.8: Exemplar architecture of our hybrid simulator where a neural network learns passive forces τ given joint velocities $\dot{\mathbf{q}}$ (MLP biases and joint positions $\mathbf{q}$ are not shown).

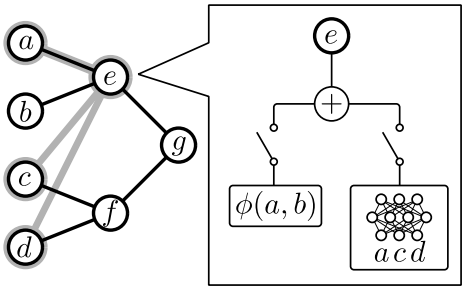


Figure 4.9: Augmentation of a differentiable simulator with our proposed neural scalar type where variable e becomes a combination of an analytical model $\phi(\cdot,\cdot)$ with inputs a and b, and a neural network whose inputs are a, c, and d.

Residual physics models [353, 3, 144, 94] augment physics engines with learned models to reduce the sim-to-real gap. Most of them introduce residual learning applied from the outside to the predicted state of the physics engine, while we propose a more fine-grained approach, similar to [144], where data-driven models are introduced only at some parts in the simulator. While in Hwangbo et al. [144] the network for actuator dynamics is trained through supervised learning, our end-to-end differentiable model allows backpropagation of gradients from high-level states to any part of the computation graph, including neural network weights, so that these parameters can be optimized efficiently, for example from end-effector trajectories.

4.2.3 Approach

Our simulator implements the Articulated Body Algorithm (ABA) [76] to compute the forward dynamics (FD) for articulated rigid-body mechanisms. Given joint positions $\mathbf{q}$, velocities $\dot{\mathbf{q}}$, torques τ in generalized coordinates, and external forces $\mathbf{f}_{\text{ext}}$, ABA calculates the joint accelerations $\ddot{\mathbf{q}}$. We use semi-implicit Euler integration to advance the system dynamics in time. We support two types of contact models: an impulse-level solver that uses a form of the Projected Gauss Seidel algorithm [309, 134] to solve a nonlinear complementarity problem (NCP) to compute contact impulses for multiple simultaneous points of contact, and a penalty-based contact model that computes joint forces individually per contact point via the Hunt-Crossley nonlinear spring-damper system [142]. In addition to the contact normal force, the NCP-based model resolves lateral friction following Coulomb's law, whereas the spring-based model implements a nonlinear friction force model following Brown and McPhee [34].

Each link i in a rigid-body system has a mass value m_i, an inertia tensor $\mathbf{I}_i \in \mathbb{R}^{6\times6}$, and a center of mass $\mathbf{h}_i \in \mathbb{R}^{6\times6}$. Each corresponding n_d degree of freedom joint i between link i and its parent $\lambda(i)$ has a spring stiffness coefficient k_i, a damping coefficient d_i, a transformation from its axis to its parent ${}^{\lambda(i)}\mathbf{X}_i \in \mathbb{SE}(3)$, and a motion subspace matrix $\mathbf{S}_i \in \mathbb{R}^{6\times n_d}$.

For problems involving n_c possible contacts, each possible pair j of contacts has a friction coefficient $\mu_j \in \mathbb{R}$ describing the shape of the Coulomb friction cone. Additionally, to maintain contact constraints during dynamics integration, we use Baumgarte stabilization [19] with parameters $\alpha_j, \beta_j \in \mathbb{R}$. To soften the shape of the contact NCP, we use Constraint Force Mixing (CFM), a regularization technique with parameters $\mathbf{R}_j \in \mathbb{R}^{n_c}$.

$$\theta_{AM} = \{m_i, \mathbf{I}_i, \mathbf{h}_i, {}^{\lambda(i)}\mathbf{X}_i, \mathbf{S}_i\}_i \cup \{\mu_j, \alpha_j, \beta_j, \mathbf{R}_j\}_j$$

are the *analytical model parameters* of the system, which have an understandable and physically relevant effect on the computed dynamics. Any of the parameters in θ_{AM} may be measurable with certainty a priori, or can be optimized as part of a system identification problem according to some data-driven loss function.

With these parameters, we view the dynamics of the system as

$$\tau = \mathbf{H}_{\theta_{AM}}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}_{\theta_{AM}}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{g}_{\theta_{AM}}(\mathbf{q}) \tag{4.9}$$

where $\mathbf{H}_{\theta_{AM}}$ computes the generalized inertia matrix of the system, $\mathbf{C}_{\theta_{AM}}$ computes nonlinear Coriolis and centrifugal effects, and $\mathbf{g}_{\theta_{AM}}$ describes passive forces and forces due to gravity.

4.2.3.1 Hybrid Simulation

Even if the best-fitting analytical model parameters have been determined, rigid-body dynamics alone often does not exactly predict the motion of mechanisms in the real world. To address this, we propose a technique for hybrid simulation that leverages differentiable physics models and neural networks to allow the efficient reduction of the simulation-to-reality gap. By enabling any part of the simulation to be replaced or augmented by neural networks with parameters θ_{NN}, we can learn unmodeled effects from data which the analytical models do not account for.

Our physics engine allows any variable participating in the simulation to be augmented by neural networks that accept input connections from any other variable. In the simulation code, such *neural scalars* (Fig. 4.9) are assigned a unique name, so that in a separate experiment code a "neural blueprint" is defined that declares the neural network architectures and sets the network weights.

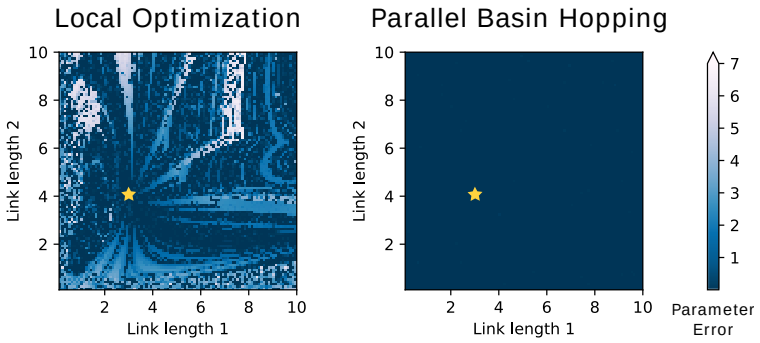


Figure 4.10: Comparison of system identification results from local optimization (left) and the parallel basin hopping strategy (right), where the grid cells indicate the initial parameters from which the optimization was started. The link lengths of a double pendulum are the two simulation parameters to be estimated from joint position trajectories. The true parameters are 3 and 4 (indicated by a star), and the colors indicate the $\ell 2$ distance between the ground truth and the parameters obtained after optimization (darker shade indicates lower error).

4.2.3.2 Overcoming Local Minima

For problems involving nonlinear or discontinuous systems, such as rigid-body mechanisms undergoing contact, our strategy yields highly nonlinear loss landscape which is fraught with local minima. As such, we employ global optimization strategies, such as the parallel basin hopping strategy [216] and population-based techniques from the Pagmo library [27]. These global approaches run a gradient-based optimizer, such as L-BFGS in our case, locally in each individual thread. After each evolution of a population of local optimizers, the best solutions are combined and mutated so that in the next evolution each optimizer starts from a newly generated parameter vector. As can be seen in Fig. 4.10, a system identification problem as basic as estimating the link lengths of a double pendulum from a trajectory of joint positions, results in a complex loss landscape where a purely local optimizer often only finds suboptimal parameters (left). Parallel Basin Hopping (right), on the other hand, restarts multiple local optimizers with an initial value around the currently best found solution, thereby overcoming poor local minima in the loss landscape.

4.2.3.3 Implementation Details

We implement our physics engine *Tiny Differentiable Simulator* (TDS) in C++ while keeping the computations agnostic to the implementation of linear algebra structures and operations. Via template metaprogramming, various math libraries, such as Eigen [105] and Enoki [148] are supported without requiring changes to the experiment code. Mechanisms can be imported into TDS from Universal Robot Description Format (URDF) files.

To compute gradients, we currently support the following third-party automatic differentiation (AD) frameworks: the "Jet" implementation from the Ceres solver [1] that evaluates the partial derivatives for multiple parameters in forward mode, as well as the libraries Stan Math [39], CppAD [20], and CppADCodeGen [178] that support both forward- and reverse-mode AD. We supply operators for taking gradients through conditionals such that these constructs can be traced by the tape recording mechanism in CppAD. We further exploit code generation not only to compile the gradient pass but to additionally speed up the regular simulation and loss evaluation by compiling models to non-allocating C code.

In our benchmark shown in Fig. 4.11, we compare the gradient computation runtimes (excluding compilation time for CppADCodeGen) on evaluating a 175-parameter neural network, simulating 5000 steps of a 5-link pendulum falling on the ground via the NCP contact model, and simulating 500 steps of a double pendulum without contact. Similar to Giftthaler et al. [92], we observe orders of magnitude in speed-up by leveraging CppADCodeGen to precompile the gradient pass of the cost function that rolls out the system with the current parameters and computes the loss.

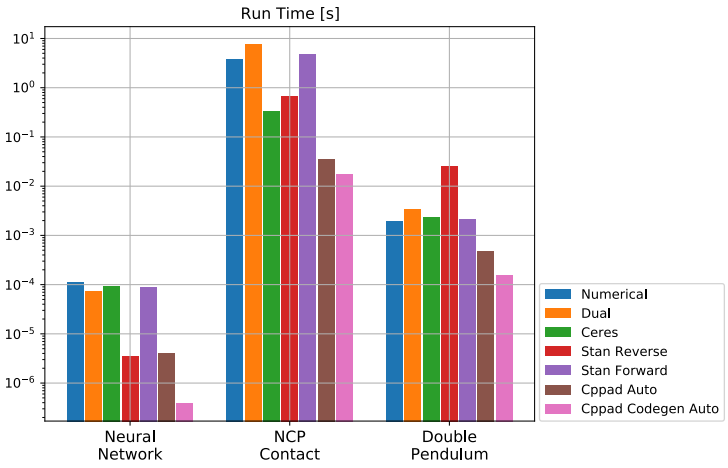


Figure 4.11: Runtime comparison of finite differences (blue) against the various automatic differentiation frameworks supported by our physics engine. Note the log scale for the time (in seconds).

4.2.4 Experimental Results

4.2.4.1 Friction Dynamics in Planar Pushing

Combining a differentiable physics engine with learned, data-driven models can help reduce the simulation-to-reality gap. Based on the Push Dataset released by the MCube lab [350], we demonstrate how an analytical contact model can be augmented by neural networks to predict complex frictional contact dynamics.

The dataset contains a variety of planar shapes that are pushed by a robot arm equipped with a cylindrical tip on different types of surface materials. Numerous trajectories of planar pushes with different velocities and accelerations have been recorded, from which we select trajectories with constant velocity. We model the contact between the tip and the object via the NCP-based contact solver (Sec. 4.2.3). For the ground, we use a nonlinear spring-damper contact model that computes the contact forces at predefined contact points for each of the shapes. We pre-processed each shape to add contact points along a uniform grid with a resolution of 2 cm (Fig. 4.12 right). Since the NCP-based contact model involves solving a nonlinear complementarity problem, we found the force-level contact model easier to augment by neural networks in this scenario since the normal and friction forces are computed independently for each contact point.

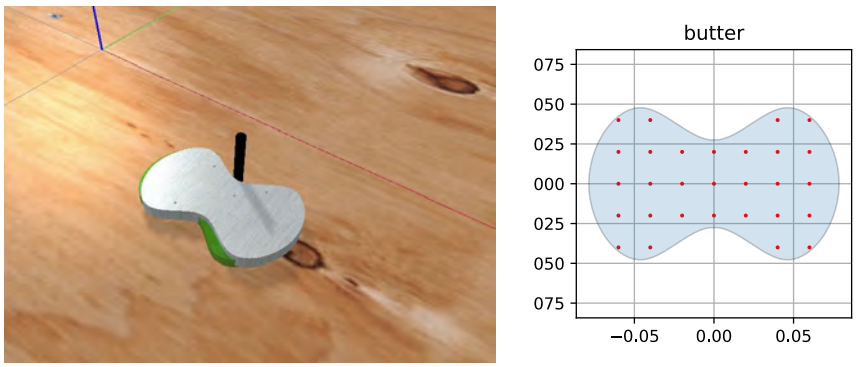


Figure 4.12: *Left:* example setup for the push experiment (Section 4.2.4.1) where the rounded object is pushed by the black cylindrical tip at a constant velocity on a plywood surface. Without tuning the analytical physics engine, a deviation between the pose of the object in the real world (green) and in the simulator (metallic) becomes apparent. *Right:* predefined contact points (red dots) for one of the shapes in the push dataset.

As shown in Fig. 4.12 (left), even when these two contact models and their analytical parameters (e.g. friction coefficient, Baumgarte stabilization gains, etc.) have been tuned over a variety of demonstrated trajectories, a significant error between the ground truth and simulated pose of the object remains. By augmenting the lateral 2D friction force of the object being pushed, we achieve a significantly closer match to the real-world ground truth (cf. Fig. 4.13). The neural network inputs are given by the object's current pose, its velocity, as well as the normal and the penetration depth at the point of contact between the tip and the object. Such contact-related variables are influenced by the friction force correction that the neural network provides. Unique to our differentiable simulation approach is the ability to compute gradients for the neural network weights from the trajectories of the object poses compared against the ground truth, since the influence of the friction force on the object pose becomes apparent only after the forward dynamics equations have been applied and integrated.

4.2.4.2 Passive Frictional Forces for Articulated Systems

While in the previous experiment an existing contact model has been enriched by neural networks to compute more realistic friction forces, in this experiment we investigate how such augmentation can learn entirely new phenomena which were not accounted for in the analytical model. Given simulated trajectories

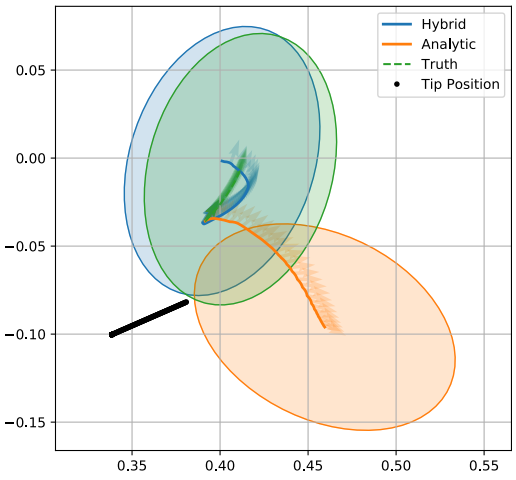


Figure 4.13: Trajectories of pushing an ellipsoid object from our hybrid simulator (blue) and the non-augmented rigid-body simulation (orange) compared against the real-world ground truth from the MCube push dataset [350]. The shaded ellipsoids indicate the final object poses.

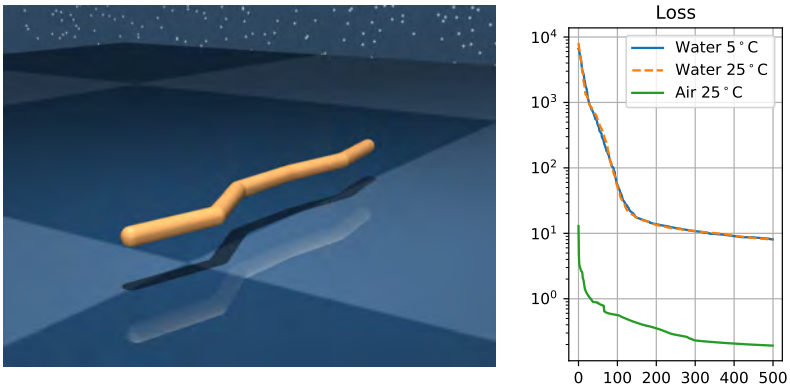


Figure 4.14: *Left:* Model of a swimmer in MuJoCo with five revolute joints connecting its links simulated as capsules interacting with ambient fluid. *Right:* Evolution of the cost function during the training of the neural-augmented simulator for the swimmer in different media.

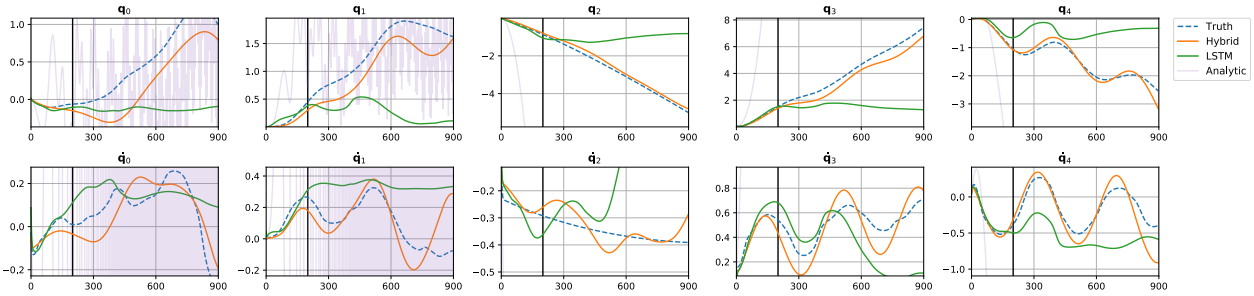


Figure 4.15: Trajectory of positions $\mathbf{q}$ and velocities $\dot{\mathbf{q}}$ of a swimmer in water at 25 °C temperature actuated by random control inputs. In a low-data regime, our approach (orange) generates accurate predictions, even past the training cutoff (vertical black bar), while an entirely data-driven model (green) regresses to the mean. The behavior of the unaugmented analytical model, which corresponds to a swimmer in vacuum, is shown for comparison.

Medium	Dynamic Viscosity μ [Pa·s]	Density ρ [kg / m^3]
Vacuum	0	0
Water (5 °C)	0.001 518	1000
Water (25 °C)	0.000 89	997
Air (25 °C)	0.000 018 49	1.184

Table 4.3: Physical properties of media used in the swimmer experiment.

of a multi-link robot swimming through an unidentified viscous medium (Fig. 4.14), the augmentation is supposed to learn a model for the passive forces that describe the effects of viscous friction and damping encountered in the fluid. Since the passive forces exerted by a viscous medium are dependent on the angle of attack of each link, as well as the velocity, we add a neural augmentation to τ, $\psi_\theta(\mathbf{q}, \dot{\mathbf{q}})$, analogous to Fig. 4.8.

Using our hybrid physics engine, and given a set of generalized control forces u, we integrate Eq. 4.9 to form a trajectory $\hat{T} = \{\hat{\mathbf{q}}_i, \hat{\dot{\mathbf{q}}}_i\}_i$, and compute an MSE loss $\mathcal{L}(T, \hat{T})$ against T, a ground truth trajectory of generalized states and velocities recorded using the MuJoCo [319] simulator. We are able to compute exact gradients $\nabla_\theta \mathcal{L}$ end-to-end through the entire trajectory, given the differentiable nature of our simulator, integrator, and function approximator.

We train the hybrid simulator under different ground-truth inputs. We consider water at 5 °C and 25 °C temperature, plus air at 25 °C, as media, which have different properties for dynamic viscosity and density (see Tab. 4.3).

Leveraging the mechanical structure of problems aids in long-term physically relevant predictive power, especially in the extremely sparse data regimes common in learning for robotics. As shown in Fig. 4.15, our approach, trained end-to-end on the initial 200 timesteps of 10 trajectories of a 2-link swimmer moving in water at 25 °C, exhibits more accurate long-term rollout prediction over 900 timesteps than a deep long short-term memory (LSTM) network. In this experiment, the augmentation networks for TDS have 1637 trainable parameters, while the LSTM has 1900.

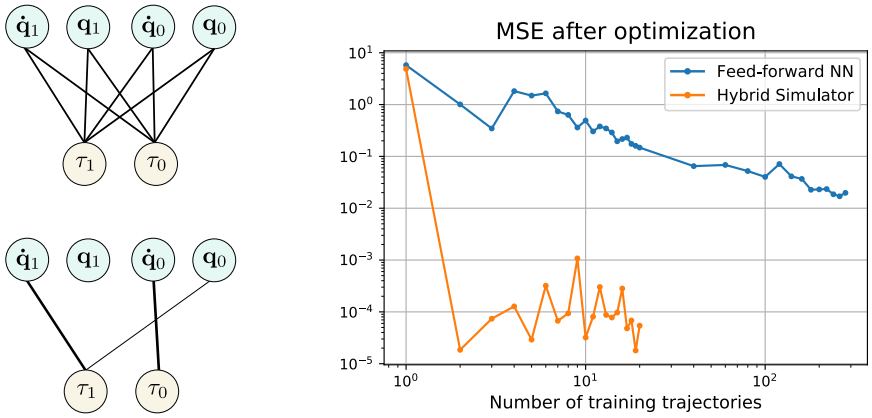


Figure 4.16: *Left:* Neural augmentation for a double pendulum that learns joint damping. Given the fully connected neural network (top), after 15 optimization steps using the sparse group lasso cost function (Eq. 4.10) (bottom), the input layer (light green) is sparsified to only the relevant quantities that influence the dynamics of the observed system. *Right:* Final mean squared error (MSE) on test data after training a feed-forward neural network and our hybrid physics engine.

4.2.4.3 Discovering Neural Augmentations

In our previous experiments we carefully selected which simulation quantities to augment by neural networks. While it is expected that air resistance and friction result in passive forces that need to be applied to the simulated mechanism, it is often difficult to foresee which dynamical quantities actually influence the dynamics of the observed system. Since we aim to find minimally invasive data-driven models, we focus on neural network augmentations that have the least possible effect while achieving the goal of high predictive accuracy and generalizability. Inspired by Sparse Input Neural Networks (SPINN) [78], we adapt the sparse-group lasso [306] penalty for the cost function:

$$\mathcal{L} = \sum_t ||f_\theta(s_{t-1}) - s_t^*||_2^2 + \kappa||\theta_{[:1]}||_1 + \lambda||\theta_{[1:]}||_2^2, \tag{4.10}$$

given weighting coefficients $\kappa, \lambda > 0$, where $f_\theta(s_{t-1})$ is the state predicted by the hybrid simulator given the previous simulation state s_{t-1} (consisting of $\mathbf{q}$, $\dot{\mathbf{q}}$), s_t^* is the state from the observed system, $\theta_{[:1]}$ are the network weights of the first layer, and $\theta_{[1:]}$ are the weights of the upper layers. Such cost function sparsifies the input weights so that the augmentation depends on only a small subset of physical quantities, which

is desirable when simpler models are to be found that potentially overfit less to the training data while maintaining high accuracy (Occam's razor). In addition, the $L2$ loss on the weights from the upper network layers penalizes the overall contribution of the neural augmentation to the predicted dynamics since we only want to augment the simulation where the analytical model is unable to match the real system.

We demonstrate the approach on a double pendulum system that is modeled as a frictionless mechanism in our analytical physics engine. The reference system, on the other hand, is assumed to exhibit joint friction, i.e., a force proportional to and opposed to the joint velocity. As expected, compared to a fully learned dynamics model, our hybrid simulator outperforms the baselines significantly (Fig. 4.16). As shown on the right, convergence is achieved while requiring orders of magnitude less training samples. At the same time, the resulting neural network augmentation converges to weights that clearly show that the joint velocity $\dot{\mathbf{q}}_i$ influences the residual joint force τ_i that explains the observed effects of joint friction in joint i (Fig. 4.16 left).

4.2.4.4 Imitation Learning from MPC

Starting from a rigid-body simulation and augmenting it by neural networks has shown significant improvements in closing the sim-to-real gap and learning entirely new effects, such as viscous friction that has not been modeled analytically. In our final experiment, we investigate how the use of learned models can benefit a control pipeline for a quadruped robot. We simulate a Laikago quadruped robot and use a model-predictive controller (MPC) to generate walking gaits. The MPC implements [162] and uses a quadratic programming (QP) solver leveraging the OSQP library [308] that computes the desired contact forces at the feet, given the current measured state of the robot (center of mass (COM), linear and angular velocity, COM estimated position (roll, pitch and yaw), foot contact state as well as the desired COM position and velocity). The swing feet are controlled using PD position control, while the stance

feet are torque-controlled using the QP solver to satisfy the motion constraints. This controller has been open-sourced as part of the quadruped animal motion imitation project [256][†].

We train a neural network policy to imitate the QP solver, taking the same system state as inputs, and predicting the contact forces at the feet which are provided to the IK solver and PD controller. As training dataset we recorded 10k state transitions generated by the QP solver, using random linear and angular MPC control targets of the motion of the torso. We choose a fully connected neural network with 41-dimensional input and two hidden layers of 192 nodes that outputs the 12 joint torques, all using ELU activation [50].

The neural network controller is able to produce extensive walking gaits in simulation for the Laikago robot, and in-place trotting gaits on a real Unitree A1 quadruped (see Fig. 4.6). The learned QP solver is an order of magnitude faster than the original QP solver (0.2 ms inference time versus 2 ms on an AMD 3090 Ryzen CPU).

Thanks to improved performance, the control frequency increased from 160 Hz to 300 Hz. Since the model was trained using data only acquired in simulation, there is some sim-to-real gap, causing the robot to trot with a slightly lower torso height. In future work, we plan to leverage our end-to-end differentiable simulator to fine-tune the policy while closing the simulation-to-reality gap from real-world sensor measurements. Furthermore, the use of differentiable simulators, such as in Um et al. [324], to train data-driven models with faster inference times is an exciting direction to speed up conventional physics engines.

4.2.5 Conclusion

We presented a differentiable simulator for articulated rigid-body dynamics that allows the insertion of neural networks at any point in the computation graph. Such neural augmentation can learn dynamical effects that the analytical model does not account for, while requiring significantly less training data from

[†]The implementation of the QP-based MPC is available open-source at https://github.com/google-research/motion_imitation

the real system. Our experiments demonstrate benefits for sim-to-real transfer, for example in planar pushing where, contrary to the given model, the friction force depends not only on the objects velocity but also on its position on the surface. Effects that have not been present altogether in the rigid-body simulator, such as drag forces due to fluid viscosity, can also be learned from demonstration. Throughout our experiments, the efficient computation of gradients that our implemented simulator facilitates, has considerably sped up the training process. Furthermore, replacing conventional components, such as the QP-based controller in our quadruped locomotion experiments, has resulted in an order of magnitude computation speed-up.

In future work we plan to investigate how physically meaningful constraints, such as the law of conservation of energy, can be incorporated into the training of the neural networks. We will further our preliminary investigation of discovering neural network connections and the viability of such techniques on real-world systems, where hybrid simulation can play a key role in model-based control.

Acknowledgments

We thank Carolina Parada and Ken Caluwaerts for their helpful feedback and suggestions.

4.3 Inferring Articulated Rigid Body Dynamics from RGBD Video

Being able to reproduce physical phenomena ranging from light interaction to contact mechanics, simulators are becoming increasingly useful in more and more application domains where real-world interaction or labeled data are difficult to obtain. Despite recent progress, significant human effort is needed to configure simulators to accurately reproduce real-world behavior. We introduce a pipeline that combines inverse rendering with differentiable simulation to create digital twins of real-world articulated mechanisms from depth or RGB videos. Our approach automatically discovers joint types and estimates their kinematic parameters, while the dynamic properties of the overall mechanism are tuned to attain physically accurate simulations. Control policies optimized in our derived simulation transfer successfully back to the original system, as we demonstrate on a simulated system. Further, our approach accurately reconstructs the kinematic tree of an articulated mechanism being manipulated by a robot, and highly nonlinear dynamics of a real-world coupled pendulum mechanism.

4.3.1 Introduction

From occupancy grids to modern SLAM pipelines, scene representations in robotics are becoming increasingly capable of informing complex behaviors. Approaches such as Kimera [280] are equipping robots with a world model that allows them to intelligently reason about spatiotemporal and semantic concepts of the world around them. While still centered around metric world representations, the interest in dynamics-aware representations is increasing.

Simulators, especially those that incorporate dynamics models, are world representations that can potentially reproduce a vast range of behavior in great detail. Provided these models are calibrated correctly, they can generalize exceptionally well compared to most purely data-driven models. They are an indispensable tool in the design of machines where the cost of prototyping hardware makes iterating in the real

Figure 4.17: Experimental setup of the Craftsman experiment from Section 4.3.4.3. A Franka Emika Panda robot arm pushing an articulated system consisting of wooden toy construction parts via a cylindrical tip. The motion is recorded by an Intel RealSense L515 LiDAR sensor.

world prohibitively expensive. Robot control pipelines are often trained and developed in simulation due to the orders of magnitudes of speed-ups achievable by simulating many interaction scenarios in parallel faster than real time without causing damage in the early phases of training.

Nonetheless, it remains a challenge to leverage such tools for real-world robotic tasks. Not only is there a sim2real gap due to inherent model incompleteness, but it is often difficult to find the correct simulation settings that yield the most accurate results. Despite recent advances in bridging the sim2real gap (see [132] for an overview), deriving a Unified Robot Description Format (URDF) file or analogous scene specifications poses a challenge when a real-world system needs to be simulated accurately.

We tackle the problem of automatically finding the correct simulation description for real-world articulated mechanisms. Given a depth or RGB video of an articulated mechanism undergoing motion, our pipeline determines the kinematic topology of the system, i.e. the types of joints connecting the rigid bodies and their kinematic properties, and the dynamical properties that explain the observed physical behavior. Relying on camera input our pipeline opens the avenue to future work integrating simulators more firmly into the representation stack that robots can use to reason about the physical world around them and make high-level decisions leveraging semantic information that the simulator encodes.

By inferring the number of rigid objects and their relationship to each other without manually specifying such articulations upfront, our work extends previous works that tackle system identification, i.e.

	Graph NN [292, 291]	gradSim [239]	VRDP [62]	Galileo [343]	ScrewNet [147]	Ours
Infers joint topology	implicitly				✓	✓
Articulated physics	✓	✓				✓
Differentiable physics	✓	✓	✓			✓
Differentiable rendering		✓				✓
Real-world ground-truth	✓			✓	✓	✓
Pixel-based observations		✓	✓	✓	✓	✓

Table 4.4: Comparison of selected inference and modeling approaches that reduce the gap between the real and simulated world.

the estimation of dynamical properties of a known model of the physical system. Our proposed pipeline consists of the following four steps:

1. We identify the type and segmentation mask of rigid shapes in one frame of the video.

2. We instantiate a differentiable rasterizer on a scene that consists of the previously identified shapes. Given the depth image sequence from the real system, we optimize the 3D poses of such rigid bodies via gradient-based optimization.

3. We identify the types of joints connecting the rigid bodies based on their motion, and determine the kinematic properties of the mechanism.

4. We infer the dynamical simulation parameters through a gradient-based Bayesian inference approach that combines differentiable rendering and simulation.

Our experiments demonstrate the effectiveness of our proposed pipeline, as well as show the benefits of learning a physical simulator for control applications, and prediction of the behavior of real articulated systems.

4.3.2 Related Work

In Tab. 4.4 we provide a high-level overview of selected related works and how their capabilities compare to our work.

4.3.2.1 Articulation inference

Inferring the type and parameters of joints by observing moving bodies has been an important area in robotic perception, where affordances of objects need to be known for a robot to interact with them. This has motivated interactive perception approaches, where action selection is informed by particle filtering [112] or recursive Bayesian filtering (RBF) [71]. The latter approach builds on earlier work [212] that we adapt in our approach to find the parameters of revolute and prismatic joints given the motion of the rigid transforms of the objects in the scene (see Section 4.3.3.3).

Articulation inference often hinges on accurate pose tracking of the rigid bodies, which is why many early works relied on fiducial markers to accurately track objects and subsequently infer articulations [310, 242, 198]. We do not require markers but do need to have accurate 3D meshes of the objects to be tracked in our approach. Learning-based approaches, such as ScrewNet [147] and DUST-net [146] infer single articulations from depth images, whereas our approach recovers multiple articulations between an arbitrary number of rigid objects in the scene. In [235] and [245] signed distance fields are learned in tandem with the articulation of objects to infer kinematic 3D geometry, without considering the dynamics of the system.

4.3.2.2 Learning simulators

Our approach instantiates a simulation from real-world observations, where the inference of articulations is the first step. Alternative approaches propose different pipelines to the problem of learning a simulator. Entirely data-driven physics models often leverage graph neural networks to discover constraints between particles or bodies ([18, 348, 291]). Physics-based machine learning approaches introduce a physics-informed inductive bias for learning models for dynamical systems from data [269, 205, 311].

Our work is closer to VRDP [62], a pipeline that leverages differentiable simulation to learn the parameters underlying rigid body dynamics. Contrary to our approach, VRDP does not consider articulated

objects. Galileo [343] infers physical dynamics from video via Markov Chain Monte Carlo (MCMC) that samples a simulator as likelihood function, which, in contrast to our work, is not evaluated in image space but on low-dimensional pose tracking data and requires the full simulation with all objects to be manually specified.

4.3.2.3 Differentiable simulation

Differentiable simulation has gained increasing attention, as it allows for the use of efficient gradient-based optimization algorithms to tune simulation parameters or control policies [14, 266, 91, 239], as well as LiDAR sensor models [119].

A particularly related prior work is gradSim [239] that, as in our approach, combines a differentiable physics engine with a differentiable renderer to solve inverse problems from high-dimensional pixel inputs. As in the other related works on differentiable simulators in this review, the simulator needs to have been set up manually before any optimization can begin. This entails that a URDF or other scene description format has to be provided that encodes the system topology and parameters. In this work, we introduce a method that automatically discovers the joint connections between rigid bodies from a mechanism observed through high-level inputs, such as depth or RGB camera images.

4.3.3 Approach

In the following we describe the pipeline of our approach, which we summarize in the diagram in Fig. 4.18.

4.3.3.1 Geometry identification

In the first phase, we use Detectron2 [345], an object detection and instance segmentation network, to find the geometry instances and their segmentation in each frame of the input video. We found input images with three channels per pixel necessary to yield accurate results, so that we converted depth images to

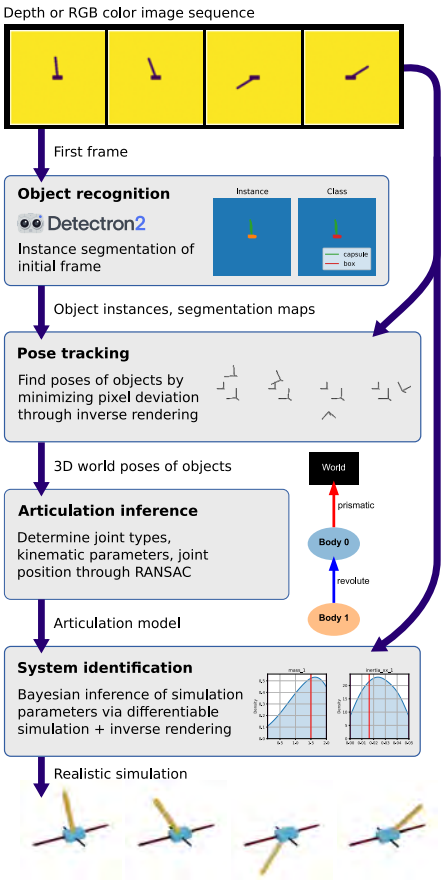


Figure 4.18: **Pipeline.** Our proposed simulation inference approach derives an articulated rigid body simulation from pixel inputs. The shown exemplary results generated by the phases in this diagram stem from the cartpole inference experiment from Section 4.3.4.1.

normal maps via finite differencing in pixel space. We generated two datasets in this work to train Detectron2: a synthetic set of normal maps of primitive shapes (capsules, boxes, spheres) of various sizes and in various configurations, and a dataset of camera images for our real-robot experiment in Section 4.3.4.3. An exemplar segmentation map predicted by Detectron2 on our Craftsman experiment is shown in Fig. 4.19.

Given the segmentation map from the first step, we track the poses of the rigid bodies over time. We leverage nvdiffrast [175], an efficient, GPU-driven differentiable rasterizer to generate an observation $\mathbf{x}$ (can be either a depth or RGB color image) given the 3D vertices of the object meshes $i \in 1, \ldots, M$ corresponding to the rigid bodies of interest in the scene. Such 3D vertices are transformed based on the world poses T_0^i of the shapes we want to track. To solve the inverse rendering problem, i.e. finding object

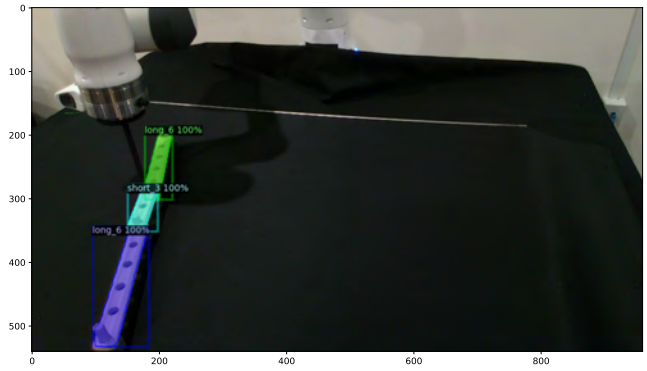


Figure 4.19: **Segmentation map prediction.** Detectron2 segmentation map predicted from an RGB image of a mechanism made of wooden parts from the Craftsman toolkit (see Section 4.3.4.3). The three pieces and their types of mesh (either `long_6` or `short_3`) have been correctly identified.

poses given pixel inputs, we minimize the L2 distance between the real-world image and the simulated rendering at each frame of the input video:

$$\underset{[T_0^1, T_0^2, \ldots, T_0^M]}{\text{minimize}} \| f_{\text{rast}}([T_0^1, T_0^2, \ldots, T_0^M]) - \mathbf{x} \|^2,$$

where f_{rast} denotes the differentiable rasterizer function.

4.3.3.2 Rigid body tracking

Note that our approach requires the background of the scene, i.e., the static bodies not part of the system, to be instantiated in the simulator if they influence the dynamics. In the case of a table-top manipulation task (e.g. the one from Fig. 4.17), this may be the floor plane which is necessary for contact mechanics. Furthermore, the camera pose needs to be known upfront, and the 3D meshes of the objects in the mechanism have to be given.

4.3.3.3 Joint identification and kinematic tracking

We use the rigid pose trajectories to infer the types of joints that connect the rigid bodies, as well as the joint parameters (e.g. axis of rotation, pivot point, etc.). We follow a RANSAC approach [81] that yields robust results despite noisy pose predictions. For each pair of rigid bodies, we set up one RANSAC estimator per

joint type (model equations follow below): revolute, prismatic, and static (fixed) joint. We select the joint model with the least model error found during the RANSAC iterations. Given the guess for the joint type and its parameters, we compute the joint positions q over time.

Given the sequence of world transform T_0^i of all rigid bodies i in the scene, in the joint identification step we determine for each unique pair of rigid bodies i and j the relative transform $T_i^j[t]$ at each time step t. In the following, $T.r$ denotes the 3D axis-angle rotation vector, and $T.p$ the translation vector of a rigid pose T.

Revolute joint model Determine joint axis $\mathbf{s}$, pivot point $\mathbf{p}$ and joint angle q from two consecutive transforms:

$$\Delta r = T_i^j[t+1].r - T_i^j[t].r \qquad \Delta p = T_i^j[t+1].p - T_i^j[t].p$$
$$\mathbf{s} = \frac{\Delta r}{\|\Delta r\|} \qquad \mathbf{p} = T_i^j[t].p + \frac{\Delta r \times \Delta p}{\|\Delta r\|^2} \qquad q = \|\Delta r\|$$

Prismatic joint model Determine joint axis $\mathbf{s}$ and joint position q from two consecutive transforms:

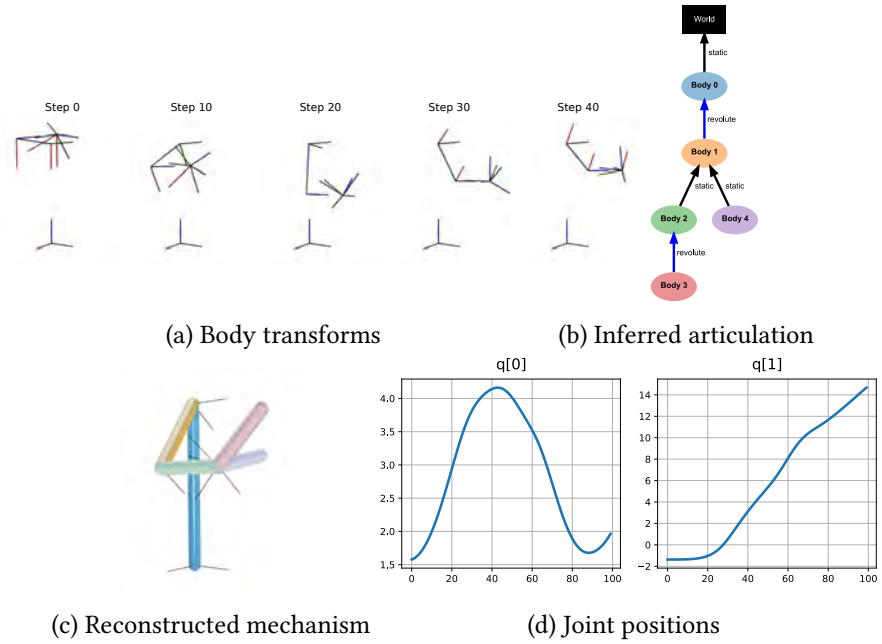
$$\Delta p = T_i^j[t+1].p - T_i^j[t].p$$
$$\mathbf{s} = \frac{\Delta p}{\|\Delta p\|} \qquad q = \mathbf{s} \cdot T_i^j[t+1].p$$

Static joint model The static (also known as fixed) joint is parameterized by the constant relative transform T_i^j between bodies i and j.

Algorithm 6 summarizes our inference approach to determine the articulations between rigid bodies given their observed motions. We first find the most likely joint types and corresponding joint parameters between unique pairs of rigid bodies via RANSAC for the three different joint models (revolute, prismatic

Algorithm 6 Determine articulation of observed motion

Require: world transform sequence T_0^i for each rigid body $i = 1..n$
  $C = \mathbf{0}_{n \times n}$
  for body $i \in \{1..n\}$ **do**
    for body $j \in \{i{+}1..n\}$ **do**
      Calculate relative transform sequence T_i^j
      Determine joint parameters θ_{joint} for revolute (r), prismatic (p), static (s) joint type given T_i^j via RANSAC, as well as their respective model errors c_r, c_p, c_s
      if RANSAC found at least one joint candidate **then**
        $C[i, j] = C[j, i] = \min\{c_r, c_p, c_s\}$
        Memorize θ^*_{joint} of candidate with lowest cost
      else
        $C[i, j] = C[j, i] = \infty$
      end if
    end for
  end for
  Determine minimum spanning forest on C, retrieve root bodies I_{root}
  for body $i \in I_{root}$ **do**
    Construct kinematic tree A rooted at body i, with parent-child connections from the corresponding minimum spanning tree and respective memorized joint parameters
    Determine joint candidate θ_{joint} via RANSAC given T_0^i
    if RANSAC found at least one joint candidate **then** ▷ fixed-base case
      Attach A to world via lowest-cost joint model θ^*_{joint}
    else ▷ floating-base case
      Attach A to world via free joint
    end if
  end for
  return world model consisting of articulations



(a) Body transforms (b) Inferred articulation

(c) Reconstructed mechanism (d) Joint positions

Figure 4.20: Inference of articulations in a complex mechanism consisting of static and revolute joints.

and static). If no joint model could be found that matches the relative transform sequence between two rigid bodies, they are considered to be disconnected.

Having computed the cost matrix C of the joint model errors from the previous RANSAC estimation, we find the minimum spanning forest via Prim's algorithm that we run on each component of the undirected graph described by the weighted adjacency matrix C. We select the root node i from each minimum spanning tree as the top-level body in the kinematic tree to which all the rigid bodies within the same component are connected via the previously found joint models according to the hierarchy of the spanning tree. Note that since the joint models have been determined for a particular ordering of unique pairs of rigid bodies (to avoid duplicate computation), the direction the spanning tree imposes on the particular connection may require the joint model (e.g., in the case of revolute joints, the joint axis and pivot point) to be inverted.

Given the root node's sequence of world transforms T_0^i, we determine the most likely joint model again via RANSAC. If such a model has been found, the corresponding articulated mechanism is considered fixed-base and gets connected through this joint to the world. If no such joint model could be found,

the articulation is floating-base and needs to be considered as such in the world model (either by adding degrees of freedom corresponding to a rigid-body motion, or via a flag that ensures the mechanism is simulated as a floating-base system).

4.3.3.4 Simulation parameter estimation

Thus far, the pipeline was dedicated to the inference of the kinematic properties of the mechanism we observed. Having found the joint topology and approximate transforms between the bodies in the system, we will now consider the dynamics of the mechanism. This entails finding simulation parameters and an initial state vector that yield a motion, which when rendered via our differentiable rasterizer, closely matches the given image sequence.

Leveraging Bayesian inference, we infer the dynamical and (optionally, for fine-tuning) kinematic parameters (such as joint axes and pivot points) of the mechanism. We model the inference problem as a Hidden Markov Model (HMM) where the observation sequence $\mathcal{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_T]$ of T video frames is derived from latent states $\mathbf{s}_t$ ($t \in [1..T]$). We assume the observation model is a deterministic function which is realized by the differentiable rasterization engine that turns a system state $\mathbf{s}_t$ into an observation image $\mathbf{x}_t$ (either depth or RGB color). The states are advanced through the dynamics model which we assume is fully dependent on the previous state and the simulation parameter vector θ. In our model, this transition function is assumed to be the differentiable simulator that implements the articulated rigid-body dynamics equations and contact models. We use the Tiny Differentiable Simulator [122] that implements end-to-end differentiable contact models and articulated rigid-body dynamics following Featherstone's formulation [76]. For an articulated rigid body system, the parameters θ may include the masses, inertial properties and geometrical properties of the bodies in the mechanism, as well as joint and contact friction coefficients.

Following Bayes' law, the posterior $p(\theta|\mathcal{X})$ over simulation parameters $\theta \in \mathbb{R}^M$ is calculated via $p(\theta|D_\mathcal{X}) \propto p(D_\mathcal{X}|\theta)p(\theta)$.

We leverage the recently introduced Constrained Stein Variational Gradient Descent (CSVGD) algorithm [117] that adds constraints to the gradient-based, nonparametric Bayesian inference method SVGD. Such constraint handling allows us to enforce parameter limits and optimize simulation parameters via *multiple shooting*. This technique splits up the trajectory into shooting windows for which the start states need to be learned. Defect constraints are introduced that enforce continuity at the start and end states of adjacent shooting windows. Despite requiring extra variables to be optimized, multiple shooting significantly improves the convergence of gradient-based parameter inference approach when parameters need to be inferred from long time horizons.

4.3.3.5 Limitations

We assume the possible geometric shapes of the rigid bodies of interest in the scene are known, so that they can be rendered. The sensor remains at a static pose, its intrinsics are known. The mechanism to be simulated must be in the viewport at least at the beginning of the video. The tracking and inference currently does not run in real time.

4.3.4 Experiments

We evaluate our approach on three mechanisms.

4.3.4.1 Simulated Cartpole

In our first experiment we consider a simulated cartpole in the Bullet physics engine [54], which allows us to evaluate the accuracy of our method against the known ground-truth simulation parameters. As shown through the exemplary results in Fig. 4.18, we follow our pipeline to infer a realistic simulation given a sequence of 200 depth images. The cart and pole are correctly identified by Detectron2 as a box and

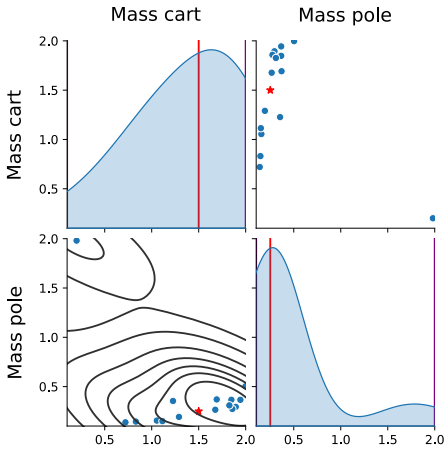


Figure 4.21: Parameter posterior distribution of the two link masses of the cartpole. Inference from depth video via the multiple-shooting Bayesian inference approach CSVGD. The red lines and stars indicate the ground-truth parameters.

capsule, which are then tracked via our inverse rendering approach. The subsequent articulation inference step find the expected joint topology where the pole is connected through a revolute joint with the cart, which in turn is connected by a prismatic joint to the world. Finally, we infer the simulation parameters related to the inertial properties of the two links (masses and inertia matrix diagonal entries). We visualize the posterior distribution found by CSVGD over the two link masses in Fig. 4.21, and compare the error over the inferred parameters compared to other methods in the top row of Tab. 4.5, which demonstrates that our approach is able to closely recover the true parameter settings (shown in red). In particular, the gradient-based estimators Adam and CSVGD achieve a lower normalized (w.r.t. parameter limits) parameter error within 2000 optimization steps than the CMA-ES [111] and MCMC method. We choose 16 particles both for CSVGD and CMA-ES. For MCMC, we also use 16 particles via the parallel, *smooth-step* sampling scheme from the Emcee library [83]. We select the particle with the highest likelihood for the comparison in Tab. 4.5.

4.3.4.2 Rott's Chaotic Pendulum

In our next experiment, we aim to identify a real-world chaotic mechanism – a coupled pendulum first analyzed by Nikolaus Rott [282]. Rott's mechanism consists of two pendula with resonant frequencies at approximately two to one. One L-shaped pendulum is attached to a fixed pivot via a revolute joint, and a

	Adam	CSVGD (Ours)	CMA-ES	MCMC	GT
Parameter error:					
NMAE	0.299	0.161	0.369	0.328	0
Average control reward (mean±std dev):					
Swing-up	0.222 ± 0.072	0.235 ± 0.059	0.190 ± 0.104	0.229 ± 0.071	0.230 ± 0.077
Balancing	0.734 ± 0.081	0.742 ± 0.081	0.779 ± 0.073	0.744 ± 0.128	0.751 ± 0.093

Table 4.5: *Top section:* normalized mean absolute error on the inferred parameters of the various methods compared against the true parameters. *Bottom section:* average reward statistics obtained from 10 training runs of the MPPI controller evaluated on the Bullet cartpole system from Section 4.3.5.3 on the swing-up and balancing tasks.

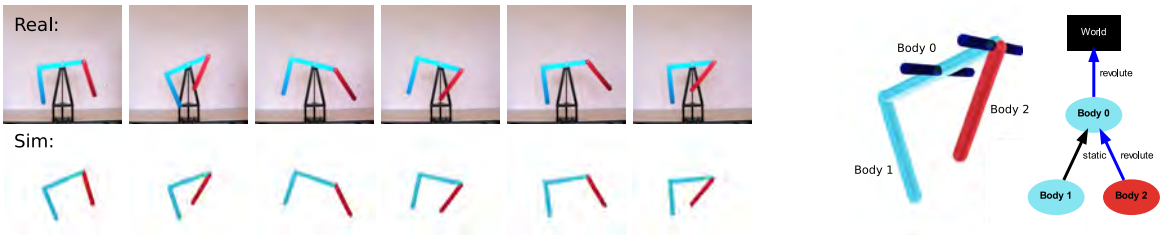
single body is attached to this L-shaped pendulum via another revolute joint. Given a video taken with an RGB camera of such a mechanism, we aim to reconstruct a digital twin in simulation.

In the first step of our pipeline, we identify the rigid objects in the video by converting the RGB images to binary images with an appropriate threshold so that the background becomes white and the mechanism itself appears white. By applying a blur filter to the image, we obtain a fake depth which is sufficient for the Detectron2 network to identify the three links as capsules. We manually designate the instances of the segmentation map found by Detectron2 to pregenerated capsule shapes in our simulator.

Since we cannot rely on depth information to inform the 3D poses of the rigid bodies, we assume the mechanism to be a planar system. Therefore, the rigid body tracking system is constructed such that each body only has three degrees of freedom (x, z position and yaw angle). We set up the rasterizer to produce RGB images (where the blue and red color has been manually assigned to the correct capsule shapes upfront).

4.3.4.3 Real articulated system

In our final experiment we reconstruct a simulation from a real-world articulated system being pushed by a robot on a table. We build an articulated mechanism with the "Craftsman" toy toolkit which contains wooden levers and linkages. We command a Franka Emika Panda robot arm equipped with a cylindrical



(a) Simulation derived from RGB video

(b) Inferred articulation

Figure 4.22: Our proposed framework infers articulated rigid body dynamics simulations from video. In this example, Rott's pendulum is identified from real RGB camera footage of the system in motion. In (a), the original video is compared to the simulated motion in the learned physics engine. In (b), the inferred joints are visualized on the left, where blue cylinders correspond to the axes of the revolute joints. The estimated kinematic tree is shown on the right.

	Adam	CSVGD (Ours)	CMA-ES	MCMC	PhyDNet
Cartpole depth image sequence					
MSE	0.0015	0.0008	0.0018	0.0019	0.1665
MAE	0.0073	0.0042	0.0089	0.0093	0.4056
SSIM	0.9136	0.9452	0.8908	0.8893	0.7985
Rott's pendulum RGB image sequence					
MSE	0.0112	0.0134	0.0122	0.0177	0.0054
MAE	0.0175	0.0219	0.0187	0.0274	0.0153
SSIM	0.8929	0.8370	0.8774	0.8001	0.9412

Table 4.6: Video forecasting performance. For each of the experiments, the models were evaluated by observing the first 10 frames from the test dataset and predicting the next 150 frames.

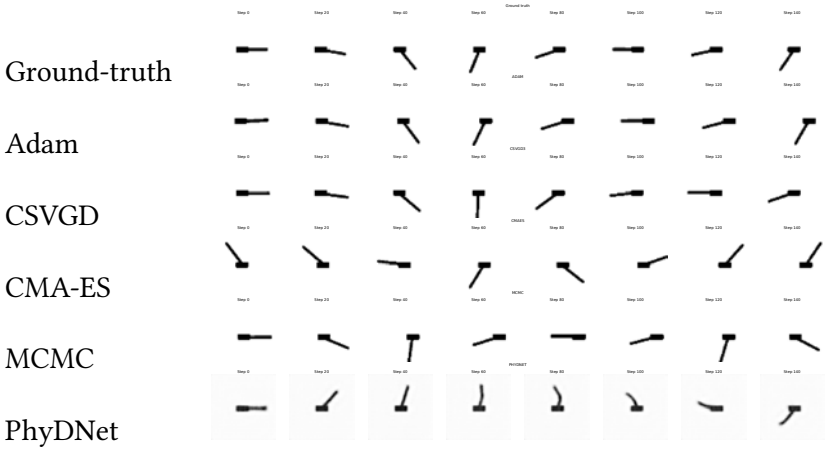


Figure 4.23: Video forecasting results on the test dataset of a simulated cartpole. 150 frames must be predicted given the first 10 frames of the motion.

tip at its end-effector to move sideways along the y direction by 60 cm over a duration of 20 s at a constant velocity.

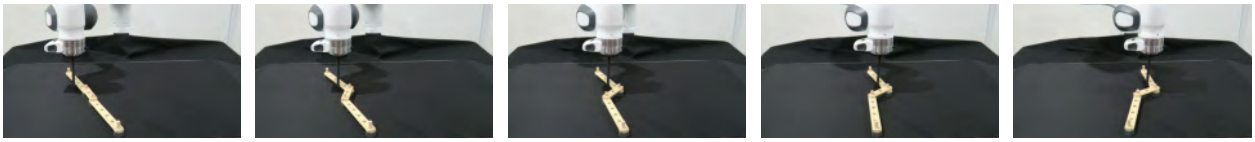
We record RGB and depth video with a RealSense L515 LiDAR sensor. The depth images are aligned to the color images by considering the differing resolution and field-of-view of the two input modalities. Since the depth output from the RealSense L515 LiDAR sensor does not have sufficient resolution to clearly identify the sub-centimeter parts of the Craftsman construction kit, we choose the RGB image stream as input to the instance segmentation model. The image segmentation model is a mask region-based convolutional neural network (R-CNN) pretrained on the COCO instance segmentation dataset [192]. We collected a dataset of 30 RGB images of three Craftsman parts (two different types of parts) in various spatial configurations for finetuning Detectron2 on our domain.

Given the instance segmentation map, we extract the corresponding 3D point cloud segments for each instance (see Fig. 4.19) and use iterative closest point to estimate an initial pose of each shape from the construction kit. Given this initialization, as before, the poses of the Craftsman parts are optimized through the differentiable rasterizer, where the camera has been aligned in the simulator from a perspective-n-point calibration routine. As shown in Fig. 4.24, we are able to discover the two revolute joints connecting the three links of the mechanism, as well as the planar floating base (prismatic x, y and revolute z joints). The inferred joint positions allow the kinematic tree (shown in (b)) to move in a way that closely matches the real motion (a).

4.3.5 Results

4.3.5.1 Rigid Pose Tracking

We investigate the accuracy of the rigid pose tracking phase in our approach on the simulated cartpole system which provides us access to the true 6D poses (3D positions x, y, z, and 3D rotations rx, ry, rz



(a) Real motion

(b) Inferred joints and positions

Figure 4.24: Joint inference for the Craftsman system. The articulation is inferred; the two revolute joints are indicated by blue cylinders in (b).

in axis-angle form) of the two rigid objects (cart and pole) in the system. As shown in Fig. 4.25, our inverse rendering approach leveraging gradient-based optimization achieves a significantly higher accuracy compared to a classical pose tracking approach, namely point-to-point iterative closest point (ICP). For ICP, we generate point clouds from the known object meshes via Poisson disk sampling [352], and align them to the point clouds from the depth images in the cartpole dataset (given the initial alignment at the first time step from our inverse rendering tracking approach). ICP achieves an MAE over the 6D poses of 0.755, compared to our inverse rendering approach which achieves a MAE of 0.024. The significantly higher accuracy in pose tracking is the enabler of our articulation inference phase which relies on pose estimates with few outliers and overall low noise to clearly identify the different joint types. For example, even from small changes along the x axis (as in the case of the cart in the top row of Fig. 4.25), the cart's prismatic joint can be correctly inferred due to the overall low noise level in our pose estimates.

4.3.5.2 Video Forecasting

In Tab. 4.6, we compare the performance of our method on video forecasting against PhyDNet [104]. PhyDNet introduces "PhyCells" to disentangle physical dynamics knowledge from residual information and generates PDE-constrained future frame predictions. We extract the first 195 frames from the Rott's

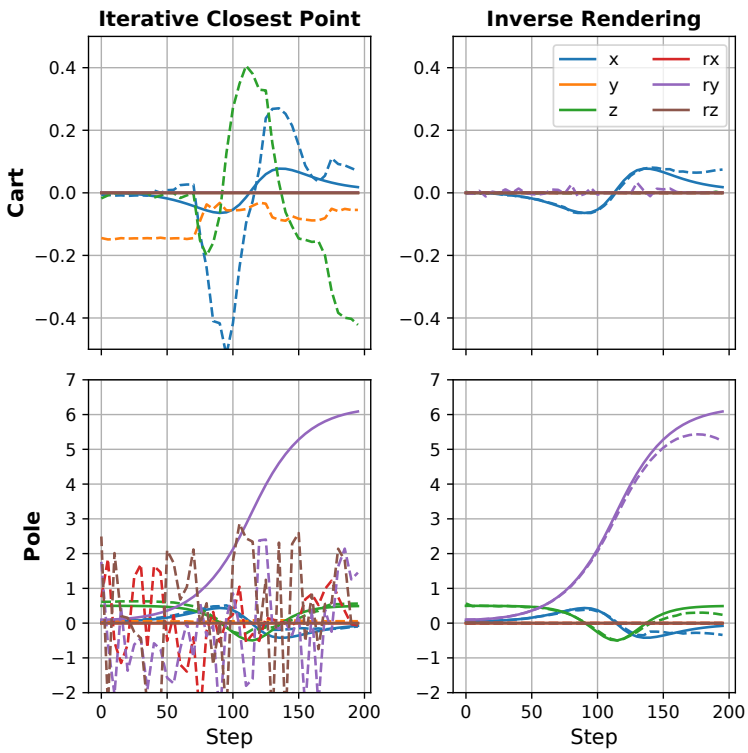


Figure 4.25: The point-to-point iterative closest point (ICP) algorithm (left column) and our inverse rendering approach (right column) are compared on the tracking of the rigid poses of the cart (top row) and the pole (bottom row) in the simulated cartpole experiment from Section 4.3.4.1. The ground-truth coordinates of the 6D poses are shown in solid lines, the corresponding estimated results are indicated by dashed lines.

pendulum video and generate 175 sequences of length 20 for training, and test on frames in the later part of the video. Given the first 10 frames of the sequence as input, PhyDNet predicts the next 150 frames, which we compare against the GT frames.

The results indicate that purely vision-based model PhyDNet has an advantages where the test dataset is close to the training data, as is the case in the videos of Rott's pendulum where in both cases the pendulum calmly swings back and forth. While outperforming our simulation-based approach on all metrics, including the Structural Similarity (SSIM) index, on the cartpole test data the results clearly show an advantage of the strong inductive bias our simulation-based inference approach provides. With the accurate parameters settings (see NMAE in the top section of Tab. 4.5) CSVGD finds, it is able to outperform all the other inference approaches on all the video forecasting accuracy metrics.

4.3.5.3 Model-Predictive Control

Given the inferred simulator for the observed cartpole system, we train a controller in simulation and investigate how well it performs on the original system.

We leverage Model Predictive Path Integral (MPPI) [336], an information-theoretic model predictive control (MPC) algorithm that has been shown to being particularly suited to control nonlinear systems.

In the swing-up task, the cartpole is resting with its pole pointing downwards. For the balancing task, the cart has a $0.1\,\mathrm{m\,s^{-1}}$ velocity applied to it, and the pole is at an angle of $20°$ from its upright position at the beginning. For both tasks the goal is to bring it into an upright position while keeping the cart and pole velocities as small as possible.

The bottom two rows of Tab. 4.5 summarize the average rewards over a trajectory length of 200 steps (with time step 0.05 s) obtained by the MPPI controller running with 200 samples per step with a lookahead window length of 60 time steps. We compare the returns of the controller run on the inferred system with the most likely parameter configuration found by the parameter inference algorithms. For reference, in the last column, we report the mean reward on the ground-truth cartpole simulation (GT).

The inferred dynamics model with the parameters found by CSVD allow the MPPI controller to swing up the cartpole and maintain the pole in an almost upright position without falling back down. Similarly the balancing task succeeds after an initial phase where the pole swings downwards.

4.3.6 Conclusions

Our proposed pipeline allows the automatic inference of articulated rigid-body simulations from video by leveraging differentiable physics simulation and rendering. Our results on a simulated system demonstrate that we can achieve accurate trajectory predictions that benefit model-based control, while the learned parameters are physically meaningful. On a real-world coupled pendulum system, our approach predicts the correct joint topology and results in a simulation that accurately reproduces the real RGB video of the

mechanism. In future work, we are planning to incorporate the learning of geometric shapes, and improve our implementation and algorithms to achieve real-time simulation inference.

Acknowledgements

We thank Chris Denniston and David Millard for their feedback and discussion around this work, as well as Vedant Mistry for his contributions to an earlier prototype of our implementation. We are grateful to Maria Hakuba, Hansjörg Frei and Christoph Schär for kindly permitting us to use their video of Rott's mechanism[‡] in this work.

[‡] https://youtu.be/dhZxdV2naw8. Accessed on Feb. 22, 2022.

Chapter 5

Conclusion

In this thesis, we proposed simulation-based inference and control as a methodology to overcome the reality gap in robotics simulators. To this end, we developed methods that facilitate the calibration of physics engines to real-world sensor measurements. Differentiable simulation, in particular, enables efficient algorithms to leverage gradients of the simulation parameters and control variables to design new mechanisms, improve the predictive performance through system identification, and optimize motion plans. Besides applications in LiDAR sensing and robotic cutting, we introduced a framework that combines data-driven models with differentiable simulators, a novel Bayesian inference algorithm that is particularly tailored to such gradient-based models, and a pipeline that finds articulations in videos and creates fully interactive simulations from real systems. Next to the transfer learning techniques that exist to improve the robustness of robot controllers facing the sim2real gap, we see simulation-based inference as another important pillar of the robot learning methodology.

5.1 Future Applications

Beyond robot learning, the capability of automatically generating and updating simulators from real data offers wide-ranging future applications:

- Closed-loop model-predictive controllers that leverage the simulator as world model which is kept synchronized via online system identification from new measurements as they are perceived from interacting with the real world. This could enable robots to autonomously explore the environment to build a more detailed simulation. Furthermore, the high fidelity that a physics engine affords could allow intelligent agents to virtuously manipulate complex systems where effects from fluid mechanics, among others, have to be accounted for, such as in autonomous flight under challenging wind conditions.

- High-level reasoning abilities in robot control, such as task and motion planning, where the semantic information that the simulator encodes implicitly is leveraged in symbolic reasoning methods. At the same time, we have the opportunity to update these symbols from measurements through our simulation-based inference techniques. Logic-geometric programming has been shown to yield such capabilities given a differentiable physics model, where it was applied to tool use in simulated environments [321].

- Human-interpretable robot assistants that can, at any time during their deployment, present their current state of the simulated world to the user, providing deep insights into the reasoning and estimation process. This could greatly ease the debugging of robot control pipelines, since the simulator as a white-box world representation is well understood by the human operator. As is becoming common practice in autonomous driving, where dangerous scenarios encountered in the real world can be re-enacted from sensor measurements of the vehicle, the simulated world at the time of when a problem occurred can be used to generate training scenarios or counterfactual test cases for the robot controllers to improve in simulation before they are deployed to the real world again.

- Automatic creation of virtual worlds, which is a core enabler of virtual reality (VR) applications. Beyond reproducing the scene geometry and surface materials for visual fidelity, the ability to physically simulate objects we encounter in the real world will allow people to haptically interact with them in VR from anywhere. The applications here are wide-spread, ranging from gaming and education to tele-operation and -diagnosis of systems at remote locations.

5.2 Final Remarks

The results we presented in our work have tapped into the potential of simulation-based inference and control. However, considerable engineering and research effort is needed to overcome computational challenges, such as the runtime performance achievable by simulation-based methods, and resolve theoretical questions, e.g., how to combine learning-based approaches with analytical physics models to dynamically trade off speed with accuracy depending on the application needs, or how to improve the convergence of optimization algorithms operating on problems involving discontinuous and highly nonlinear systems, among many other. As progress is made in all of these areas, we believe that simulation-based inference and control will bring significant advancements to the field of robotics and beyond.

Appendix A

Probabilistic Inference of Simulation Parameters via Parallel Differentiable Simulation

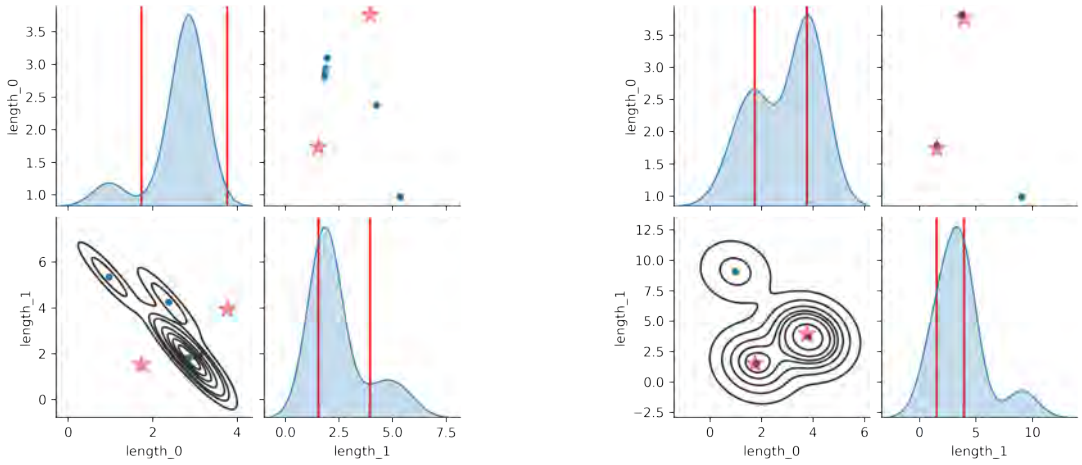
A.1 Additional Technical Details

In this section we provide further technical details on our approach.

A.1.1 Initializing the Estimators

For each experiment, we initialize the particles for the estimators via the deterministic Sobol sequence on the intervals specified through the parameter limits. Our code uses the Sobol sequence implementation from Burkardt [35] which is based on a Fortran77 implementation by Fox [84]. For the MCMC methods that sample a single Markov chain, we used the center point between the parameter limits as initial guess.

A.1.2 Likelihood Model for Sets of Trajectories



(a) Product (b) Sum

Figure A.1: Comparison of posterior parameter distributions obtained from fitting the parameters to two ground-truth trajectories generated from different link lengths of a simulated double pendulum (units of the axes in meters). The trajectories were 300 steps long (which corresponds to a length of 3 s) and contain the 2 joint positions and 2 joint velocities of the uncontrolled double pendulum which starts from a zero-velocity initial configuration where the first angle is at $90°$ (sideways) and the other at $0°$. In (a), the product of the individual per-trajectory likelihoods is maximized (Eq. A.2). In (b), the sum of the likelihoods is maximized (Eq. A.1).

In this work we assume that trajectories may have been generated by a distribution over parameters. In the case of a replicable experimental setup, this could be a point distribution at the only true parameters. However, when trajectories are collected from multiple robots, or with slightly different experimental setups between experiments, there may be a multimodal distribution over parameters which generated the set of trajectories.

Note, that irrespective of the choice of likelihood function we do not make any assumption about the shape of the posterior distribution by leveraging SVGD which is a non-parametric inference algorithm. In trajectory space, the Gaussian likelihood function is a common choice as it corresponds to the typical least-squares estimation methodology. Other likelihood distributions may be integrated with our method, which we leave up to future work.

The likelihood which we use is a mixture of equally-weighted Gaussians centered at each reference trajectory $\mathcal{X}^{\text{real}}$:

$$p_{sum}(D_{\mathcal{X}}^{\text{real}}|\theta) = \sum_{\mathcal{X}^{\text{real}} \in D_{\mathcal{X}}^{\text{real}}} p_{ss}(\mathcal{X}^{\text{real}}|\theta). \tag{A.1}$$

If we were to consider each trajectory as an independent sample from the same trajectory distribution (the product), the likelihood function would be

$$p_{product}(D_{\mathcal{X}}^{\text{real}}|\theta) = \prod_{\mathcal{X}^{\text{real}} \in D_{\mathcal{X}}^{\text{real}}} p_{ss}(\mathcal{X}^{\text{real}}|\theta). \tag{A.2}$$

While both Eqs. (A.1) and (A.2) define the likelihood for a combination of single-shooting likelihood functions p_{ss} for a set of real trajectories $D_{\mathcal{X}}^{\text{real}}$, the same combination operators (sum or product) apply to the combination of multiple-shooting likelihood functions p_{ms} analogously.

The consequence of using these likelihoods can be seen in Fig. A.1 where Fig. A.1a shows the resulting posterior distribution (in parameter space) when treating a set of trajectories as independent and taking the product of their likelihoods (Eq. A.2), while Fig. A.1b shows the result of treating them as a sum of Gaussian likelihoods (Eq. A.1). In Fig. A.1a the posterior becomes the average of the two distributions since that is the most likely position that generated both of the distinct trajectories. In contrast, the posterior approximated by the same algorithm (CSVGD) but using the sum of Gaussian likelihoods, successfully captures the multimodality in the trajectory space since most particles have aligned near the two modes of the true distribution in parameter space.

A.1.3 State and Parameter Normalization

The parameters we are estimating are valid over only particular ranges of values. These ranges are often widely different - in the case of our real-world pendulum experiment, the center of mass of a link in a pendulum may be in the order of centimeters, while the angular velocity at the beginning of the recorded

motion can reach values on the orders of meters per second. It is therefore important to scale the parameters to a common range to avoid any dimension to dominate smaller parameter ranges during the estimation.

Similarly, the state dimensions are of different units - for example, we typically include velocities and positions in the recorded data over which we compute the likelihood. Therefore, we also normalize the range over the state dimensions. Given the state vector, respective parameter vector, w, we normalize each dimension i by its statistical variance σ^2, i.e. w_i/σ_i^2.

A.1.4 KNN-based Approximation for KL Divergence

In this work, we compare a set of parameter guesses (particles) to the ground-truth parameters, or a set of trajectories generated by simulating trajectories from each parameter in the particle distribution to a set of trajectories created on a physical system. To compare these distributions, we use the KL divergence to determine how the two distributions differ from each other. Formally, the KL divergence is the expected value of the log likelihood ratio between two distributions, and is an asymmetric divergence that does not satisfy the triangle inequality.

The KL divergence is easily computable in the case of discrete distributions or simple parametric distributions, but is not easily calculable for samples from non-parametric distributions such as those over trajectories. Instead, we use an approximation to the KL divergence which uses the relative distances between samples in a set to estimate the KL divergence between particle distributions. This method has been used to compare particle distributions over robot poses to asses the performance of particle filter distributions [47]. To estimate the KL divergence between particle distributions over trajectories $D_{\mathcal{X}}^p$ and $D_{\mathcal{X}}^q$ we adopt the formulation from [328, 47]:

$$\tilde{d}_{\mathrm{KL}}(D_{\mathcal{X}}^p \parallel D_{\mathcal{X}}^q) = \frac{N}{|D_{\mathcal{X}}^p|} \sum_{i=1}^{|D_{\mathcal{X}}^p|} \log \frac{\mathrm{KNN}_{k_i}^p(i)}{\mathrm{KNN}_{l_i}^q(i)} + \frac{1}{|D_{\mathcal{X}}^p|} \sum_{i=1}^{|D_{\mathcal{X}}^p|} [\psi(l_i) - \psi(k_i)] + \log \frac{|D_{\mathcal{X}}^q|}{|D_{\mathcal{X}}^p| - 1}, \tag{A.3}$$

where N is the dimensionality of the trajectories, $|D^p_{\mathcal{X}}|$ is the number of trajectories in the $D^p_{\mathcal{X}}$ dataset, $|D^q_{\mathcal{X}}|$ is the number of particles in the $D^q_{\mathcal{X}}$ dataset, $\text{KNN}^p_{k_i}(i)$ is the distance from trajectory $\mathcal{X}_i \in D^p_{\mathcal{X}}$ to its k_i-th nearest neighbor in $D^q_{\mathcal{X}}$, $\text{KNN}^q_{l_i}(i)$ is the distance from trajectory $\mathcal{X}_i \in D^p_{\mathcal{X}}$ to its l_i-th nearest neighbor in $D^p_{\mathcal{X}} \backslash \mathcal{X}_i$, and ψ is the digamma function. Note that this approximation of KL divergence can also be applied to compare parameter distributions, as we show in the synthetic data experiment from Section 4.1.5.1 (cf. Fig. 4.4a and Fig. 4.4b) where the ground-truth parameter distribution is known.

Throughout this work, we set k_i and l_i to 3 as this reduces the bias in the approximation, but does not require a large amount of samples from the ground-truth distribution.

A.2 Experiments

In the following, we provide further technical details and results from the experiments we present in the main paper.

A.2.1 Differentiable Simulator

Other than requiring a differentiable forward dynamics model which allows to simulate the system in its entirety following the Markov assumption, our proposed algorithm does not rely on a particular choice of dynamical system or simulator for which its parameters need to be estimated. For our experiments, we use the Tiny Differentiable Simulator [122] that implements end-to-end differentiable contact models and the Articulated Body Algorithm (ABA) [76] to compute the forward dynamics (FD) for articulated rigid-body mechanisms. Given joint positions $\mathbf{q}$, velocities $\dot{\mathbf{q}}$, torques τ in generalized coordinates, and external

forces $\mathbf{f}^{\text{ext}}$, ABA calculates the joint accelerations $\ddot{\mathbf{q}}$. We use semi-implicit Euler integration to advance the system dynamics in time for a time step Δt:

$$\begin{aligned}
\ddot{\mathbf{q}}_{t+1} &= \text{ABA}(\mathbf{q}_t, \dot{\mathbf{q}}_t, \tau_t, \mathbf{f}_t^{\text{ext}}; \theta), \\
\dot{\mathbf{q}}_{t+1} &= \dot{\mathbf{q}}_t + \ddot{\mathbf{q}}_{t+1}\Delta t, \\
\mathbf{q}_{t+1} &= \mathbf{q}_t + \dot{\mathbf{q}}_{t+1}\Delta t.
\end{aligned} \tag{A.4}$$

The second-order ODE described by Eq. A.4 is lowered to a first-order system, with state $\mathbf{s}_t = \begin{bmatrix} \mathbf{q}_t & \dot{\mathbf{q}}_t \end{bmatrix}$. Furthermore, we deal primarily with the discrete time-stepped dynamics function $\mathbf{s}_{t+1} = f_{\text{step}}(\mathbf{s}_t, t, \theta)$, assuming that Δt is constant. The function $f_{\text{sim}}(\theta, \mathbf{s}_0)$ uses f_{step} iteratively to produce a trajectory of states $[\mathbf{s}]_{t=1}^{T}$ given an initial state $\mathbf{s}_0$ and the parameters θ. Many systems of practical interest for robotics are controlled by an external input. In our formulation for parameter estimation, we include controls as explicit dependencies on the time parameter t.

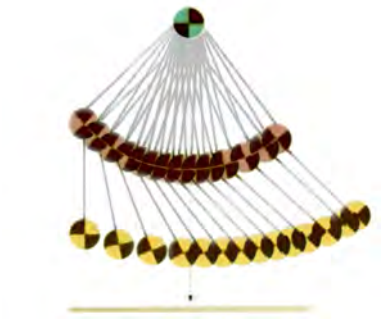
For an articulated rigid body system, the parameters θ may include (but are not limited to) the masses, inertial properties and geometrical properties of the bodies in the mechanism, as well as joint and contact friction coefficients. Given $\frac{\partial f_{\text{step}}}{\partial \theta}$ and $\frac{\partial f_{\text{step}}}{\partial \mathbf{s}}$, gradients of simulation parameters with respect to the state trajectories can be computed directly through the chain rule, or via the adjoint sensitivity method [263].

A.2.2 Identify Real-world Double Pendulum

We set up the double pendulum estimation experiment with the 11 parameters shown in Fig. A.2a to be estimated. The state space consists of the two positions and velocities of both joints: $\mathbf{s} = \begin{bmatrix} \mathbf{q}_{0:1} & \dot{\mathbf{q}}_{0:1} \end{bmatrix}$. The dataset of trajectories contains image sequences (see time lapse of an excerpt from a trajectory in Fig. A.2b) and annotated pixel coordinates of the three vertices in the double pendulum, from which we

Link	Parameter	Minimum	Maximum
Link 1	Mass	0.05 kg	0.5 kg
	I_{xx}	0.002 kg m^2	1.0 kg m^2
	COM x	-0.2 m	0.2 m
	COM y	-0.2 m	0.2 m
	Joint friction	0.0	0.5
Link 2	Length	0.08 m	0.3 m
	Mass	0.05 kg	0.5 kg
	I_{xx}	0.002 kg m^2	1.0 kg m^2
	COM x	-0.2 m	0.2 m
	COM y	-0.2 m	0.2 m
	Joint friction	0.0	0.5

(a) Parameters to be estimated. I refers to the 3×3 inertia matrix, COM stands for center of mass.



(b) Time lapse of a double pendulum trajectory from the IBM dataset [9].

Figure A.2: Physical double pendulum experiment from Sec. A.2.2.

extracted joint positions and velocities (via finite differencing given the known recording frequency of 400 Hz).

Since we know that all trajectories in this dataset stem from the same double pendulum [9], we only used a single reference trajectory as target trajectory $\mathcal{X}^{\text{real}}$ during the estimation. We let each estimator run for 2000 iterations. For evaluation, we calculate the consistency metrics from Tab. 4.1 over 10 held-out trajectories from a test dataset. For comparison, we visualize the trajectory density over simulations rolled out from the last 100 Markov samples (or 100 particles in the case of particle-based approaches) in Fig. A.3. The ground-truth shown in these plots again stems from the unseen test dataset. This experiment further demonstrates the generalizability of simulation-based inference, which, when an adequate model has been implemented and its parameters identified, can predict outcomes under various novel conditions even though the training dataset consisted of only a single trajectory in this example.

A.2.3 Ablation Study on Multiple Shooting

We evaluate the baseline estimation algorithms with our proposed multiple-shooting likelihood function (using 10 shooting windows) on the physical double pendulum dataset from before. To make the constrained optimization problem amenable to the MCMC samplers, we formulate the defect constraint

Algorithm	$d_{\mathrm{KL}}(D_{\mathcal{X}}^{\mathrm{real}} \parallel D_{\mathcal{X}}^{\mathrm{sim}})$		$d_{\mathrm{KL}}(D_{\mathcal{X}}^{\mathrm{sim}} \parallel D_{\mathcal{X}}^{\mathrm{real}})$		MMD	
	SS	MS	SS	MS	SS	MS
Emcee	**8542.2466**	8950.4574	4060.6312	N/A	1.1365	N/A
CEM	8911.1798	**8860.5115**	8549.5927	N/A	0.9687	**0.5682**
SGLD	8788.0962	**5863.2728**	7876.0310	**2187.2825**	2.1220	**0.0759**
NUTS	9196.7461	**8785.5326**	6432.2131	**4935.8983**	**0.5371**	1.1642
(C)SVGD	8803.5683	**5204.5336**	10283.6659	**2773.1751**	0.7177	**0.0366**

Table A.1: Consistency metrics of the posterior distributions approximated from the physical double pendulum dataset (Sec. A.2.2) by the different estimation algorithms using the single-shooting likelihood $p_{ss}(\mathcal{X}^{\mathrm{real}}|\theta)$ (column "SS") and the multiple-shooting likelihood $p_{ms}(\mathcal{X}^{\mathrm{real}}|\theta)$ (column "MS") with 10 shooting windows. Note that SVGD with multiple-shooting corresponds to CSVGD.

through the likelihood defined in Eq. 4.5 where we tune σ_{def}^2 to a small value (on the order of 10^{-2}) such that the defects are minimized during the estimation. As we describe in Section 4.1.4.3, the parameter space is augmented by the shooting variables $\mathbf{s}_t^s$.

As shown in Tab. A.1, the MCMC approaches Emcee and NUTS do not benefit meaningfully from the multiple-shooting approach. Emcee often yields unstable simulations from which we are not able to compute some of the metrics. The increased dimensionality of the parameter space appears to add a significant challenge to these methods, which are known to scale poorly to higher dimensions. Despite being configured to use a Gaussian mixture model of 3 kernels, the CEM posterior immediately collapses to a single point such that the KL divergence of simulated against real trajectories cannot be computed.

We observe a significant improvement in estimation accuracy on SGLD, where the multiple-shooting approach allowed it to converge to closely matching trajectories, as shown in Fig. A.4. As with SVGD, the availability of gradients allows this method to scale better to the higher dimensional parameter space, while the smoothed likelihood landscape further helps the approach to find better fitting parameters.

A.2.4 Comparison to Likelihood-free Inference

Our Bayesian inference approach leverages the simulator as part of the likelihood model to approximate posterior distributions over simulation parameters, which means the simulator is indispensable in our estimation process. In the following, we compare our approach against the likelihood-free inference approach BayesSim [274] that leverages approximate Bayesian computation (ABC) which is the most popular family of algorithms within likelihood-free methods.

Likelihood-free methods assume the simulator is a black box that can generate a trajectory given a parameter setting. Instead of querying the simulator to evaluate the likelihood (as in our approach), a conditional density $q(\theta|D_{\mathcal{X}})$ is learned directly from a dataset of simulation parameters and their corresponding rolled-out trajectories via supervised learning to approximate the posterior. A common choice of model for such density is a mixture density network [28], which parameterizes a Gaussian mixture model. This is in contrast to our (C)SVGD algorithm that can approximate any shape of posterior by being a nonparametric inference algorithm.

For our experiments we collect a dataset of 10,000 simulated trajectories of parameters randomly sampled from the prior distribution. We train the density network via the Adam optimizer with a learning rate of 10^{-3} for 3000 epochs, after which we observed no meaningful improvement to the calculated log-likelihood loss during training. In the following, we provide further details on the likelihood-free inference pipeline we consider, by describing the input data processing and the model used for approximating the posterior.

A.2.4.1 Input Data Processing

The input to the learned density model has to be a sufficient statistic of the underlying data, while being low-dimensional in order to keep the learning problem computationally tractable. We consider the following four methods of processing the trajectories that are the input to the likelihood-free methods, as

visualized for an example trajectory in Fig. A.5. Note that we configured the following input processing methods to generate a one-dimensional input vector that has a reasonable length to be computationally feasible to train on (given the 10,000 trajectories from the training dataset), while achieving acceptable performance which we validated through testing various settings.

Downsampled we down-sample the trajectory so that for the double pendulum experiment (Sec. A.2.2) we use only every 20th state, for the Panda arm experiment (Section 4.1.5.3) only every 200-th state of the trajectory. Finally, the state dimensions per trajectory are concatenated to a one-dimensional vector.

Difference we adapt the input statistic from the original BayesSim approach in [274, Eq. (22)] where the differences of two consecutive states along the trajectory are used in concatenation with their mean and variance:

$$\psi(\mathcal{X}) = (\text{downsample}(\tau), \mathbb{E}[\tau], \text{Var}[\tau]),$$
$$\text{where } \tau = \{\mathbf{s}_t - \mathbf{s}_{t-1}\}_{t=1}^{T}$$

As before, we down-sample these state differences and concatenate them to a vector.

Summary for each state dimension of the trajectory, we compute the following statistics typical for time series: mean, variance, cross correlation between state dimensions of the trajectory, as well as auto-correlations for each dimension at 5 different time delays: [5, 10, 20, 50, 100] time steps. These numbers are concatenated for all state dimensions to a one-dimensional vector per input trajectory.

Signature we compute the signature transform from the signatory package [161] over the input trajectory. Such so-called path signatures have been recently introduced to extract information about order and area, thereby preserving features inherent to nonlinear trajectories. We select a depth for the signature

transform of 3 for the double pendulum experiment, and 2 for the Panda arm experiment, to obtain feature vectors of comparable size to the aforementioned input techniques.

A.2.4.2 Density Model

As the density model for the learned posterior $q(\theta|D_{\mathcal{X}})$, we select the following commonly used representations.

Mixture density network (MDN) uses neural network features from a feed-forward neural network using two hidden layers with 24 units each.

Mixture density random Fourier features (MDRFF) this density model uses Fourier features and a kernel. We evaluate the MDRFF with the following common choices for the kernel:

- Radial Basis Function (**RBF**) kernel
- **Matérn** kernel [275, Equation (4.14)] with $\nu = 5/2$

A.2.4.3 Evaluation

Note that instead of action generation, which is part of the proposed BayesSim pipeline [274], we only focus on the inference of the posterior density over simulation parameters in order to compare such likelihood-free inference approach against our method.

Finally, to evaluate the metrics shown in Tab. A.2 for each BayesSim instantiation (input method and density model), we sample 100 parameter vectors from the learned posterior $q(\theta|D_{\mathcal{X}})$ and simulate them to obtain 100 trajectories which are compared against the reference trajectory sets, as we did in the comparison for the other Bayesian inference methods in Tab. 4.1.

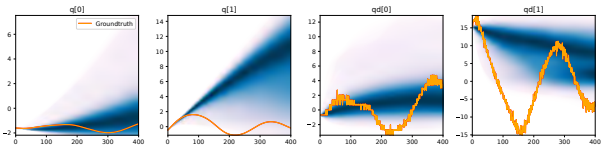
Input	Model	Double Pendulum Experiment			Panda Arm Experiment
		$d_{\mathrm{KL}}(D_{\mathcal{X}}^{\mathrm{real}} \parallel D_{\mathcal{X}}^{\mathrm{sim}})$	$d_{\mathrm{KL}}(D_{\mathcal{X}}^{\mathrm{sim}} \parallel D_{\mathcal{X}}^{\mathrm{real}})$	MMD	$\log p_{\mathrm{obs}}(D_{\mathcal{X}}^{\mathrm{real}} \parallel D_{\mathcal{X}}^{\mathrm{sim}})$
Downsampled	MDN	8817.9222	4050.4666	0.6748	-17.4039
Difference	MDN	8919.2463	4633.2637	0.6285	-17.1646
Summary	MDN	9092.5575	5093.8851	0.5664	-18.3481
Signature	MDN	8985.8056	4610.5438	0.5807	-19.3432
Downsampled	MDRFF (RBF)	9027.9474	5091.5283	0.5593	-17.2335
Difference	MDRFF (RBF)	8936.3823	4282.8599	0.5988	-18.4892
Summary	MDRFF (RBF)	9063.1753	4884.1398	0.5672	-19.5430
Signature	MDRFF (RBF)	8980.9080	4081.1160	0.6016	-18.3458
Downsampled	MDRFF (Matérn)	8818.1830	3794.9873	0.6110	-17.6395
Difference	MDRFF (Matérn)	8859.2156	4349.9971	0.6176	-17.2752
Summary	MDRFF (Matérn)	8962.0501	4241.4551	0.5999	-19.6672
Signature	MDRFF (Matérn)	9036.9626	4620.9517	0.5715	-18.1652
CSVGD		**5204.5336**	**2773.1751**	**0.0366**	**-15.1671**

Table A.2: Consistency metrics of the posterior distributions approximated by the different BayesSim instantiations, where the input method and model name (including the kernel type for the MDRFF model) are given. Each metric is calculated across simulated and real trajectories. Lower is better on all metrics except the log-likelihood $\log p_{\mathrm{obs}}(D_{\mathcal{X}}^{\mathrm{real}} \parallel D_{\mathcal{X}}^{\mathrm{sim}})$ from the Panda arm experiment. For comparison, in the last row, we reproduce the numbers from CSVGD shown in Tab. 4.1.

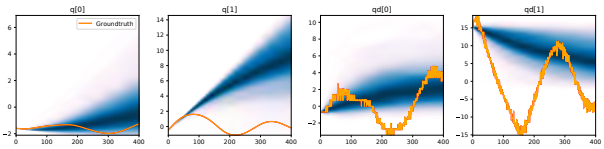
A.2.4.4 Discussion

The results from our experiments with the various likelihood-free approaches in Tab. A.2 indicate that, among the tested pipelines, the MDRFF model with Matérn kernel and downsampled trajectory input overall performed the strongest, followed by the MDN with downsampled input. In comparison to the likelihood-based algorithms from Tab. 4.1, these results are comparable on the double pendulum experiment. However, in comparison to CSVGD, the estimated likelihood-free posteriors are significantly less accurate, which can also be clearly seen in the density plots over the rolled out trajectories from such learned densities in Tab. A.3. On the Panda arm experiment, the likelihood-free methods are outperformed by the likelihood-based algorithms (such as the Emcee sampler) more often on the likelihood of the learned parameter densities. CSVGD again achieves a much more accurate posterior in this experiment than any likelihood-free approach.

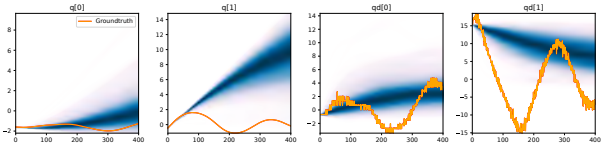
Why do these likelihood-free methods perform so poorly on a seemingly simple double pendulum? One would expect that this kind of dynamical system poses no greater challenge to BayesSim when it was



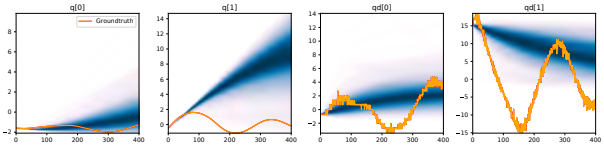
MDN – Downsampled



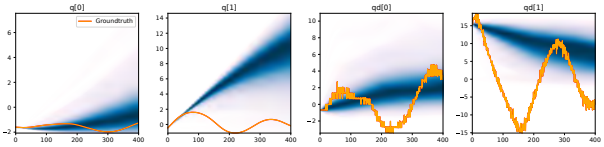
MDN – Difference



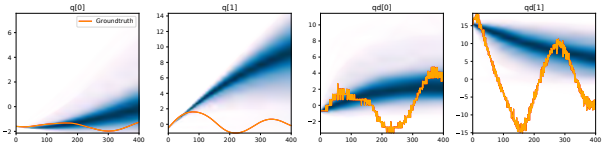
MDN – Summary



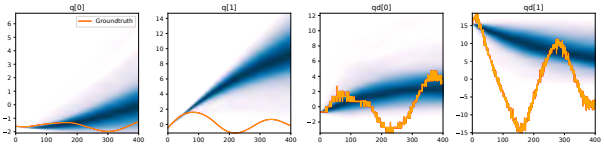
MDN – Signature



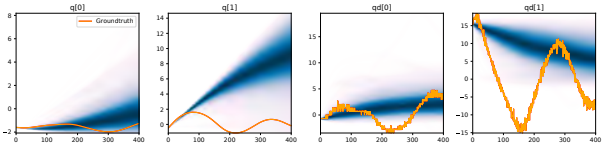
MDRFF (RBF) – Downsampled



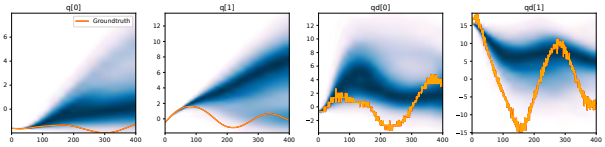
MDRFF (RBF) – Difference



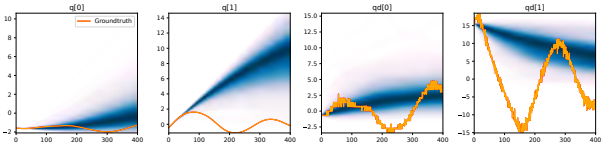
MDRFF (RBF) – Summary



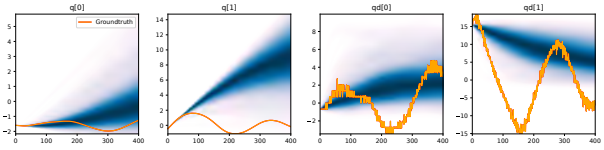
MDRFF (RBF) – Signature



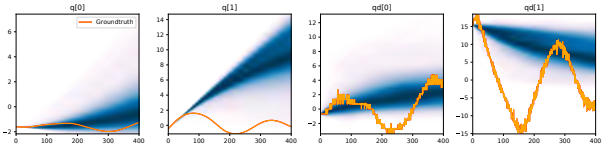
MDRFF (Matérn) – Downsampled



MDRFF (Matérn) – Difference



MDRFF (Matérn) – Summary



MDRFF (Matérn) – Signature

Table A.3: Kernel density estimation over trajectory roll-outs from 100 parameter samples drawn from the posterior of each BayesSim method (model name with kernel choice in bold font + input method, see Sec. A.2.4), applied to the physical double pendulum dataset from Sec. A.2.2. The ground-truth trajectory here stems from the test dataset of 10 trajectories that were held out during training.

shown to identify a cartpole's link length and cart mass successfully [274]. To investigate this problem, we revisit the simplified double pendulum estimation experiment from Section 4.1.5.2 of our main paper, where only the two link lengths need to be estimated from simulated trajectories. As before, we create a dataset with 10,000 trajectories of 400 time steps based on the two simulation parameters sampled from a uniform distribution ranging between 0.5 m and 5 m. While keeping all parameters the same as in our previous double-pendulum experiment where eleven parameters had to be inferred, all of the density models in combination with both the "difference" and "downsampled" input statistic infer a highly accurate parameter distribution, as shown in Fig. A.6a. The trajectories produced by sampling from the BayesSim posterior (Fig. A.6b) also match the reference observations much more closely than any of the BayesSim models on the previous 11-parameter double pendulum (Tab. A.3). These results suggest that BayesSim and potentially other likelihood-free method have problems in inferring higher dimensional parameter distributions. The experiments in [274] demonstrated as many as four parameters being estimated (for the acrobot), while showing inference results for simulated systems only. While our double pendulum system from Sec. A.2.2 is basic in principle, the higher dimensional parameter space (see parameters in Fig. A.2a) and the fact that we need to fit against real-world data makes it a significantly harder problem for most state-of-the-art inference algorithms. CSVGD is able to achieve a close fit thanks to the multiple-shooting segmentation of the trajectory which improves the convergence (see more ablation results for multiple-shooting on this experiment in Sec. A.2.3).

A.2.5 Identify Inertia of an Articulated Rigid Object

The state space consists of the positions and velocities of the seven degrees of freedom of the robot arm and the two degrees of freedom in the universal joint, resulting in a 20-dimensional state vector $\mathbf{s} = \begin{bmatrix} \mathbf{q}_{0:8} & \dot{\mathbf{q}}_{0:8} & \mathbf{q}^d_{0:6} & \dot{\mathbf{q}}^d_{0:6} \end{bmatrix}$ consisting of nine joint positions and velocities, plus the PD control position and velocity targets, $\mathbf{q}^d$ and $\dot{\mathbf{q}}^d$, for the actuated joints of the robot arm. We control the arm using the

MoveIt! motion planning framework [51] by moving joints 6 and 7 to predefined joint-space offsets of 0.1 and -0.1 radians, in sequence. We use the default Iterative Parabolic Time Parameterization algorithm with a velocity scaling factor of 0.1. We track the motion of the acrylic box via a Vicon motion capture system and derive four Cartesian coordinates as observation $\mathbf{x} = \begin{bmatrix} \mathbf{p}_o & \mathbf{p}_x & \mathbf{p}_y & \mathbf{p}_z \end{bmatrix}$ to represent the frame of the box (shown in Fig. A.7): a point of origin located at the center of the upper lid of the box, and three points located 1 m away from the origin into the x, y, and z direction (transformed by the reference frame of the box). We chose this state representation to ease the computation of the likelihood, since we only need to compute differences between 3D points instead of computing the distances over 3D rotations which requires special treatment [143].

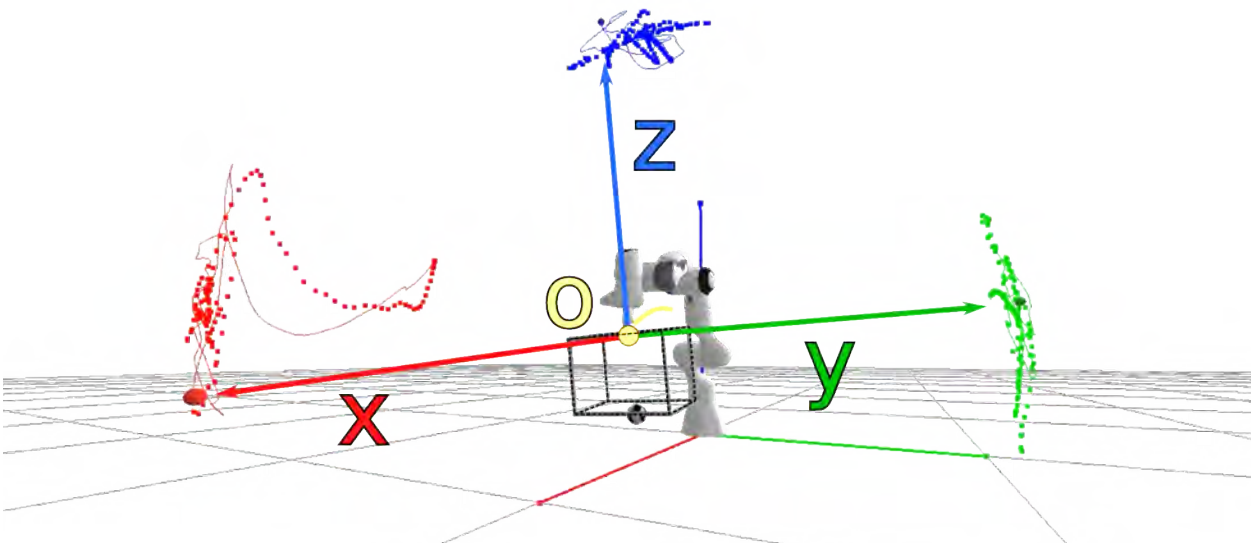


Figure A.7: Rendering of the simulation for the system from Section 4.1.5.3, where the four reference points for the origin, unit x, y, and z vectors are shown. The trace of the simulated trajectory is visualized by the solid lines, the ground-truth trajectories of the markers are shown as dotted lines.

We first identify the simulation parameters pertaining to the inertial properties of the box and the friction parameters of the universal joint. As shown in Tab. A.4a, the symmetric inertia matrix of the box is fully determined by the first six parameters, followed by the 3D center of mass. We have measured the mass to be 920 g, so we do not need to estimate it. We simulate the universal joint with velocity-dependent

	Parameter	Minimum	Maximum
Box	I_{xx}	0.05 kg m^2	0.1 kg m^2
	I_{yy}	0.05 kg m^2	0.1 kg m^2
	I_{zz}	0.05 kg m^2	0.1 kg m^2
	I_{xy}	-0.01 kg m^2	0.01 kg m^2
	I_{xz}	-0.01 kg m^2	0.01 kg m^2
	I_{yz}	-0.01 kg m^2	0.01 kg m^2
	COM x	-0.005 m	0.005 m
	COM y	-0.005 m	0.005 m
	COM z	0.1 m	0.4 m
U-Joint	Friction DOF 1	0.0	0.15
	Friction DOF 2	0.0	0.15

(a) Phase I

	Parameter	Minimum	Maximum
Weight 1	Position x	-0.14 m	0.14 m
	Position y	-0.08 m	0.08 m
Weight 2	Position x	-0.14 m	0.14 m
	Position y	-0.08 m	0.08 m

(b) Phase II

Table A.4: Parameters to be estimated and their ranges for the two estimation phases of the underactuated mechanism experiment from Section 4.1.5.3.

damping, with possibly different friction coefficients for both degrees of freedom. The simulation parameters yielding the most accurate fit to a ground-truth trajectory from the physical robot shaking an empty box is shown in Fig. A.8. We were able to find such parameters via SVGD, CSVGD and Emcee (shown is a parameter configuration from the particle distribution estimated by CSVGD with the highest likelihood).

While the simulated trajectory matches the real data significantly better after the inertial parameters of the empty box have been identified (Fig. A.8b) than before (Fig. A.8a), a reality gap remains. We believe this to be a result from a slight modeling error that the rigid body simulator cannot capture, e.g. the top of the box where the universal joint is attached bends slightly while the box is moving, and there may be slight geometric offsets between the real system and the model of it we use in the simulator. The latter parameters could have been further identified with our approach, nonetheless the simulation given the identified parameters is sufficient to be used in the next phase of the inference experiment.

Given the parameters found in the first phase, we now investigate how well the various approaches can cope with dependent variables. By fixing two 500 g to the bottom of the acrylic box, the 2D locations of such weights need to be inferred. Naturally, such assignment is symmetric, i.e. weight 1 and 2 can swap locations without affecting the dynamics. What would significantly alter the dynamics, however, is an unbalanced configuration of the weights which would cause the box to tilt.

We use 50 particles and run each estimation algorithm for 500 iterations. For each baseline method, we carefully tuned the hyper parameters to facilitate a fair comparison. Such tuning included selecting an appropriate measurement noise variance, which, as we observed on Emcee and SGLD in particular, had a significant influence on the exploration behavior of these algorithms. With a larger observation noise variance the resulting posterior distribution became wider, however we were unable to attain such behavior with the NUTS estimator whose iterates quickly collapsed to a single point at the center of the box (see Fig. A.9d). Similarly, CEM immediately became stuck in the suboptimal configuration shown in Fig. A.9b. Nonetheless, after 500 iterations, all methods predicted weight positions that were aligned opposite to one another to balance the box.

As can be seen in Fig. A.9e, SVGD achieves a fairly predictive posterior approximation, with many particles aligned close to the true vertical position at $y = 0$. With the introduction of the multiple-shooting constraints, Constrained SVGD (CSVGD) converges significantly faster to a posterior distribution that accurately captures the true locations of the box, while retaining the exploration performance of SVGD that spreads out the particles over multiple modes, as shown in Fig. A.9f.

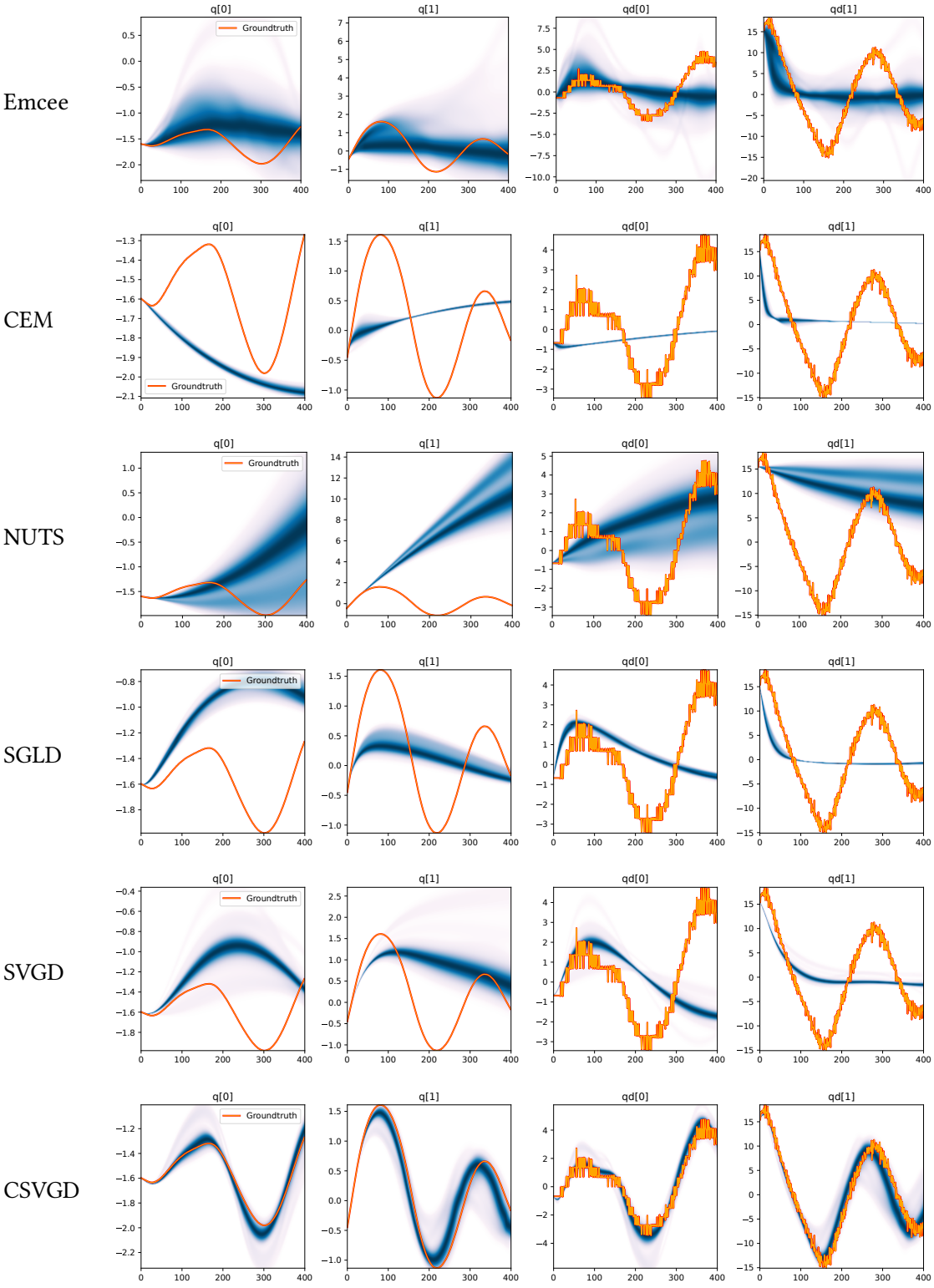


Figure A.3: Kernel density estimation over trajectory roll-outs from the last estimated 100 parameter guesses of each method, applied to the physical double pendulum dataset from Sec. A.2.2. The ground-truth trajectory here stems from the test dataset of 10 trajectories that were held out during training. The particle-based approaches (CEM, SVGD, CSVGD) use 100 particles.

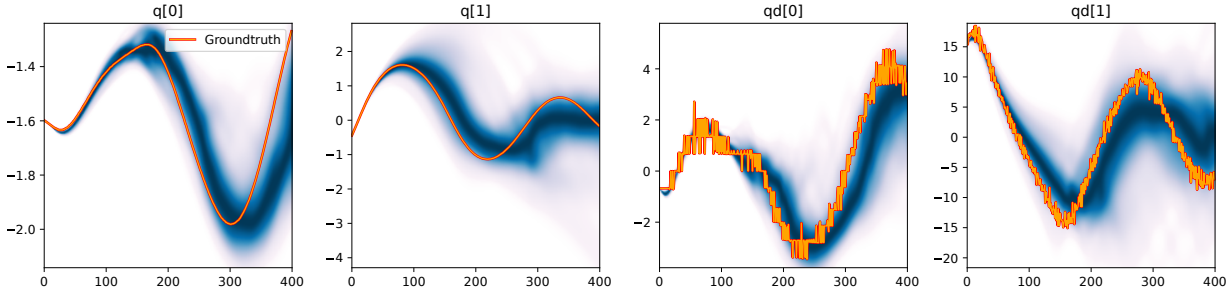
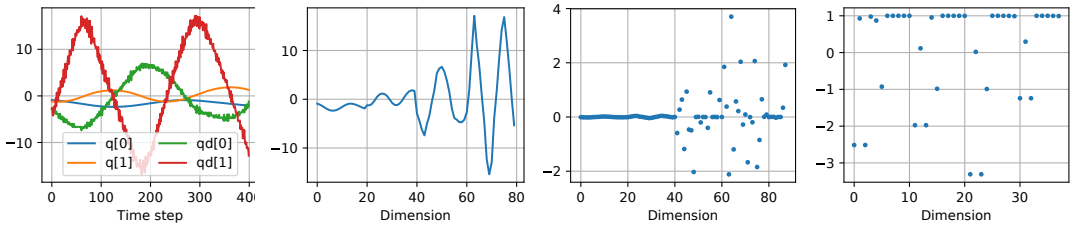
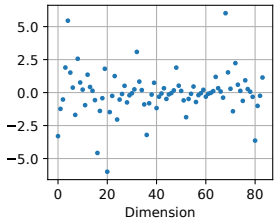


Figure A.4: Kernel density estimation over trajectory roll-outs from the last estimated 100 parameter guesses of SGLD with the multiple-shooting likelihood model (see Sec. A.2.3), applied to the physical double pendulum dataset from Sec. A.2.2. Similarly to SVGD, SGLD benefits significantly from the smoother likelihood function while being able to cope with the augmented parameter space thanks to its gradient-based approach.

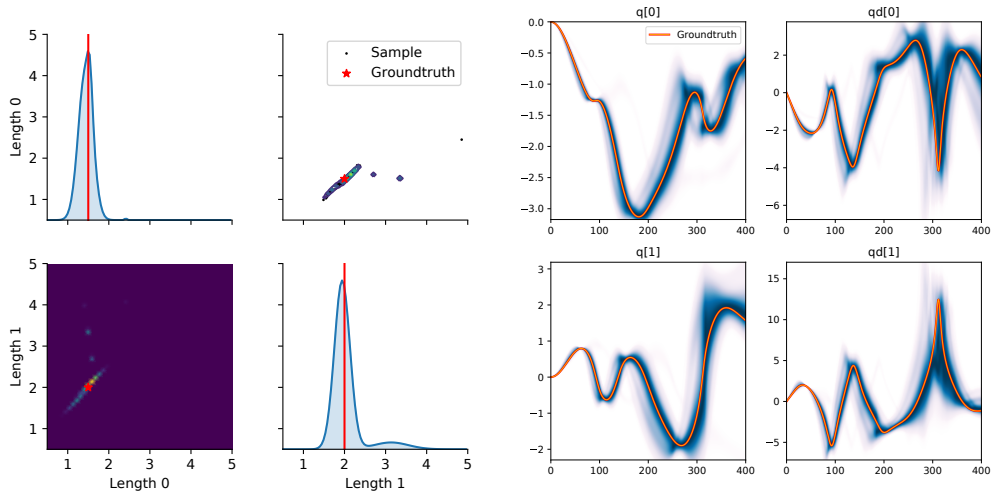


(a) Raw Input (b) Downsampled (c) Difference (d) Summary



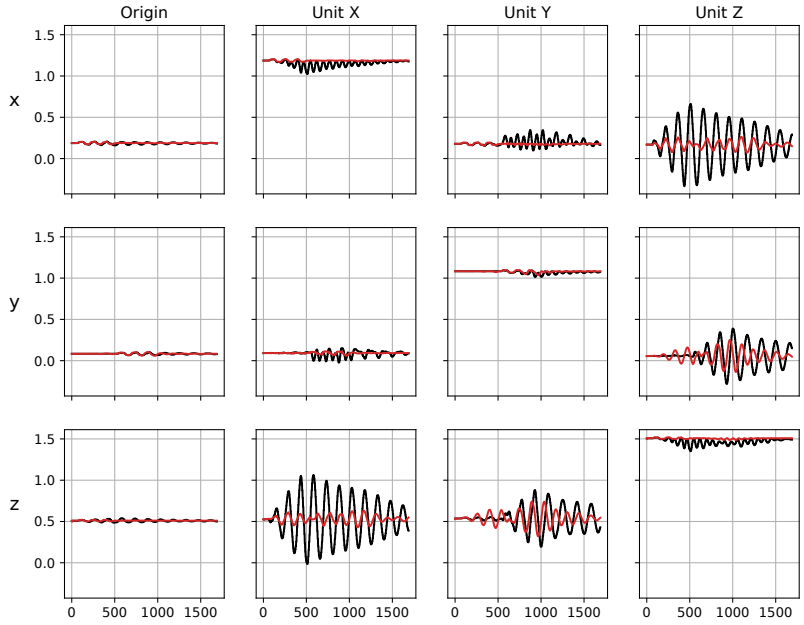
(e) Signature

Figure A.5: Exemplary visualization of the input processing methods for the likelihood-free baselines from Sec. A.2.4 applied to a trajectory from the double pendulum experiment in Sec. A.2.2.

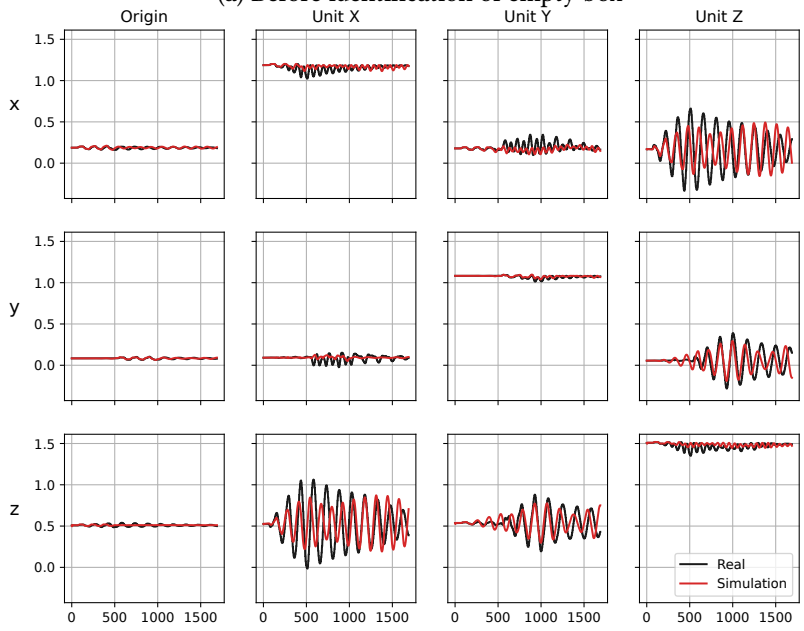


(a) Posterior (b) Trajectory density

Figure A.6: Results from BayesSim on the simplified double pendulum experiment where only the two link lengths need to be inferred. (a) shows the approximated posterior distribution by the MDN model and "downsampled" input statistics. The diagonal plots show the marginal parameter distributions, the bottom-left heatmap and the top-right contour plot show the 2D posterior where the ground-truth parameters at (1.5 m, 2 m) are indicated by a red star. The black dots in the top-right plot are 100 parameters sampled from the posterior which are rolled out to generate trajectories for the trajectory density plot in (b). (b) shows a kernel density estimation over these 100 trajectories for the four state dimensions $(\mathbf{q}_{[0:1]}, \dot{\mathbf{q}}_{[0:1]})$ of the double pendulum.

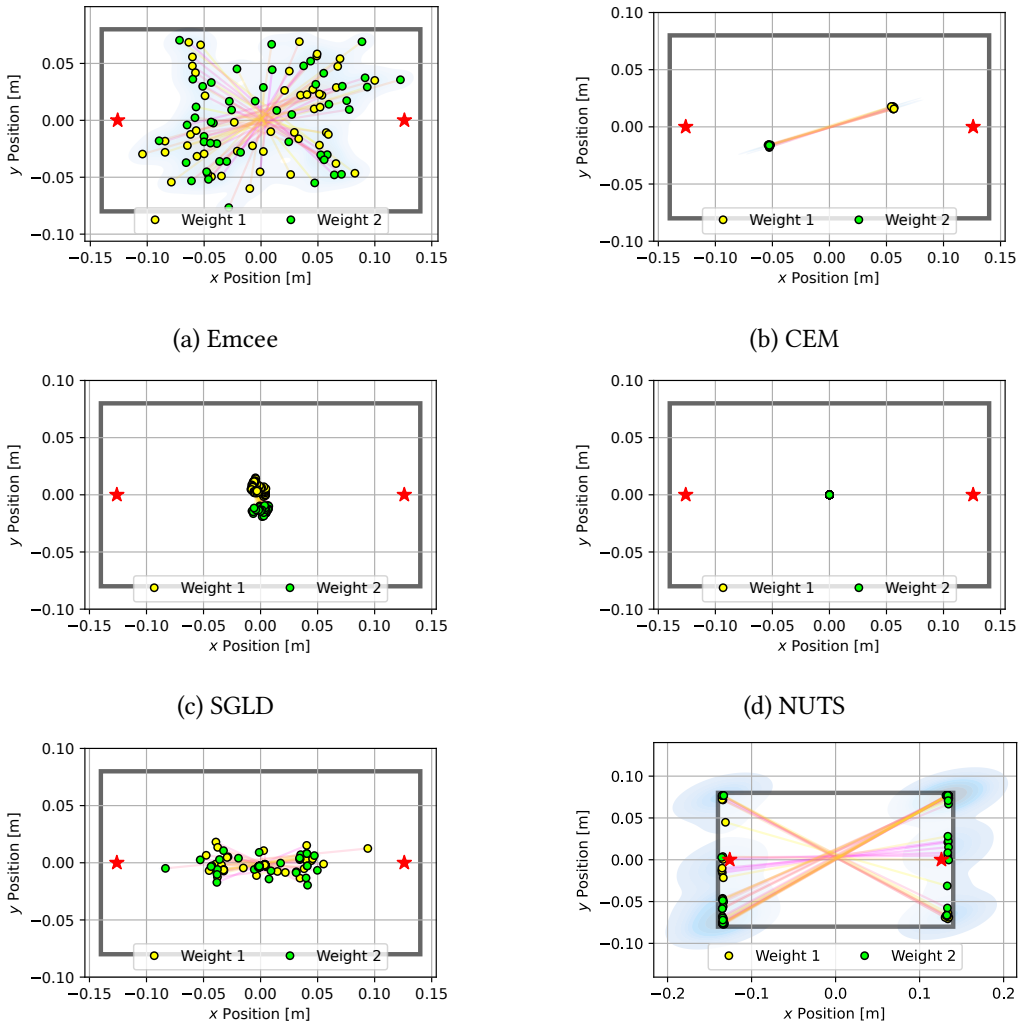


(a) Before identification of empty box



(b) After identification of empty box

Figure A.8: Trajectories from the Panda robot arm shaking an empty box. Visualized are the simulated (red) and real (black) observations before (a) and after (b) the inertial parameters of the empty box and the friction from the universal joint (Tab. A.4a) have been identified. The columns correspond to the four reference points in the frame of the box (see a rendering of them in Fig. A.7), the rows show the x, y, and z axes of these reference points in meters. The horizontal axes show the time step.



(a) Emcee

(b) CEM

(c) SGLD

(d) NUTS

(e) SVGD

(f) CSVGD

Figure A.9: Posterior plots over the 2D weight locations approximated by the estimation algorithms applied to the underactuated mechanism experiment from Section 4.1.5.3. Blue shades indicate a Gaussian kernel density estimation computed over the inferred parameter samples. Since it is an unbounded kernel density estimate, the blue shades cross the parameter boundaries in certain areas (e.g. for CSVGD), while in reality none of the estimated particles violate the parameter limits.

Appendix B

DiSECt: A Differentiable Simulator for Parameter Inference and Control in Robotic Cutting

B.1 Additional Algorithmic Details

B.1.1 Adam Optimizer

At its core, Adam updates the parameters θ interactively as $\theta_i \leftarrow \theta_{i-1} - \alpha \cdot \hat{m}_i/(\sqrt{\hat{v}_i} + \epsilon)$, where α is the learning rate, $\hat{m}_i$ and $\hat{v}_i$ represent the first and second order decaying averages (or momentum) after correcting for biases, and ϵ is a small value to prevent numerical issues. The full algorithm is shown in

Algorithm 7 Frank-Wolfe method

Result: Barycentric coordinate u of point on edge (p_1, p_2) with lowest signed distance w.r.t. SDF $d : \mathbb{R}^3 \to \mathbb{R}$
$u \leftarrow 1/2$
for $i = 0 \dots \text{max\_iterations}$ **do**
    $\delta \leftarrow \dfrac{\partial}{\partial u} d\left((1-u)p_1 + up_2\right)$
    if $\delta < 0$ **then**
        $s \leftarrow 1$
    else
        $s \leftarrow 0$
    end if
    $\gamma \leftarrow \dfrac{2}{2+i}$
    $u \leftarrow u + \gamma(s - u)$
end for

Algorithm 8. In Algorithm 9, we describe Stochastic Gradient Langevin Dynamics (SGLD) with the Adam update rule. A detailed description of SGLD is given in Section 3.2.4.2.

Algorithm 8 SGD with Adam

Given: learning rate α, initial parameters θ_0, likelihood function $l(\cdot)$
$m_0 \leftarrow 0$
$v_0 \leftarrow 0$
for $i = 1 \ldots \text{max\_iterations}$ **do**
    $g_i \leftarrow \nabla_\theta l(\theta_{i-1})$
    $m_i \leftarrow \beta_1 \cdot m_{i-1} + (1 - \beta_1) \cdot g_i$
    $v_i \leftarrow \beta_2 \cdot v_{i-1} + (1 - \beta_2) \cdot g_i^2$
    $\hat{m}_i \leftarrow m_i / (1 - \beta_1^i)$
    $\hat{v}_i \leftarrow v_i / (1 - \beta_2^i)$
    $\theta_i \leftarrow \theta_{i-1} - \alpha \cdot \hat{m}_i / (\sqrt{\hat{v}_i} + \epsilon)$
end for
return θ_i

Algorithm 9 SGLD with Adam update rule

Given: learning rate α, initial parameters θ_0, likelihood function $l(\cdot)$
$m_0 \leftarrow 0$
$v_0 \leftarrow 0$
for $i = 1 \ldots \text{max\_iterations}$ **do**
    $g_i \leftarrow \nabla_\theta U(\theta_{i-1})$
    $m_i \leftarrow \beta_1 \cdot m_{i-1} + (1 - \beta_1) \cdot g_i$
    $v_i \leftarrow \beta_2 \cdot v_{i-1} + (1 - \beta_2) \cdot g_i^2$
    $\hat{m}_i \leftarrow m_i / (1 - \beta_1^i)$
    $\hat{v}_i \leftarrow v_i / (1 - \beta_2^i)$
    $A_i \leftarrow \text{diag}\left(\mathbf{1} \oslash (\sqrt{\hat{v}_i} + \epsilon)\right)$
    $\eta_i \sim \mathcal{N}(0, \alpha A_i)$
    $\theta_i \leftarrow \theta_{i-1} - \alpha \cdot \hat{m}_i \cdot A_i + \eta_i$
end for
return θ_i

B.1.2 Loss Functions

We investigated various loss functions for evaluating the closeness between knife force profiles effectively. Given the two-dimensional parameter inference experiment from Section 3.2.5.1, we test the following cost functions which we minimize via the Adam optimizer:

- $L1$ loss
- $L2$ loss
- Inverse Cosine Similarity
- LogSumExp

We visualize the trace of the optimized parameters in Fig. B.1. The $L1$ loss function compares favorably to the other formulations as it allows for a relatively fast convergence to the true parameters, while also yielding a "bouncing" behavior when the iterates are close to the (local) optimum. Since we use the $L1$ loss

function in the likelihood term for our probabilistic parameter inference experiments 3.2.5.1, the likelihood distribution corresponds to a Laplace distribution which is known to have sharp peaks and heavy tales.

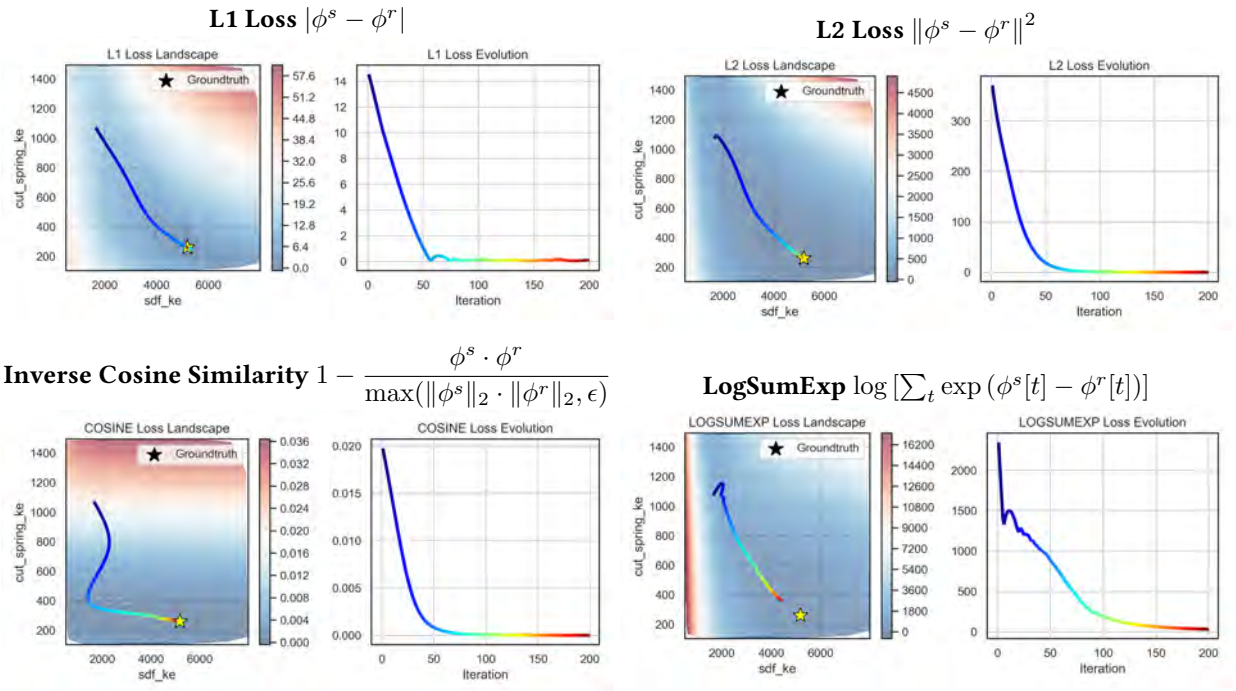


Figure B.1: Evaluation of different loss functions on the `actual_2d` scenario from Section 3.2.5.1. For 500 randomly sampled parameter vectors, the trajectories are simulated (shown as dots) and the loss is computed between the simulated trajectory ϕ^s and the ground-truth knife force profile ϕ^r. The interpolated loss landscape is shown one the left, and the training loss evolution using the Adam optimizer is shown on the right. Overall, the $L1$ loss performs favorably compared to the other cost functions and is therefore our choice to evaluate the closeness between knife force trajectories across this work.

B.1.3 BayesSim Implementation

BayesSim approximates the posterior over the simulation parameters as

$$p(\theta \mid \phi) \approx p(\theta)/\tilde{p}(\theta)\; q(\theta \mid \phi),$$

where $p(\theta)$ is the prior distribution and $\tilde{p}(\theta)$ is a proposal prior used to sample the simulator and collect N samples, $\{\theta_i, \phi_i^s\}_{i=1}^N$, to learn $q(\theta \mid \phi)$. When a real trajectory ϕ^r is observed, BayesSim computes

$p(\theta \mid \phi = \phi^r)$ that represents the posterior over simulation parameters given the real data. As inferring simulator parameters given trajectories is a type of inverse problem, it can admit a multitude of solutions.

For the BayesSim baseline, we train a mixture density network (MDN) representing $q(\theta \mid \phi)$ with 10 components on a dataset consisting of 500 knife force trajectories that have been generated in our simulator by uniformly sampling the simulation parameters within their respective bounds. Depending on the experiment, we limit ourselves to only a subset of all the available parameters, due to the exponential increase in sample complexity. As summary statistics input to BayesSim, we downsample the 0.9 s knife force profiles consisting of 90,000 time steps by a factor of 1000 to 90-dimensional summary statistics using polyphase filtering. While training the MDN, we project the true parameter values to the unit interval using their possible ranges (see Tab. B.2), as we found the MDN to be sensitive to the scale of the parameters, and to perform more accurately with homogeneous value ranges.

B.1.4 Optimal Transport of Cutting Spring Parameters

The Earth Mover's Distance can be interpreted as the minimal cost associated with transforming a constant volume pile of dirt into another, where the cost is defined as the amount of dirt moved multiplied by the distance travelled. Formally, given two sets of vertices P and Q, associated sets of vertex weights $\mathbf{w}_P$ and $\mathbf{w}_Q$, and a cost matrix given by the Euclidean distance between two points $D = |d_{i,j}| = \|p_i - q_j\|^2$, optimal transport finds the solution of the optimization problem

$$\min_F \frac{1}{Z} \sum_{i=1}^{m} \sum_{j=1}^{n} f_{i,j}\, d_{i,j}, \tag{B.1}$$

where $F = f_{i,j}$ is the movement or flow between p_i and q_j which we try to minimize, and $Z = \sum_{i=1}^{m} \sum_{j=1}^{n} f_{i,j}$ is a normalization constant. In our formulation, we assume uniform weights for both sets of vertices.

This problem can be efficiently solved using the network simplex algorithm [56] and has typical complexity $O(n^3)$, but sparsity can be exploited to reduce this cost.

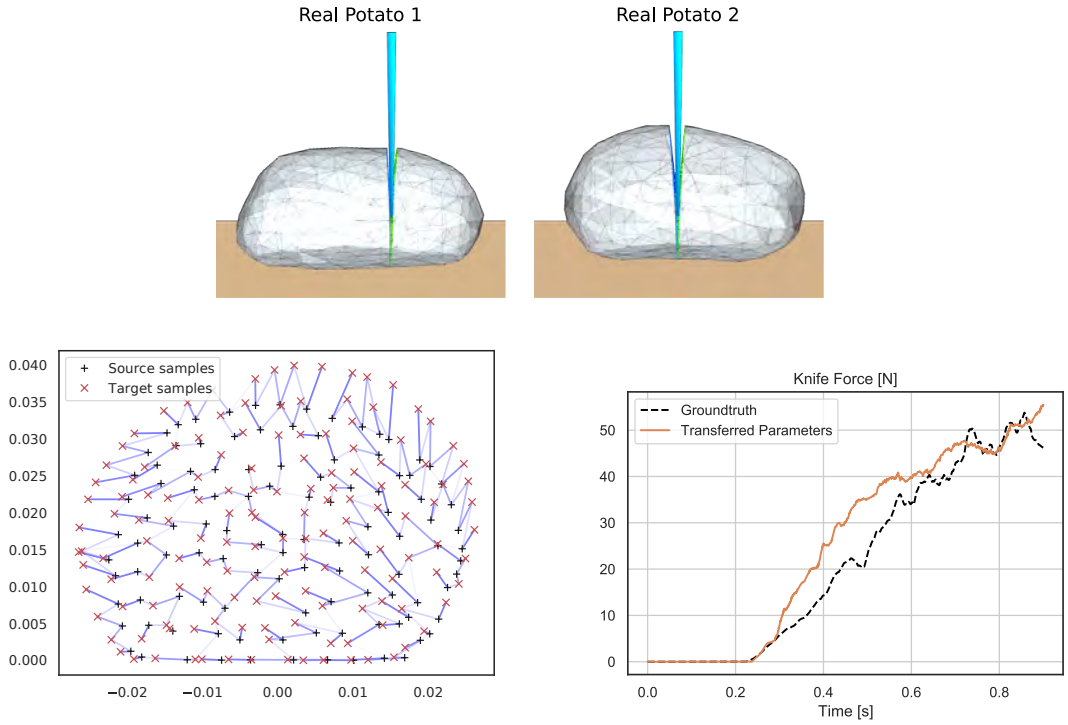


Figure B.2: Optimal transport of simulation parameters from one mesh of a real potato (top left) to another real potato mesh (top right). The correspondences (bottom left) have been found via optimal transport using the Earth Mover's Distance (Eq. B.1) between the cutting springs in two potato meshes. The weighted mapping between the 2D positions of the springs at the cutting interface from the source domain (`ybj_potato1`) to the target domain (`ybj_potato2`) is used to transfer the cutting spring parameters between the two topologically different meshes, resulting in a close fit to the ground-truth trajectory (bottom right).

Material	E (N m^{-2})	ν	ρ (kg m^{-3})
Apple	3.0×10^6	0.17	787
Potato	2.0×10^6	0.45	630
Cucumber	2.5×10^6	0.37	950
Banana	0.003048×10^6	0.28	1350

Table B.1: Properties of common biomaterials: Young's modulus E, Poisson's ratio ν, density ρ.

B.2 Details of Experimental Setup

B.2.1 Simulation Parameters

DiSECt introduces additional degrees of freedom via the insertion of virtual nodes and cutting springs that connect them (see Section 3.2.3.2). In Tab. B.2, we list each available parameter with its description and default value. Parameters that are allowed to be individually tuned for each spring can be set in two modes:

- Shared parameterization: the parameter is a single scalar that gets replicated across all cutting springs.
- Individual parameterization: the parameter is a vector where each entry can be tuned separately for each spring.

To enforce hard parameter limits in our simulator, throughout our experiments, we impose bounds on the estimated simulation parameters by projecting them through the sigmoid function. Thus, given an unconstrained real number x to be optimized, the resulting projected parameter value is $\mathrm{sigmoid}(x) \cdot (p_{ub} - p_{lb}) + p_{lb}$, where p_{lb} and p_{ub} are the upper and lower bounds of the parameter, and $\mathrm{sigmoid}(x) = 1/(1 + \exp(-x))$.

Name	Description	Default value	
Knife geometry parameters (fixed)			
`edge_dim`	Lower diameter of knife (see Fig. B.3 right)	0.08	mm
`spine_dim`	Upper diameter of knife (spine)	2	mm
`spine_height`	Height of knife spine	40	mm
`tip_height`	Height of knife tip	0.04	mm
`depth`	Length of knife blade (along z axis)	150	mm
Knife motion			
`velocity_y`	Vertical knife velocity	-0.05	$\mathrm{m\,s^{-1}}$
`initial_y`	Initial vertical knife position (height)	80	mm
Spring-damper parameters (individual for each spring)			
`cut_spring_ke`	Spring stiffness coefficient at the start of the simulation (initial stiffness of the cutting spring)	500	
`cut_spring_kd`	Spring damping coefficient	0.1	
`cut_spring_softness`	Softness coefficient γ used in the linear spring loosening (Eq. 3.7)	500	
Knife contact dynamics parameters (individual for each spring)			
`sdf_radius`	Radius around SDF to consider for contact dynamics	0.5	mm
`sdf_ke`	Positional penalty coefficient (contact normal stiffness)	1000	
`sdf_kd`	Damping coefficient	1	
`sdf_kf`	Contact friction stiffness (tangential stiffness used in Coulomb friction model)	0.01	
`sdf_mu`	Friction coefficient (μ)	0.5	
Ground contact dynamics parameters (fixed)			
`ground_ke`	Positional penalty coefficient (contact normal stiffness)	100	
`ground_kd`	Damping coefficient	0.1	
`ground_kf`	Contact friction stiffness (Coulomb friction model)	0.2	
`ground_mu`	Friction coefficient (μ)	0.6	
`ground_radius`	Radius around mesh vertices	1	mm
Material properties			
`young`	Young's modulus E	see Tab. B.1	
`poisson`	Poisson's ratio ν	see Tab. B.1	
`density`	Density ρ	see Tab. B.1	

Table B.2: Overview of the model parameters used by our cutting simulator.

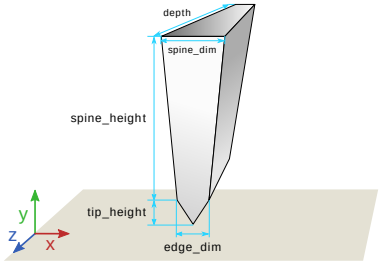


Figure B.3: Parameters that describe the knife geometry.

B.2.2 Commercial Simulation Setup

In the commercial solver, each simulation consisted of a rigid knife, 1 of 3 deformable fruits/vegetables (apple, cucumber, or potato), and a rigid table. The apple, cucumber, and potato geometries were represented by sphere, cylinder, and rectangular shape primitives, respectively. Each primitive had a 10 mm-thick slice, centered along the long axis, that defined the volumetric region that could be cut. All regions were assigned an isotropic elastic failure material model, with density, Lamé parameters (see Tab. B.1), yield stress, and failure strain obtained from the agricultural mechanics literature [188, 232, 69] and the FoodData Central database from the U.S. Department of Agriculture [323]. The non-slice regions were simulated using FEM with a tetrahedral mesh-based discretization. The slice region was simulated using the smoothed particle Galerkin (SPG) method with a particle-based discretization. SPG is a numerical method related to smoothed particle hydrodynamics (SPH) [194, 228] and the element-free Galerkin method (EFG) [21], and has been validated for simulating large deformation and failure of elastoplastic solids [340]. Continuity conditions were imposed between the non-slice mesh and the slice particles. Coulomb frictional contact was defined between the knife and the deformable object, as well as between the object and the table, with a coefficient of friction of 0.6. A constant downward velocity was applied to the knife until contact with the surface of the table, and gravity was applied to the deformable object.

B.3 Additional Experiments

B.3.1 Generalization Results

Fig. B.4 shows the trajectories for the generalization experiment (Sec. 3.2.5.4) where the parameters have been optimized for a shorter simulation time window of 0.4 s, and tested against the ground-truth trajectory (green, dashed line) over a duration of 0.9 s.

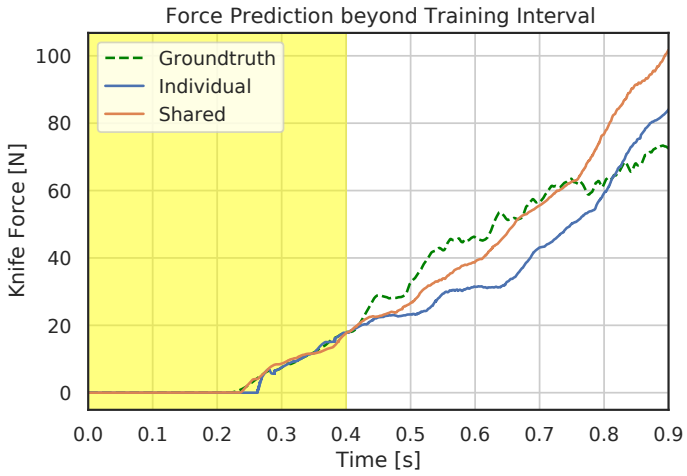


Figure B.4: Generalization performance on the apple cutting trajectory by simulating the parameters which have been optimized from a 0.4 s ground-truth trajectory (highlighted in yellow) from a commercial simulator. At test time, the force profile is simulated over a duration of 0.9 s (see Sec. 3.2.5.4). The blue line shows the estimated trajectory when all parameters were optimized individually for each cutting spring. Shown in orange is the resulting trajectory from optimizing parameters shared across all cutting springs.

Analogous to the apple cutting results with ground-truth from the commercial solver in Fig. 3.20, the bar plot in Fig. B.5 visualizes the normalized mean absolute error (NMAE) for different knife velocities at test time. The simulation parameters have been optimized for a vertical downward velocity of 50 mm (green shade in the background), given a knife force profile of cutting a cylindrical mesh with cucumber material properties from the commercial simulator. At test time, we use the same optimized simulation parameters to evaluate the accuracy of the force profile simulation against the commercial simulator on different downward velocities. As in the experiments of cutting an apple (Fig. 3.20), the results in Fig. B.5 show that the individual cutting spring parameterization outperforms the shared parameterization across all tested velocities. While the performance improvement over the shared parameterization becomes less significant, it can be concluded that the individual parameterization generalizes better to novel test velocities.

B.3.2 Posterior Over Simulation Parameters

The marginal plots in Fig. B.6 and Fig. B.7 show the posteriors from iterative BayesSim and SGLD, respectively. These densities over a subset of the simulation parameters (shared across the cutting springs)

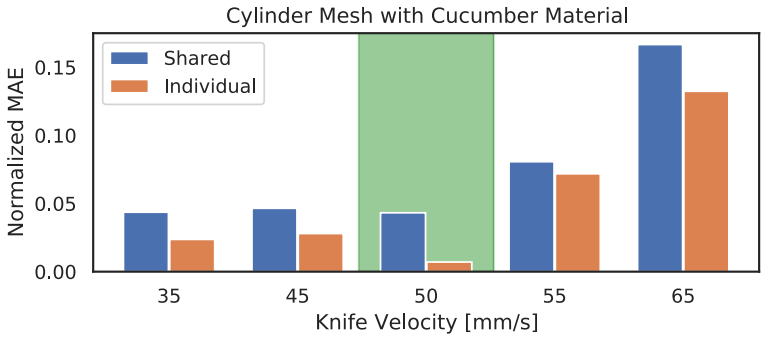


Figure B.5: Velocity generalization results for a cylinder mesh with cucumber material properties given ground-truth simulations with different vertical knife velocities from a commercial solver. Two versions of DiSECt were calibrated: by sharing the parameters across all cutting springs (blue) and by optimizing each value individually (orange), given a ground-truth trajectory with the knife sliding down at 50 mm s^{-1} speed (highlighted in green). The normalized mean absolute error (MAE) is evaluated against the ground-truth by rolling out the estimated parameters for the given knife velocity.

have been inferred from knife force profiles of cutting a sphere mesh using apple material properties. The ground-truth, from which the posteriors are inferred, stems from a high-fidelity commercial simulator (see description in Section 3.2.5.2).

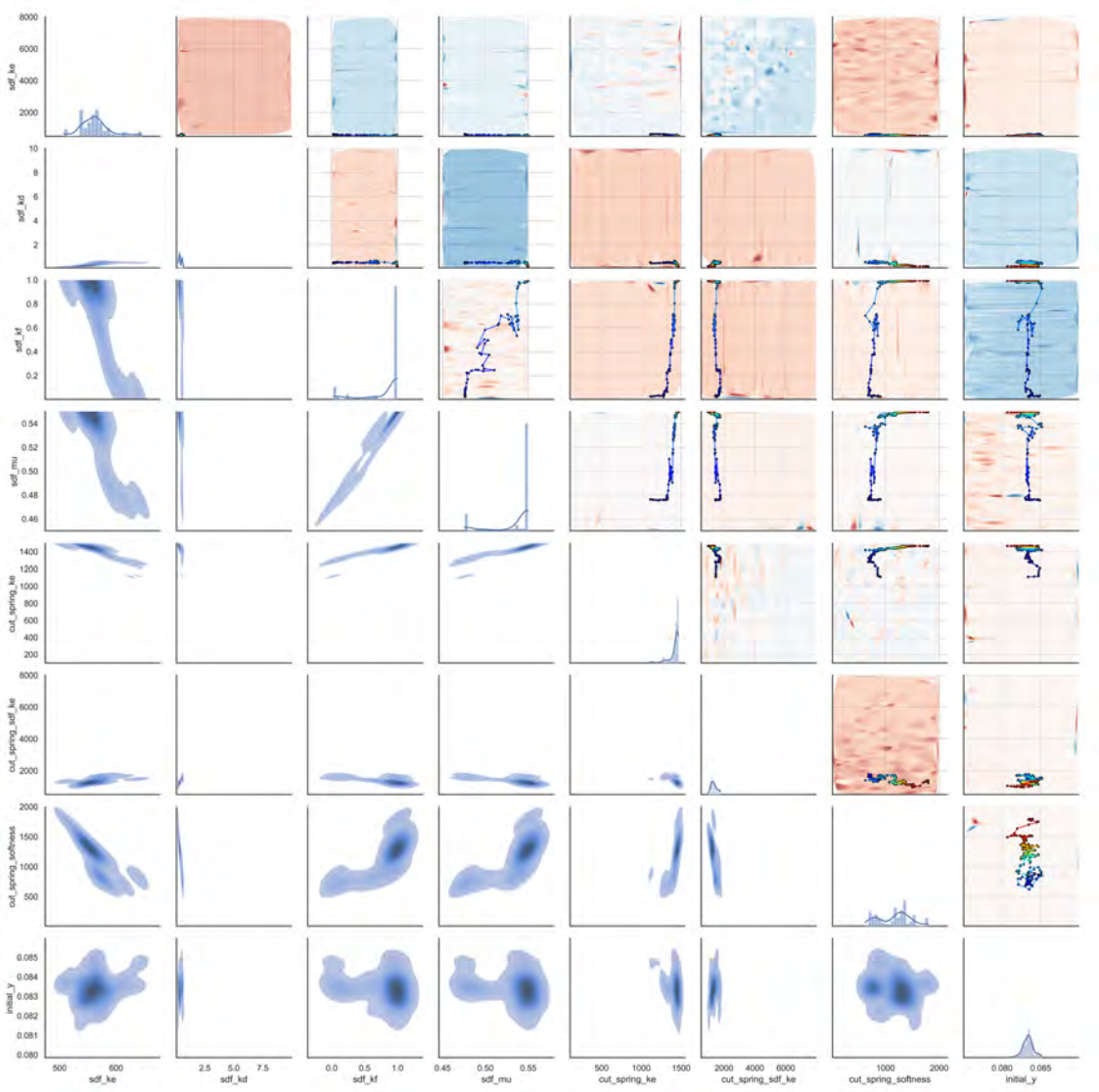


Figure B.6: Posterior obtained by SGLD in our differentiable simulator after 300 trajectory roll-outs. Marginals are shown on the diagonal. The sampled chain is visualized for pairs of parameter dimensions in the upper triangle, along with an approximate rendering of the loss surface as heatmap in the background. The heatmaps in the lower triangle visualize the kernel densities for all pair-wise combinations of the parameters.

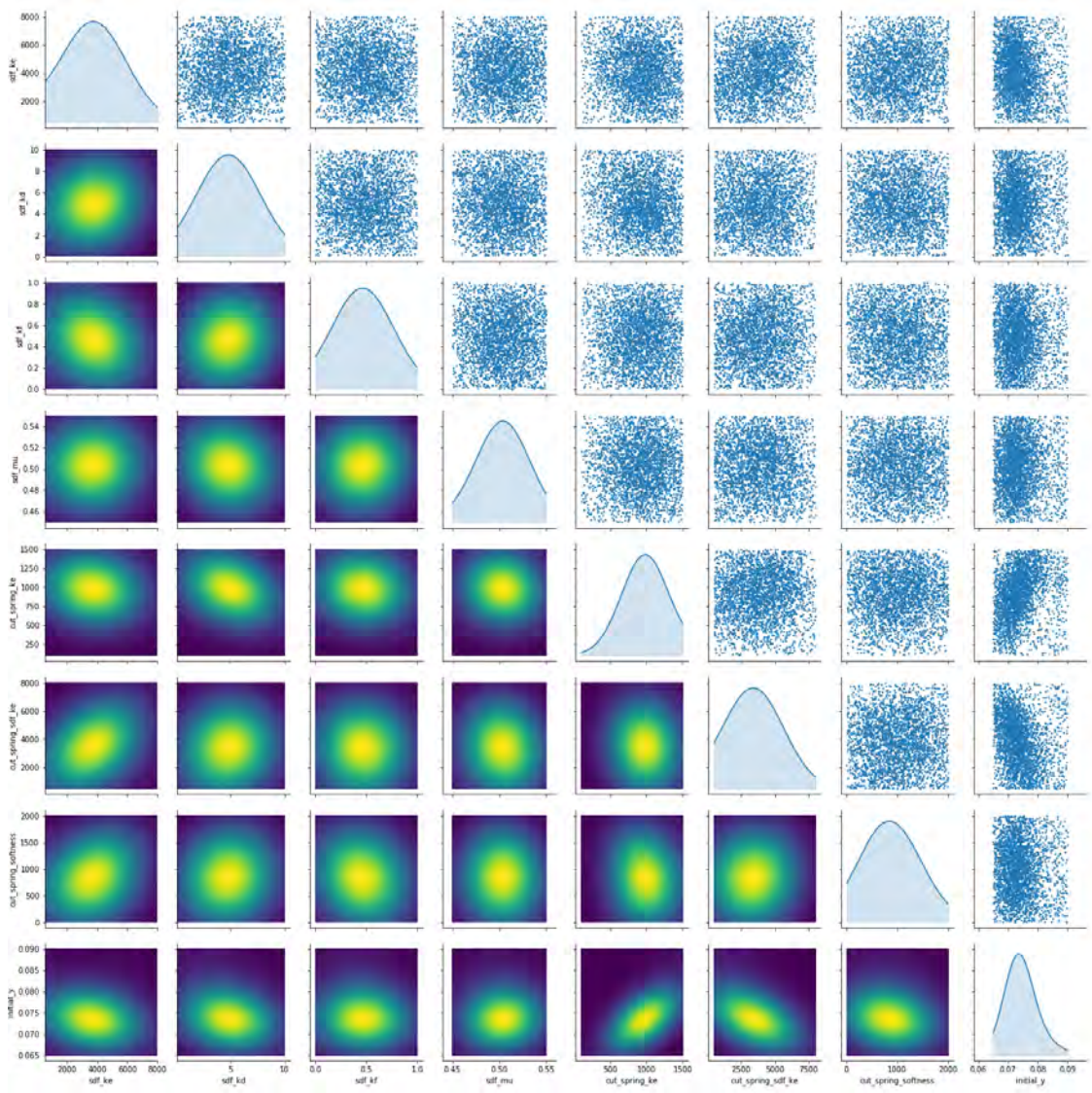


Figure B.7: Posterior obtained by BayesSim after 100 iterative updates of the training dataset with 20 new trajectories per iteration. Marginals are shown on the diagonal. Parameter samples are represented by the dots in the scatter plots from the upper triangle. The heatmaps in the lower triangle visualize the kernel densities for all pair-wise combinations of the parameters.

Bibliography

[1] Sameer Agarwal, Keir Mierle, et al. *Ceres Solver*. http://ceres-solver.org.

[2] Ali-akbar Agha-mohammadi, Eric Heiden, Karol Hausman, and Gaurav Sukhatme. "Confidence-rich grid mapping". In: *The International Journal of Robotics Research* (2019). DOI: 10.1177/0278364919839762. eprint: https://doi.org/10.1177/0278364919839762.

[3] Anurag Ajay, Jiajun Wu, Nima Fazeli, Maria Bauza, Leslie P Kaelbling, Joshua B Tenenbaum, and Alberto Rodriguez. "Augmenting Physical Simulators with Stochastic Neural Networks: Case Study of Planar Pushing and Bouncing". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018.

[4] Brandon Amos and J. Zico Kolter. "OptNet: Differentiable Optimization as a Layer in Neural Networks". In: *Proceedings of the 34th International Conference on Machine Learning*. Vol. 70. Proceedings of Machine Learning Research. PMLR, 2017, pp. 136–145.

[5] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. "Hindsight experience replay". In: *Advances in Neural Information Processing Systems*. 2017, pp. 5048–5058.

[6] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Józefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. "Learning dexterous in-hand manipulation". In: *The International Journal of Robotics Research* 39.1 (2020). Publisher: SAGE Publications Ltd STM, pp. 3–20. ISSN: 0278-3649. DOI: 10.1177/0278364919887447. (Visited on 06/13/2021).

[7] G. Antonelli, F. Caccavale, and P. Chiacchio. "A systematic procedure for the identification of dynamic parameters of robot manipulators". en. In: *Robotica* 17.4 (1999). Publisher: Cambridge University Press, pp. 427–435. ISSN: 1469-8668, 0263-5747. DOI: 10.1017/S026357479900140X. (Visited on 08/20/2021).

[8] P Areias and Timon Rabczuk. "Steiner-point free edge cutting of tetrahedral meshes with applications in fracture". In: *Finite Elements in Analysis and Design* 132 (2017), pp. 27–41.

[9] Alexis Asseman, Tomasz Kornuta, and Ahmet Ozcan. “Learning beyond simulated physics”. In: *Neural Information Processing Systems. Modeling and Decision-making in the Spatiotemporal Domain Workshop*. 2018. URL: https://openreview.net/forum?id=HylajWsRF7.

[10] Christopher G. Atkeson, Chae H. An, and John M. Hollerbach. “Estimation of Inertial Parameters of Manipulator Loads and Links”. In: *The International Journal of Robotics Research* 5.3 (1986), pp. 101–119. DOI: 10.1177/027836498600500306. eprint: https://doi.org/10.1177/027836498600500306.

[11] Christopher G. Atkeson, Andrew W. Moore, and Stefan Schaal. “Locally Weighted Learning for Control”. In: *Artificial Intelligence Review* 11.1 (1997), pp. 75–113. ISSN: 1573-7462. DOI: 10.1023/A:1006511328852.

[12] A.G. Atkins and T. Atkins. *The Science and Engineering of Cutting: The Mechanics and Processes of Separating, Scratching and Puncturing Biomaterials, Metals and Non-metals*. Elsevier Science, 2009.

[13] AG Atkins, X Xu, and G Jeronimidis. “Cutting by ‘pressing and slicing’ of thin floppy slices of materials illustrated by experiments on cheddar cheese and salami”. In: *Journal of Materials Science* 39 (2004), pp. 2761–2766.

[14] Filipe de Avila Belbute-Peres, Kevin Smith, Kelsey Allen, Josh Tenenbaum, and J. Zico Kolter. “End-to-End Differentiable Physics for Learning and Control”. In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Curran Associates, Inc., 2018, pp. 7178–7189. URL: http://papers.nips.cc/paper/7948-end-to-end-differentiable-physics-for-learning-and-control.pdf.

[15] Ozgur Aydogmus and Ali Hakan TOR. “A Modified Multiple Shooting Algorithm for Parameter Estimation in ODEs Using Adjoint Sensitivity Analysis”. In: *Applied Mathematics and Computation* 390 (2021), p. 125644. ISSN: 0096-3003. DOI: https://doi.org/10.1016/j.amc.2020.125644.

[16] Samuel Barrett, Matthew E. Taylor, and Peter Stone. “Transfer Learning for Reinforcement Learning on a Physical Robot”. In: *Ninth International Conference on Autonomous Agents and Multiagent Systems - Adaptive Learning Agents Workshop (AAMAS - ALA)*. 2010. URL: http://www.cs.utexas.edu/users/ai-lab/?AAMASWS10-barrett.

[17] J. -F. M. Barthelemy and L. E. Hall. “Automatic differentiation as a tool in engineering design”. In: *Structural optimization* 9.2 (1995), pp. 76–82. ISSN: 1615-1488. DOI: 10.1007/BF01758823.

[18] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. “Interaction networks for learning about objects, relations and physics”. In: *Advances in Neural Information Processing Systems*. 2016, pp. 4502–4510.

[19] Joachim Baumgarte. “Stabilization of constraints and integrals of motion in dynamical systems”. In: *Computer methods in applied mechanics and engineering* 1.1 (1972), pp. 1–16.

[20] Bradley Bell. *CppAD: a package for C++ algorithmic differentiation*. http://www.coin-or.org/CppAD. 2020.

[21] T Belytschko, YY Lu, and L Gu. “Element-free Galerkin methods”. In: *International Journal for Numerical Methods in Engineering* 37 (1994), pp. 229–256.

[22] Ted Belytschko, Wing Kam Liu, Brian Moran, and Khalil Elkhodary. *Nonlinear finite elements for continua and structures*. John Wiley & Sons, 2013.

[23] I. Berndt, R. Torchelsen, and A. Maciel. “Efficient Surgical Cutting with Position-Based Dynamics”. In: *IEEE Computer Graphics and Applications* 37.3 (2017), pp. 24–31.

[24] P. J. Besl and N. D. McKay. “A method for registration of 3-D shapes”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14.2 (1992), pp. 239–256. DOI: 10.1109/34.121791.

[25] P. Biber and W. Strasser. “The normal distributions transform: a new approach to laser scan matching”. In: *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*. Vol. 3. 2003, pp. 2743–2748. DOI: 10.1109/IROS.2003.1249285.

[26] Daniel Bielser, Pascal Glardon, Matthias Teschner, and Markus Gross. “A state machine for real-time cutting of tetrahedral meshes”. In: *11th Pacific Conference on Computer Graphics and Applications*. 2003, pp. 377–386.

[27] Francesco Biscani and Dario Izzo. “A parallel global multiobjective framework for optimization: pagmo”. In: *Journal of Open Source Software* 5.53 (2020), p. 2338. DOI: 10.21105/joss.02338.

[28] Christopher M Bishop. *Mixture density networks*. Tech. rep. 1994.

[29] David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. “Variational Inference: A Review for Statisticians”. In: *Journal of the American Statistical Association* 112.518 (2017). Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/01621459.2017.1285773, pp. 859–877. ISSN: 0162-1459. DOI: 10.1080/01621459.2017.1285773. (Visited on 06/17/2021).

[30] H. G. Bock. “Recent Advances in Parameteridentification Techniques for O.D.E.” en. In: *Numerical Treatment of Inverse Problems in Differential and Integral Equations: Proceedings of an International Workshop, Heidelberg, Fed. Rep. of Germany, August 30 — September 3, 1982*. Ed. by Peter Deuflhard and Ernst Hairer. Progress in Scientific Computing. Boston, MA: Birkhäuser, 1983, pp. 95–121. ISBN: 978-1-4684-7324-7. DOI: 10.1007/978-1-4684-7324-7_7. (Visited on 06/06/2021).

[31] 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*. 2014, pp. 1–8. DOI: 10.1109/ADPRL.2014.7010608.

[32] R Breda. “Experimental measurement of parameters of the spatial scanner Hokuyo URG-04LX”. In: *Przeglad Elektrotechniczny* 88.5b (2012), pp. 132–135.

[33] Peter Brown. “Contact modelling for forward dynamics of human motion”. MA thesis. University of Waterloo, 2017.

[34] Peter Brown and John McPhee. “A 3D ellipsoidal volumetric foot–ground contact model for forward dynamics”. In: *Multibody System Dynamics* 42.4 (2018), pp. 447–467.

[35] John Burkardt. *SOBOL - The Sobol Quasirandom Sequence*. https://people.sc.fsu.edu/~jburkardt/cpp_src/sobol/sobol.html. Accessed: 2021-06-23.

[36] Daniel Burkhart, Bernd Hamann, and Georg Umlauf. "Adaptive and feature-preserving subdivision for high-quality tetrahedral meshes". In: *Computer Graphics Forum* 29.1 (2010), pp. 117–127.

[37] Fabrizio Caccavale and Pasquale Chiacchio. "Identification of Dynamic Parameters for a Conventional Industrial Manipulator". en. In: *IFAC Proceedings Volumes*. IFAC Symposium on System Identification (SYSID'94), Copenhagen, Denmark, 4-6 July 27.8 (1994), pp. 871–876. ISSN: 1474-6670. DOI: 10.1016/S1474-6670(17)47819-0. (Visited on 08/20/2021).

[38] Cang Ye and J. Borenstein. "Characterization of a 2D laser scanner for mobile robot obstacle negotiation". In: *IEEE International Conference on Robotics and Automation*. Vol. 3. 2002, 2512–2518 vol.3. DOI: 10.1109/ROBOT.2002.1013609.

[39] Bob Carpenter, Matthew D. Hoffman, Marcus Brubaker, Daniel Lee, Peter Li, and Michael Betancourt. "The Stan Math Library: Reverse-Mode Automatic Differentiation in C++". In: *CoRR* abs/1509.07164 (2015). arXiv: 1509.07164. URL: http://arxiv.org/abs/1509.07164.

[40] Justin Carpentier and Nicolas Mansard. "Analytical Derivatives of Rigid Body Dynamics Algorithms". In: *Robotics: Science and Systems*. 2018.

[41] Michael B Chang, Tomer Ullman, Antonio Torralba, and Joshua B Tenenbaum. "A Compositional Object-Based Approach to Learning Physical Dynamics". In: *International Conference on Learning Representations* (2017).

[42] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox. "Closing the Sim-to-Real Loop: Adapting Simulation Randomization with Real World Experience". In: *2019 International Conference on Robotics and Automation (ICRA)*. ISSN: 2577-087X. 2019, pp. 8973–8979. DOI: 10.1109/ICRA.2019.8793789.

[43] Y. Chebotar, K. Hausman, Z. Su, G.S. Sukhatme, and S. Schaal. "Self-Supervised Regrasping using Spatio-Temporal Tactile Features and Reinforcement Learning". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Daejeon, South Korea, 2016.

[44] Yevgen Chebotar, Karol Hausman, Marvin Zhang, Gaurav Sukhatme, Stefan Schaal, and Sergey Levine. "Combining model-based and model-free updates for trajectory-centric reinforcement learning". In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 703–711.

[45] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. "Neural Ordinary Differential Equations". In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Curran Associates, Inc., 2018, pp. 6571–6583. URL: http://papers.nips.cc/paper/7892-neural-ordinary-differential-equations.pdf.

[46] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. "Neural ordinary differential equations". In: *Advances in Neural Information Processing Systems*. 2018, pp. 6571–6583.

[47] Rémi Chou, Yvo Boers, Martin Podt, and Matthieu Geist. “Performance evaluation for particle filters”. In: *14th International Conference on Information Fusion*. 2011, pp. 1–7.

[48] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. “Deep reinforcement learning in a handful of trials using probabilistic dynamics models”. In: *Advances in Neural Information Processing Systems*. 2018, pp. 4754–4765.

[49] Ignasi Clavera, Anusha Nagabandi, Simin Liu, Ronald S. Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. “Learning to Adapt in Dynamic, Real-World Environments through Meta-Reinforcement Learning”. In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=HyztsoC5Y7.

[50] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)*. 2016. arXiv: 1511.07289 [cs.LG].

[51] T. Coleman David. “Reducing the Barrier to Entry of Complex Robotic Software: a MoveIt! Case Study”. en. In: (2014). Publisher: Università degli studi di Bergamo. DOI: 10.6092/JOSER_2014_05_01_P3. (Visited on 06/25/2021).

[52] Peter A Cooper. “Paradigm shifts in designed instruction: From behaviorism to cognitivism to constructivism”. In: *Educational technology* 33.5 (1993), pp. 12–19.

[53] George Corliss, Christèle Faure, Andreas Griewank, Lauren Hascoët, and Uwe Naumann, eds. *Automatic Differentiation of Algorithms: From Simulation to Optimization*. New York, NY, USA: Springer-Verlag New York, Inc., 2002. ISBN: 0-387-95305-1.

[54] Erwin Coumans and Yunfei Bai. “PyBullet, a Python module for physics simulation for games, robotics and machine learning”. In: (2020). URL: http://pybullet.org.

[55] Kyle Cranmer, Johann Brehmer, and Gilles Louppe. “The frontier of simulation-based inference”. In: *Proceedings of the National Academy of Sciences* 117.48 (2020), pp. 30055–30062. ISSN: 0027-8424. DOI: 10.1073/pnas.1912789117. eprint: https://www.pnas.org/content/117/48/30055.full.pdf.

[56] W. H. Cunningham. “A network simplex method”. In: *Mathematical Programming* 11 (1976), pp. 105–116.

[57] Debao Zhou, M. R. Claffee, Kok-Meng Lee, and G. V. McMurray. “Cutting, ’by pressing and slicing’, applied to the robotic cut of bio-materials. II. Force during slicing and pressing cuts”. In: *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* 2006, pp. 2256–2261.

[58] Debao Zhou, M. R. Claffee, Kok-Meng Lee, and G. V. McMurray. “Cutting, "by pressing and slicing", applied to robotic cutting bio-materials. I. Modeling of stress distribution”. In: *Proceedings 2006 IEEE International Conference on Robotics and Automation*. 2006, pp. 2896–2901.

[59] Jonas Degrave, Michiel Hermans, Joni Dambre, and Francis Wyffels. “A Differentiable Physics Engine for Deep Learning in Robotics”. In: *Frontiers in Neurorobotics* 13 (2019), p. 6. ISSN: 1662-5218. DOI: 10.3389/fnbot.2019.00006.

[60] Marc Deisenroth and Carl E Rasmussen. “PILCO: A model-based and data-efficient approach to policy search”. In: *International Conference on machine learning*. 2011, pp. 465–472.

[61] Stefan Depeweg, José Miguel Hernández-Lobato, Finale Doshi-Velez, and Steffen Udluft. “Learning and Policy Search in Stochastic Dynamical Systems with Bayesian Neural Networks”. In: *International Conference on Learning Representations* (2017).

[62] Mingyu Ding, Zhenfang Chen, Tao Du, Ping Luo, Joshua B Tenenbaum, and Chuang Gan. “Dynamic Visual Reasoning by Learning Differentiable Physics Models from Video and Language”. In: *Advances In Neural Information Processing Systems*. 2021.

[63] Gary L Drescher. *Made-up minds: a constructivist approach to artificial intelligence*. MIT press, 1991.

[64] Yuqing Du, Olivia Watkins, Trevor Darrell, Pieter Abbeel, and Deepak Pathak. “Auto-Tuned Sim-to-Real Transfer”. In: *arXiv:2104.07662 [cs]* (2021). arXiv: 2104.07662. URL: http://arxiv.org/abs/2104.07662 (visited on 06/13/2021).

[65] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. “Benchmarking deep reinforcement learning for continuous control”. In: *International Conference on Machine Learning*. 2016, pp. 1329–1338.

[66] Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and Pieter Abbeel. “RL^2: Fast Reinforcement Learning via Slow Reinforcement Learning”. In: *CoRR* abs/1611.02779 (2016). arXiv: 1611.02779. URL: http://arxiv.org/abs/1611.02779.

[67] Phillip J Durst, Christopher Goodin, Burhman Q Gates, Christopher L Cummins, Burney McKinley, Jody D Priddy, Peter Rander, and Brett Browning. “The need for high-fidelity robotics sensor models”. In: *Journal of Robotics* 2011 (2011).

[68] Kamak Ebadi, Yun Chang, Matteo Palieri, Alex Stephens, Alex Hatteland, Eric Heiden, Abhishek Thakur, Nobuhiro Funabiki, Benjamin Morrell, Sally Wood, Luca Carlone, and Ali-akbar Agha-mohammadi. “LAMP: Large-Scale Autonomous Mapping and Positioning for Exploration of Perceptually-Degraded Subterranean Environments”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2020, pp. 80–86. DOI: 10.1109/ICRA40945.2020.9197082.

[69] IY El Said, Mervat M Atallah, KS Khalil, and AM El-Lithy. “Physical and mechanical properties of cucumber applied to seed extractor”. In: *Journal of Soil Sciences and Agricultural Engineering* 2.8 (2011), pp. 871–880.

[70] Alberto Elfes. “Occupancy grids: A probabilistic framework for robot perception and navigation”. PhD thesis. Carnegie Mellon University, 1989.

[71] Clemens Eppner, Roberto Martín-Martín, and Oliver Brock. “Physics-Based Selection of Informative Actions for Interactive Perception”. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 2018.

[72] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. "Diversity is All You Need: Learning Skills without a Reward Function". In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=SJx63jRqFm.

[73] Alon Farchy, Samuel Barrett, Patrick MacAlpine, and Peter Stone. "Humanoid Robots Learning to Walk Faster: From the Real World to Simulation and Back". In: *International Conference on Autonomous Agents and Multiagent Systems*. 2013. URL: http://www.cs.utexas.edu/users/ai-lab/?AAMAS13-Farchy.

[74] Nima Fazeli, Russ Tedrake, and Alberto Rodriguez. "Identifiability analysis of planar rigid-body frictional contact". In: *Robotics Research*. Springer, 2018, pp. 665–682.

[75] Roy Featherstone. "An Empirical Study of the Joint Space Inertia Matrix". In: *The International Journal of Robotics Research* 23.9 (2004), pp. 859–871. DOI: 10.1177/0278364904044869. eprint: https://doi.org/10.1177/0278364904044869.

[76] Roy Featherstone. *Rigid Body Dynamics Algorithms*. Berlin, Heidelberg: Springer-Verlag, 2007. ISBN: 0387743146.

[77] Donald P. Feder. "Differentiation of Ray-Tracing Equations with Respect to Construction Parameters of Rotationally Symmetric Optics". In: *J. Opt. Soc. Am.* 58.11 (1968), pp. 1494–1505. DOI: 10.1364/JOSA.58.001494.

[78] Jean Feng and Noah Simon. *Sparse-Input Neural Networks for High-dimensional Nonparametric Regression and Classification*. 2019. arXiv: 1711.07592 [stat.ME].

[79] Chelsea Finn, Pieter Abbeel, and Sergey Levine. "Model-agnostic meta-learning for fast adaptation of deep networks". In: *Proceedings of the 34th International Conference on Machine Learning*. Vol. 70. JMLR. org. 2017, pp. 1126–1135.

[80] Chelsea Finn, Ian Goodfellow, and Sergey Levine. "Unsupervised learning for physical interaction through video prediction". In: *Advances in Neural Information Processing Systems*. 2016, pp. 64–72.

[81] Martin A. Fischler and Robert C. Bolles. "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography". In: *Communications of the ACM* 24.6 (1981), pp. 381–395. ISSN: 0001-0782. DOI: 10.1145/358669.358692.

[82] R'emi Flamary and Nicolas Courty. *POT Python Optimal Transport library*. 2017. URL: https://pythonot.github.io/.

[83] Daniel Foreman-Mackey, David W. Hogg, Dustin Lang, and Jonathan Goodman. "emcee: The MCMC Hammer". In: *Publications of the Astronomical Society of the Pacific* 125.925 (2013), pp. 306–312. ISSN: 1538-3873. DOI: 10.1086/670067.

[84] Bennett L. Fox. "Algorithm 647: Implementation and Relative Efficiency of Quasirandom Sequence Generators". In: *ACM Transactions on Mathematical Software* 12.4 (1986), pp. 362–376. ISSN: 0098-3500. DOI: 10.1145/22721.356187.

[85] Marco Frigerio, Jonas Buchli, Darwin G. Caldwell, and Claudio Semini. "RobCoGen: a code generator for efficient kinematics and dynamics of articulated robots, based on Domain Specific Languages". In: *Journal of Software Engineering for Robotics* 7.1 (Special Issue on Domain-Specific Languages and Models for Robotic Systems 2016), pp. 36–54.

[86] Justin Fu, Sergey Levine, and Pieter 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.

[87] Yarin Gal, Rowan McAllister, and Carl Edward Rasmussen. "Improving PILCO with Bayesian neural network dynamics models". In: *Data-Efficient Machine Learning workshop, International Conference on Machine Learning*. 2016.

[88] D. Garreau, W. Jitkrittum, and M. Kanagawa. *Large sample analysis of the median heuristic*. 2018.

[89] M. Gautier and W. Khalil. "Direct calculation of minimum set of inertial parameters of serial robots". In: *IEEE Transactions on Robotics and Automation* 6.3 (1990). Conference Name: IEEE Transactions on Robotics and Automation, pp. 368–373. ISSN: 2374-958X. DOI: 10.1109/70.56655.

[90] M. Gautier and W. Khalil. "Exciting trajectories for the identification of base inertial parameters of robots". In: *[1991] Proceedings of the 30th IEEE Conference on Decision and Control*. 1991, 494–499 vol.1. DOI: 10.1109/CDC.1991.261353.

[91] Moritz Geilinger, David Hahn, Jonas Zehnder, Moritz Bächer, Bernhard Thomaszewski, and Stelian Coros. "ADD: Analytically Differentiable Dynamics for Multi-Body Systems with Frictional Contact". In: *ACM Trans. Graph.* 39.6 (2020). ISSN: 0730-0301. DOI: 10.1145/3414685.3417766.

[92] Markus Giftthaler, Michael Neunert, Markus Stäuble, Marco Frigerio, Claudio Semini, and Jonas Buchli. "Automatic differentiation of rigid body dynamics for optimal control and estimation". In: *Advanced Robotics* 31.22 (2017), pp. 1225–1237.

[93] R. A. Gingold and J. J. Monaghan. "Smoothed particle hydrodynamics: theory and application to non-spherical stars". In: 181.3 (1977), pp. 375–389.

[94] Florian Golemo, Adrien Ali Taiga, Aaron Courville, and Pierre-Yves Oudeyer. "Sim-to-Real Transfer with Neural-Augmented Robot Simulation". In: *Proceedings of The 2nd Conference on Robot Learning*. Ed. by Aude Billard, Anca Dragan, Jan Peters, and Jun Morimoto. Vol. 87. Proceedings of Machine Learning Research. PMLR, 2018, pp. 817–828. URL: http://proceedings.mlr.press/v87/golemo18a.html.

[95] Jonathan Goodman and Jonathan Weare. "Ensemble samplers with affine invariance". In: *Communications in Applied Mathematics and Computational Science* 5.1 (2010). Publisher: Mathematical Sciences Publishers, pp. 65–80. ISSN: 2157-5452. DOI: 10.2140/camcos.2010.5.65. (Visited on 06/09/2021).

[96] Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander Smola. "A Kernel Two-Sample Test". In: *Journal of Machine Learning Research* 13 (2012), pp. 723–773. ISSN: 1532-4435.

[97] Samuel Greydanus, Misko Dzamba, and Jason Yosinski. "Hamiltonian Neural Networks". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 15379–15389. URL: http://papers.nips.cc/paper/9672-hamiltonian-neural-networks.pdf.

[98] Samuel Greydanus, Misko Dzamba, and Jason Yosinski. "Hamiltonian Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper/2019/file/26cd8ecadce0d4efd6cc8a8725cbd1f8-Paper.pdf.

[99] Andreas Griewank and Andrea Walther. "Introduction to Automatic Differentiation". In: *PAMM* 2.1 (2003), pp. 45–49.

[100] Alan Arnold Griffith. "The phenomena of rupture and flow in solids". In: *Philosophical transactions of the royal society of London. Series A, containing papers of a mathematical or physical character* 221.582-593 (1921), pp. 163–198.

[101] M. Grotjahn, M. Daemi, and B. Heimann. "Friction and rigid body identification of robot dynamics". In: *International Journal of Solids and Structures* 38.10 (2001), pp. 1889–1902. ISSN: 0020-7683. DOI: https://doi.org/10.1016/S0020-7683(00)00141-4.

[102] Michael Gschwandtner, Roland Kwitt, Andreas Uhl, and Wolfgang Pree. "BlenSor: Blender Sensor Simulation Toolbox". In: *Advances in Visual Computing*. Ed. by George Bebis, Richard Boyle, Bahram Parvin, Darko Koracin, Song Wang, Kim Kyungnam, Bedrich Benes, Kenneth Moreland, Christoph Borst, Stephen DiVerdi, Chiang Yi-Jen, and Jiang Ming. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 199–208. ISBN: 978-3-642-24031-7.

[103] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates". In: *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE. 2017, pp. 3389–3396.

[104] Vincent Le Guen and Nicolas Thome. "Disentangling Physical Dynamics from Unknown Factors for Unsupervised Video Prediction". In: *CoRR* abs/2003.01460 (2020). arXiv: 2003.01460. URL: https://arxiv.org/abs/2003.01460.

[105] Gaël Guennebaud, Benoît Jacob, et al. *Eigen v3*. http://eigen.tuxfamily.org. 2010.

[106] Abhishek Gupta, Russell Mendonca, YuXuan Liu, Pieter Abbeel, and Sergey Levine. "Meta-reinforcement learning of structured exploration strategies". In: *Advances in Neural Information Processing Systems*. 2018, pp. 5302–5311.

[107] Tuomas Haarnoja, Kristian Hartikainen, Pieter Abbeel, and Sergey Levine. "Latent Space Policies for Hierarchical Reinforcement Learning". In: *International Conference on Machine Learning*. 2018, pp. 1846–1855.

[108] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*. 2018. URL: https://openreview.net/forum?id=HJjvxl-Cb.

[109] David Hahn, Pol Banzet, James M Bern, and Stelian Coros. "Real2sim: Visco-elastic parameter estimation from dynamic motion". In: *ACM Transactions on Graphics (TOG)* 38.6 (2019), pp. 1–13.

[110] Jun Han and Qiang Liu. "Stein variational gradient descent without gradient". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 1900–1908.

[111] Nikolaus Hansen. *The CMA Evolution Strategy: A Tutorial*. 2016. arXiv: 1604.00772 [cs.LG].

[112] Karol Hausman, Scott Niekum, Sarah Osentoski, and Gaurav S. Sukhatme. "Active articulation model estimation through interactive perception". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 3305–3312. ISBN: 978-1-4799-6923-4. DOI: 10.1109/ICRA.2015.7139655.

[113] Karol Hausman, Jost Tobias Springenberg, Ziyu Wang, Nicolas Heess, and Martin Riedmiller. "Learning an Embedding Space for Transferable Robot Skills". In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=rk07ZXZRb.

[114] Siyu He, Yin Li, Yu Feng, Shirley Ho, Siamak Ravanbakhsh, Wei Chen, and Barnabás Póczos. "Learning to predict the cosmological structure formation". en. In: *Proceedings of the National Academy of Sciences of the United States of America* 116.28 (2019), pp. 13825–13832.

[115] Zhanpeng He, Ryan Julian, Eric Heiden, Hejia Zhang, Stefan Schaal, Joseph Lim, Gaurav Sukhatme, and Karol Hausman. "Simulator Predictive Control: Using Learned Task Representations and MPC for Zero-Shot Generalization and Sequencing". In: *Neural Information Processing Systems (NIPS) 2018 Deep RL Workshop* (2018).

[116] Nicolas Heess, Gregory Wayne, Yuval Tassa, Timothy P. Lillicrap, Martin A. Riedmiller, and David Silver. "Learning and Transfer of Modulated Locomotor Controllers". In: *CoRR* abs/1610.05182 (2016). URL: http://arxiv.org/abs/1610.05182.

[117] Eric Heiden, Christopher E. Denniston, David Millard, Fabio Ramos, and Gaurav S. Sukhatme. "Probabilistic Inference of Simulation Parameters via Parallel Differentiable Simulation". In: *International Conference on Robotics and Automation (ICRA)*. IEEE. 2022.

[118] Eric Heiden, Karol Hausman, Gaurav S. Sukhatme, and Ali-akbar Agha-mohammadi. "Planning high-speed safe trajectories in confidence-rich maps". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017, pp. 2880–2886. DOI: 10.1109/IROS.2017.8206120.

[119] Eric Heiden, Ziang Liu, Ragesh K. Ramachandran, and Gaurav S. Sukhatme. "Physics-based Simulation of Continuous-Wave LIDAR for Localization, Calibration and Tracking". In: *International Conference on Robotics and Automation (ICRA)*. IEEE. 2020.

[120] Eric Heiden, Ziang Liu, Vibhav Vineet, Erwin Coumans, and Gaurav S. Sukhatme. *Inferring Articulated Rigid Body Dynamics from RGBD Video*. 2022. DOI: 10.48550/ARXIV.2203.10488.

[121] Eric Heiden, Miles Macklin, Yashraj S Narang, Dieter Fox, Animesh Garg, and Fabio Ramos. "DiSECt: A Differentiable Simulation Engine for Autonomous Robotic Cutting". In: *Proceedings of Robotics: Science and Systems*. Virtual, 2021. DOI: 10.15607/RSS.2021.XVII.067.

[122] Eric Heiden, David Millard, Erwin Coumans, Yizhou Sheng, and Gaurav S Sukhatme. “NeuralSim: Augmenting Differentiable Simulators with Neural Networks”. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 2021. URL: https://github.com/google-research/tiny-differentiable-simulator.

[123] Eric Heiden, David Millard, and Gaurav S. Sukhatme. “Real2Sim Transfer using Differentiable Physics”. In: *R:SS Workshop on Closing the Reality Gap in Sim2real Transfer for Robotic Manipulation* (2019).

[124] Eric Heiden, David Millard, Hejia Zhang, and Gaurav S. Sukhatme. “Interactive Differentiable Simulation”. In: *CoRR* abs/1905.10706 (2019). arXiv: 1905.10706. URL: http://arxiv.org/abs/1905.10706.

[125] Eric Heiden, Luigi Palmieri, Leonard Bruns, Kai O. Arras, Gaurav S. Sukhatme, and Sven Koenig. “Bench-MR: A Motion Planning Benchmark for Wheeled Mobile Robots”. In: *IEEE Robotics and Automation Letters* 6.3 (2021), pp. 4536–4543. DOI: 10.1109/LRA.2021.3068913.

[126] Eric Heiden, Luigi Palmieri, Sven Koenig, Kai O. Arras, and Gaurav S. Sukhatme. “Gradient-Informed Path Smoothing for Wheeled Mobile Robots”. In: *International Conference on Robotics and Automation (ICRA)*. IEEE. 2018.

[127] Eric Heiden, Daniel Pastor, Pradyumna Vyshnav, and Ali-Akbar Agha-Mohammadi. “Heterogeneous sensor fusion via confidence-rich 3D grid mapping: Application to physical robots”. In: *International Symposium on Experimental Robotics*. Springer. 2018, pp. 725–736.

[128] Paul Henderson and Vittorio Ferrari. “Learning to Generate and Reconstruct 3D Meshes with only 2D Supervision”. In: *British Machine Vision Conference (BMVC)*. 2018.

[129] Geoffrey E Hinton and Ruslan R Salakhutdinov. “Reducing the dimensionality of data with neural networks”. In: *Science* 313.5786 (2006), pp. 504–507.

[130] Hirohiko Kawata, Kohei Miyachi, Yoshitaka Hara, Akihisa Ohya, and Shin’ichi Yuta. “A method for estimation of lightness of objects with intensity data from SOKUIKI sensor”. In: *IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*. 2008, pp. 661–664. DOI: 10.1109/MFI.2008.4648020.

[131] Jonathan Ho and Stefano Ermon. “Generative adversarial imitation learning”. In: *Advances in neural information processing systems* 29 (2016).

[132] Sebastian Höfer, Kostas E. Bekris, Ankur Handa, Juan Camilo Gamboa Higuera, Florian Golemo, Melissa Mozifian, Christopher G. Atkeson, Dieter Fox, Ken Goldberg, John Leonard, C. Karen Liu, Jan Peters, Shuran Song, Peter Welinder, and Martha White. “Perspectives on Sim2Real Transfer for Robotics: A Summary of the R: SS 2020 Workshop”. In: *CoRR* abs/2012.03806 (2020). arXiv: 2012.03806. URL: https://arxiv.org/abs/2012.03806.

[133] Matthew D. Hoffman and Andrew Gelman. “The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo”. In: *Journal of Machine Learning Research* 15.47 (2014), pp. 1593–1623. URL: http://jmlr.org/papers/v15/hoffman14a.html.

[134] Peter C Horak and Jeff C Trinkle. “On the similarities and differences among contact models in robot simulation”. In: *IEEE Robotics and Automation Letters* 4.2 (2019), pp. 493–499.

[135] Kelvin Hsu and Fabio Ramos. “Bayesian Learning of Conditional Kernel Mean Embeddings for Automatic Likelihood-Free Inference”. In: *Proceedings of Machine Learning Research*. Ed. by Kamalika Chaudhuri and Masashi Sugiyama. Vol. 89. Proceedings of Machine Learning Research. 2019, pp. 2631–2640.

[136] Yixin Hu, Qingnan Zhou, Xifeng Gao, Alec Jacobson, Denis Zorin, and Daniele Panozzo. “Tetrahedral Meshing in the Wild”. In: *ACM Trans. Graph.* 37.4 (2018), 60:1–60:14. ISSN: 0730-0301.

[137] Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. “DiffTaichi: Differentiable Programming for Physical Simulation”. In: *ICLR* (2020).

[138] Yuanming Hu, Yu Fang, Ziheng Ge, Ziyin Qu, Yixin Zhu, Andre Pradhana, and Chenfanfu Jiang. “A Moving Least Squares Material Point Method with Displacement Discontinuity and Two-Way Rigid Body Coupling”. In: *ACM Transactions on Graphics (TOG)* 37.4 (2018), p. 150.

[139] Yuanming Hu, Jiancheng Liu, Andrew Spielberg, Joshua B Tenenbaum, William T Freeman, Jiajun Wu, Daniela Rus, and Wojciech Matusik. “ChainQueen: A Real-Time Differentiable Physical Simulator for Soft Robotics”. In: *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)* (2019).

[140] Zhiao Huang, Yuanming Hu, Tao Du, Siyuan Zhou, Hao Su, J. Tenenbaum, and Chuang Gan. “PlasticineLab: A Soft-Body Manipulation Benchmark with Differentiable Physics”. In: *ArXiv* abs/2104.03311 (2021).

[141] Thomas JR Hughes. *The finite element method: linear static and dynamic finite element analysis*. Courier Corporation, 2012.

[142] Kenneth H Hunt and Frank R Erskine Crossley. “Coefficient of restitution interpreted as damping in vibroimpact”. In: (1975).

[143] Du Q Huynh. “Metrics for 3D rotations: Comparison and analysis”. In: *Journal of Mathematical Imaging and Vision* 35.2 (2009), pp. 155–164.

[144] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. “Learning agile and dynamic motor skills for legged robots”. In: *Science Robotics* 4.26 (2019), eaau5872.

[145] Michael Innes. *Don’t Unroll Adjoint: Differentiating SSA-Form Programs*. 2019. arXiv: 1810.07951 [cs.PL].

[146] Ajinkya Jain, Stephen Giguere, Rudolf Lioutikov, and Scott Niekum. “Distributional Depth-Based Estimation of Object Articulation Models”. In: *Conference on Robot Learning*. 2021. URL: https://openreview.net/forum?id=H1-uwiTbY9z.

[147] Ajinkya Jain, Rudolf Lioutikov, Caleb Chuck, and Scott Niekum. "ScrewNet: Category-Independent Articulation Model Estimation From Depth Images Using Screw Theory". In: *arXiv preprint*. 2020.

[148] Wenzel Jakob. *Enoki: structured vectorization and differentiation on modern processor architectures*. https://github.com/mitsuba-renderer/enoki. 2019.

[149] Prajjwal Jamdagni and Yan-Bin Jia. "Robotic cutting of solids based on fracture mechanics and FEM". In: *IEEE International Conference on Intelligent Robots and Systems (IROS)*. 2019, pp. 8246–8251.

[150] Stephen James, Andrew J Davison, and Edward Johns. "Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task". In: *Conference on Robot Learning (CoRL)* (2017).

[151] Eric Jang, Shixiang Gu, and Ben Poole. "Categorical reparameterization with gumbel-softmax". In: *International Conference on Learning Representations* (2017).

[152] Michael Janner, Sergey Levine, William T. Freeman, Joshua B. Tenenbaum, Chelsea Finn, and Jiajun Wu. "Reasoning About Physical Interactions with Object-Centric Models". In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=HJx9EhC9tQ.

[153] L. Jeřábková and T. Kuhlen. "Stable Cutting of Deformable Objects in Virtual Environments Using XFEM". In: *IEEE Computer Graphics and Applications* 29.2 (2009), pp. 61–71.

[154] Yifeng Jiang and C. Karen Liu. "Data-Augmented Contact Model for Rigid Body Simulation". In: *CoRR* abs/1803.04019 (2018). arXiv: 1803.04019. URL: http://arxiv.org/abs/1803.04019.

[155] Ryan C Julian, Eric Heiden, Zhanpeng He, Hejia Zhang, Stefan Schaal, Joseph Lim, Gaurav S Sukhatme, and Karol Hausman. "Scaling simulation-to-real transfer by learning composable robot skills". In: *International Symposium on Experimental Robotics*. Springer. 2018. URL: https://ryanjulian.me/iser%5C_2018.pdf.

[156] Ryan C Julian, Eric Heiden, Zhanpeng He, Hejia Zhang, Stefan Schaal, Joseph J Lim, Gaurav S Sukhatme, and Karol Hausman. "Scaling simulation-to-real transfer by learning a latent space of robot skills". In: *The International Journal of Robotics Research* 39.10-11 (2020), pp. 1259–1278. DOI: 10.1177/0278364920944474. eprint: https://doi.org/10.1177/0278364920944474.

[157] Sanket Kamthe and Marc Deisenroth. "Data-Efficient Reinforcement Learning with Probabilistic Model Predictive Control". In: *International Conference on Artificial Intelligence and Statistics*. 2018, pp. 1701–1710.

[158] Hirohiko Kawata, Akihisa Ohya, Shinichi Yuta, Wagle Santosh, and Toshihiro Mori. "Development of ultra-small lightweight optical range sensor system". In: *International Conference on Intelligent Robots and Systems*. IEEE/RSJ. 2005, pp. 1078–1083.

[159] Amir R Khoei. *Extended finite element method: theory and applications*. John Wiley & Sons, 2014.

[160] Pradeep K. Khosla and Takeo Kanade. “Parameter identification of robot dynamics”. In: *IEEE Conference on Decision and Control*. 1985, pp. 1754–1760. DOI: 10.1109/CDC.1985.268838.

[161] Patrick Kidger and Terry Lyons. “Signatory: differentiable computations of the signature and logsignature transforms, on both CPU and GPU”. In: *International Conference on Learning Representations*. https://github.com/patrick-kidger/signatory. 2021.

[162] Donghyun Kim, Jared Di Carlo, Benjamin Katz, Gerardo Bledt, and Sangbae Kim. *Highly Dynamic Quadruped Locomotion via Whole-Body Impulse Control and Model Predictive Control*. 2019. arXiv: 1909.06586 [cs.RO].

[163] Kyunam Kim, Adrian K. Agogino, Deaho Moon, Laqshya Taneja, Aliakbar Toghyan, Borna Dehghani, Vytas SunSpiral, and Alice M. Agogino. “Rapid prototyping design and control of tensegrity soft robot for locomotion”. In: *IEEE International Conference on Robotics and Biomimetics (ROBIO 2014)*. 2014, pp. 7–14. DOI: 10.1109/ROBIO.2014.7090299.

[164] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *International Conference for Learning Representations* (2015).

[165] L. Kneip, F. Tache, G. Caprari, and R. Siegwart. “Characterization of the compact Hokuyo URG-04LX 2D laser range scanner”. In: *2009 IEEE International Conference on Robotics and Automation*. 2009, pp. 1447–1454. DOI: 10.1109/ROBOT.2009.5152579.

[166] 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*. 2007, pp. 742–747. DOI: 10.1109/ROBOT.2007.363075.

[167] N. Koenig and A. Howard. “Design and use paradigms for Gazebo, an open-source multi-robot simulator”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Vol. 3. 2004, 2149–2154 vol.3. DOI: 10.1109/IROS.2004.1389727.

[168] Svetoslav Kolev and Emanuel Todorov. “Physically consistent state estimation and system identification for contacts”. In: *International Conference on Humanoid Robots* (2015), pp. 1036–1043.

[169] Twan Koolen and Robin Deits. “Julia for robotics: simulation and real-time control in a high-level programming language”. In: *International Conference on Robotics and Automation*. 2019.

[170] Dan Koschier, Jan Bender, and Nils Thuerey. “Robust EXtended Finite Elements for Complex Cutting of Deformables”. In: *ACM Trans. Graph.* 36.4 (2017).

[171] Dan Koschier, Sebastian Lipponer, and Jan Bender. “Adaptive Tetrahedral Meshes for Brittle Fracture Simulation”. In: *Symposium on Computer Animation* (2014).

[172] Krzysztof R. Kozlowski. *Modelling and Identification in Robotics*. en. Advances in Industrial Control. London: Springer-Verlag, 1998. ISBN: 978-3-540-76240-9. DOI: 10.1007/978-1-4471-0429-2. (Visited on 08/20/2021).

[173] Sanjay Krishnan, Animesh Garg, Richard Liaw, Brijen Thananjeyan, Lauren Miller, Florian T Pokorny, and Ken Goldberg. "SWIRL: A Sequential Windowed Inverse Reinforcement Learning Algorithm for Robot Tasks With Delayed Rewards". In: *International Journal of Robotics Research (IJRR)* (2018).

[174] Oliver Kroemer and Gaurav S. Sukhatme. "Learning Relevant Features for Manipulation Skills using Meta-Level Priors". In: *CoRR* abs/1605.04439 (2016). arXiv: 1605.04439. URL: http://arxiv.org/abs/1605.04439.

[175] Samuli Laine, Janne Hellsten, Tero Karras, Yeongho Seol, Jaakko Lehtinen, and Timo Aila. "Modular Primitives for High-Performance Differentiable Rendering". In: *ACM Transactions on Graphics* 39.6 (2020).

[176] Alexander Lambert, Adam Fishman, Dieter Fox, Byron Boots, and Fabio Ramos. "Stein Variational Model Predictive Control". In: *Conference on Robot Learning (CoRL)* (2020). URL: https://corlconf.github.io/corl2020/paper_282/.

[177] Nathan O. Lambert, Brandon Amos, Omry Yadan, and Roberto Calandra. "Objective Mismatch in Model-based Reinforcement Learning". In: *Conference on Learning for Dynamics and Control (L4DC)*. Ed. by Alexandre M. Bayen, Ali Jadbabaie, George J. Pappas, Pablo A. Parrilo, Benjamin Recht, Claire J. Tomlin, and Melanie N. Zeilinger. Vol. 120. Proceedings of Machine Learning Research. PMLR, 2020, pp. 761–770. URL: http://proceedings.mlr.press/v120/lambert20a.html.

[178] João Rui Leal and Brad Bell. *joaoleal/CppADCodeGen: CppAD 2017*. Version v2.2.0. 2017. DOI: 10.5281/zenodo.836832.

[179] Jeongseok Lee, Michael X. Grey, Sehoon Ha, Tobias Kunz, Sumit Jain, Yuting Ye, Siddhartha S. Srinivasa, Mike Stilman, and C. Karen Liu. "DART: Dynamic Animation and Robotics Toolkit". In: *Journal of Open Source Software* 3.22 (2018), p. 500.

[180] Youngwoon Lee, Shao-Hua Sun, Sriram Somasundaram, Edward S Hu, and Joseph J Lim. "Composing Complex Skills by Learning Transition Policies". In: *International Conference on Learning Representations*. 2019.

[181] Q. Lelidec, I. Kalevatykh, I. Laptev, C. Schmid, and J. Carpentier. "Differentiable simulation for physical system identification". In: *IEEE Robotics and Automation Letters* (2021), pp. 1–1. DOI: 10.1109/LRA.2021.3062323.

[182] Ian Lenz, Ross A Knepper, and Ashutosh Saxena. "DeepMPC: Learning deep latent features for model predictive control". In: *Robotics: Science and Systems* (2015).

[183] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. "End-to-end training of deep visuomotor policies". In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 1334–1373.

[184] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection". In: *The International Journal of Robotics Research* 37.4-5 (2018), pp. 421–436. DOI: 10.1177/0278364917710318. eprint: https://doi.org/10.1177/0278364917710318.

[185] Chunyuan Li, Changyou Chen, David Carlson, and Lawrence Carin. “Preconditioned stochastic gradient Langevin dynamics for deep neural networks”. In: *AAAI*. 2016.

[186] Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. “Differentiable Monte Carlo ray tracing through edge sampling”. In: *SIGGRAPH Asia 2018 Technical Papers*. ACM. 2018, p. 222.

[187] Weiwei Li and Emanuel Todorov. “Iterative linear quadratic regulator design for nonlinear biological movement systems.” In: *International Conference on Informatics in Control, Automation and Robotics*. 2004.

[188] Yancong Li, Xiaoyong Du, Jinhai Wang, and Chai Lei. “Study on mechanical properties of apple picking damage”. In: *Proceedings of the 2017 3rd International Forum on Energy, Environment Science and Materials (IFEESM 2017)*. Atlantis Press, 2018, pp. 1666–1670.

[189] Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B. Tenenbaum, and Antonio Torralba. “Learning Particle Dynamics for Manipulating Rigid Bodies, Deformable Objects, and Fluids”. In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=rJgbSn09Ym.

[190] Junbang Liang, Ming Lin, and Vladlen Koltun. “Differentiable Cloth Simulation for Inverse Problems”. In: *Advances in Neural Information Processing Systems*. 2019, pp. 771–780.

[191] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. “Continuous control with deep reinforcement learning”. In: *CoRR* abs/1509.02971 (2015). arXiv: 1509.02971. URL: http://arxiv.org/abs/1509.02971.

[192] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. “Microsoft COCO: Common Objects in Context”. In: *Computer Vision – ECCV 2014*. Ed. by David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars. Springer International Publishing, 2014, pp. 740–755. ISBN: 978-3-319-10602-1.

[193] Dong C. Liu and Jorge Nocedal. “On the limited memory BFGS method for large scale optimization”. In: *Mathematical Programming* 45.1 (1989), pp. 503–528. ISSN: 1436-4646. DOI: 10.1007/BF01589116.

[194] MB Liu and GR Liu. “Smoothed particle hydrodynamics (SPH): An overview and recent developments”. In: *Archives of computational methods in engineering* 17 (2010), pp. 25–76.

[195] Qiang Liu. “Stein Variational Gradient Descent as Gradient Flow”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/file/17ed8abedc255908be746d245e50263a-Paper.pdf.

[196] Qiang Liu and Dilin Wang. “Stein Variational Gradient Descent: A General Purpose Bayesian Inference Algorithm”. In: *Advances in Neural Information Processing Systems* 29 (2016). Ed. by D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett.

[197] Shichen Liu, Weikai Chen, Tianye Li, and Hao Li. “Soft Rasterizer: Differentiable Rendering for Unsupervised Single-View Mesh Reconstruction”. In: *arXiv preprint arXiv:1901.05567* (2019).

[198] Yizhou Liu, Fusheng Zha, Lining Sun, Jingxuan Li, Mantian Li, and Xin Wang. "Learning Articulated Constraints From a One-Shot Demonstration for Robot Manipulation Planning". In: *IEEE Access* 7 (2019), pp. 172584–172596. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2953894.

[199] Zhijian Liu, Jiajun Wu, Zhenjia Xu, Chen Sun, Kevin Murphy, William T. Freeman, and Joshua B. Tenenbaum. "Modeling Parts, Structure, and System Dynamics via Predictive Learning". In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=rJe1OiC5K7.

[200] Lennart Ljung. *System Identification (2nd Ed.): Theory for the User*. USA: Prentice Hall PTR, 1999. ISBN: 0136566952.

[201] Philip Long, Amine Moughlbay, Wisama Khalil, and Philippe Martinet. "Robotic meat cutting". In: *ICT-PAMM Workshop*. 2013.

[202] Zichao Long, Yiping Lu, Xianzhong Ma, and Bin Dong. "Pde-net: Learning pdes from data". In: *International Conference on Machine Learning*. 2018, pp. 3208–3216.

[203] Matthew M Loper and Michael J Black. "OpenDR: An approximate differentiable renderer". In: *European Conference on Computer Vision*. Springer. 2014, pp. 154–169.

[204] Yao Lu, Karol Hausman, Yevgen Chebotar, Mengyuan Yan, Eric Jang, Alexander Herzog, Ted Xiao, Alex Irpan, Mohi Khansari, Dmitry Kalashnikov, and Sergey Levine. "AW-Opt: Learning Robotic Skills with Imitation andReinforcement at Scale". In: *5th Annual Conference on Robot Learning*. 2021.

[205] Michael Lutter, Christian Ritter, and Jan Peters. "Deep Lagrangian Networks: Using Physics as Model Prior for Deep Learning". In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=BklHpjCqKm.

[206] Michael Lutter, Johannes Silberbauer, Joe Watson, and Jan Peters. "Differentiable physics models for real-world offline model-based reinforcement learning". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 4163–4170.

[207] Rosa M. Sánchez-Banderas and Miguel A. Otaduy. "Strain Rate Dissipation for Elastic Deformations". In: *Computer Graphics Forum* 37.8 (2018), pp. 161–170.

[208] Miles Macklin, Kenny Erleben, Matthias Müller, Nuttapong Chentanez, Stefan Jeschke, and Zach Corse. "Local Optimization for Robust Signed Distance Field Collision". In: *Proc. ACM Comput. Graph. Interact. Tech.* 3.1 (2020).

[209] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. "The concrete distribution: A continuous relaxation of discrete random variables". In: *International Conference on Learning Representations* (2017).

[210] Rolf Mahnken. "Identification of material parameters for constitutive equations". In: *Encyclopedia of Computational Mechanics Second Edition* (2017), pp. 1–21.

[211] Charles C. Margossian. “A review of automatic differentiation and its efficient implementation”. In: *WIREs Data Mining and Knowledge Discovery* 9.4 (2019).

[212] Roberto Martín Martín and Oliver Brock. “Online interactive perception of articulated objects with multi-level recursive estimation based on task-specific priors”. In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2014, pp. 2494–2501. ISBN: 978-1-4799-6934-0. DOI: 10.1109/IROS.2014.6942902.

[213] C. Matl, Y. Narang, R. Bajcsy, F. Ramos, and D. Fox. “Inferring the Material Properties of Granular Media for Robotic Tasks”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020, pp. 2770–2777.

[214] Carolyn Matl, Yashraj Narang, Ruzena Bajcsy, Fabio Ramos, and Dieter Fox. “Inferring the Material Properties of Granular Media for Robotic Tasks”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. ISSN: 2577-087X. 2020, pp. 2770–2777. DOI: 10.1109/ICRA40945.2020.9197063.

[215] Carolyn Matl, Yashraj S. Narang, Dieter Fox, Ruzena Bajcsy, and Fabio Ramos. “STReSSD: Sim-To-Real from Sound for Stochastic Dynamics”. In: *Conference on Robot Learning* (2020).

[216] Steven L McCarty, Laura M Burke, and Melissa McGuire. “Parallel Monotonic Basin Hopping for low thrust trajectory optimization”. In: *2018 Space Flight Mechanics Meeting*. 2018, p. 1452.

[217] Bhairav Mehta, Manfred Diaz, Florian Golemo, C. Pal, and L. Paull. “Active Domain Randomization”. In: *Conference on Robot Learning*. 2019.

[218] Bhairav Mehta, Manfred Diaz, Florian Golemo, Christopher Pal, and Liam Paull. “Active Domain Randomization”. In: *Conference on Robot Learning* (2019).

[219] Bhairav Mehta, Manfred Diaz, Florian Golemo, Christopher J. Pal, and Liam Paull. “Active Domain Randomization”. In: *Proceedings of the Conference on Robot Learning*. Ed. by Leslie Pack Kaelbling, Danica Kragic, and Komei Sugiura. Vol. 100. Proceedings of Machine Learning Research. PMLR, 2020, pp. 1162–1176. URL: https://proceedings.mlr.press/v100/mehta20a.html.

[220] Bhairav Mehta, Ankur Handa, Dieter Fox, and Fabio Ramos. “A User’s Guide to Calibrating Robotics Simulators”. In: *Conference on Robot Learning* (2020).

[221] M Eugene Merchant. “Mechanics of the metal cutting process. I. Orthogonal cutting and a type 2 chip”. In: *Journal of applied physics* 16.5 (1945), pp. 267–275.

[222] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. “Meta-Learning with Temporal Convolutions”. In: *CoRR* abs/1707.03141 (2017). arXiv: 1707.03141. URL: http://arxiv.org/abs/1707.03141.

[223] I. Mitsioni, Y. Karayiannidis, J. A. Stork, and D. Kragic. “Data-Driven Model Predictive Control for the Contact-Rich Task of Food Cutting”. In: *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. 2019, pp. 244–250.

[224] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), pp. 529–533. ISSN: 14764687. DOI: 10.1038/nature14236. arXiv: 1312.5602.

[225] Nicolas Moës, John Dolbow, and Ted Belytschko. "A finite element method for crack growth without remeshing". In: *International journal for numerical methods in engineering* 46.1 (1999), pp. 131–150.

[226] Artem Molchanov, Tao Chen, Wolfgang Hönig, James A. Preiss, Nora Ayanian, and Gaurav S. Sukhatme. "Sim-to-(Multi)-Real: Transfer of Low-Level Robust Control Policies to Multiple Quadrotors". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2019, Macau, SAR, China, November 3-8, 2019*. 2019, pp. 59–66. DOI: 10.1109/IROS40897.2019.8967695.

[227] Neil Molino, Zhaosheng Bao, and Ron Fedkiw. "A Virtual Node Algorithm for Changing Mesh Topology during Simulation". In: *ACM Trans. Graph.* 23.3 (2004), pp. 385–392.

[228] JJ Monaghan. "Smoothed particle hydrodynamics". In: *Annual Review of Astronomy and Astrophysics* 30 (1992), pp. 543–574.

[229] Andrew W. Moore. "Fast, Robust Adaptive Control by Learning Only Forward Models". In: *Proceedings of the 4th International Conference on Neural Information Processing Systems*. NIPS'91. Denver, Colorado: Morgan Kaufmann Publishers Inc., 1991, pp. 571–578. ISBN: 1558602224.

[230] Andrew W. Moore. "Fast, Robust Adaptive Control by Learning only Forward Models". In: *Advances in Neural Information Processing Systems*. Ed. by J. E. Moody, S. J. Hanson, and R. P. Lippmann. Morgan-Kaufmann, 1992, pp. 571–578. URL: http://papers.nips.cc/paper/585-fast-robust-adaptive-control-by-learning-only-forward-models.pdf.

[231] Hans P Moravec. "Sensor fusion in certainty grids for mobile robots". In: *AI magazine* 9.2 (1988), p. 61.

[232] Seyyed Javad Mousavizadeh, Ali N. Mashayekhi, Amir Daraei Garmakhany, Ab Ehtesham Nia, and Jafari M. "Evaluation of Some Physical Properties of Cucumber (Cucumis sativus L.)." In: *Journal of Agricultural Science and Technology* 4 (2010), pp. 107–115.

[233] Melissa Mozifian, Juan Camilo Gamboa Higuera, David Meger, and Gregory Dudek. "Learning Domain Randomization Distributions for Training Robust Locomotion Policies". In: *arXiv preprint arXiv:1906.00410* (2019).

[234] Damian Mrowca, Chengxu Zhuang, Elias Wang, Nick Haber, Li Fei-Fei, Joshua B Tenenbaum, and Daniel LK Yamins. "Flexible Neural Representation for Physics Prediction". In: *Advances in Neural Information Processing Systems*. 2018.

[235] Jiteng Mu, Weichao Qiu, Adam Kortylewski, Alan Yuille, Nuno Vasconcelos, and Xiaolong Wang. "A-SDF: Learning Disentangled Signed Distance Functions for Articulated Shape Representation". In: *arXiv preprint arXiv: 2104.07645* (2021).

[236] Xiaoqian Mu, Yuechuan Xue, and Yan-Bin Jia. “Robotic cutting: Mechanics and control of knife motion”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2019, pp. 3066–3072.

[237] Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. “Position based dynamics”. In: *Journal of Visual Communication and Image Representation* 18.2 (2007), pp. 109–118.

[238] Fabio Muratore, Christian Eilers, M. Gienger, and Jan Peters. “Bayesian Domain Randomization for Sim-to-Real Transfer”. In: *ArXiv* abs/2003.02471 (2020).

[239] J. Krishna Murthy, Miles Macklin, Florian Golemo, Vikram Voleti, Linda Petrini, Martin Weiss, Breandan Considine, Jérôme Parent-Lévesque, Kevin Xie, Kenny Erleben, Liam Paull, Florian Shkurti, Derek Nowrouzezahrai, and Sanja Fidler. “gradSim: Differentiable simulation for system identification and visuomotor control”. In: *International Conference on Learning Representations*. 2021.

[240] Yashraj Narang, Balakumar Sundaralingam, Miles Macklin, Arsalan Mousavian, and Dieter Fox. “Sim-to-Real for Robotic Tactile Sensing via Physics-Based Simulation and Learned Latent Projections”. In: *International Conference on Robotics and Automation* (2021).

[241] Yashraj S. Narang, Karl Van Wyk, Arsalan Mousavian, and Dieter Fox. “Interpreting and Predicting Tactile Signals via a Physics-Based and Data-Driven Framework”. In: *arXiv:2006.03777 [cs]* (2020). arXiv: 2006.03777. URL: http://arxiv.org/abs/2006.03777 (visited on 06/18/2021).

[242] Scott Niekum, Sarah Osentoski, Christopher G Atkeson, and Andrew G Barto. “Online bayesian changepoint detection for articulated motion models”. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2015, pp. 1468–1475.

[243] Merlin Nimier-David, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. “Mitsuba 2: A Retargetable Forward and Inverse Renderer”. In: *Transactions on Graphics (Proceedings of SIGGRAPH Asia)* 38.6 (2019). DOI: 10.1145/3355089.3356498.

[244] Brett Ninness and Soren Henriksen. “Bayesian system identification via Markov chain Monte Carlo techniques”. en. In: *Automatica* 46.1 (2010), pp. 40–51. ISSN: 0005-1098. DOI: 10.1016/j.automatica.2009.10.015. (Visited on 02/22/2021).

[245] Atsuhiro Noguchi, Umar Iqbal, Jonathan Tremblay, Tatsuya Harada, and Orazio Gallo. *Watch It Move: Unsupervised Discovery of 3D Joints for Re-Posing of Articulated Objects*. 2021. arXiv: 2112.11347 [cs.CV].

[246] Simon O’Callaghan, Fabio T Ramos, and Hugh Durrant-Whyte. “Contextual occupancy maps using Gaussian processes”. In: *International Conference on Robotics and Automation*. IEEE. 2009, pp. 1054–1060.

[247] Yoichi Okubo, Cang Ye, and Johann Borenstein. “Characterization of the Hokuyo URG-04LX laser rangefinder for mobile robot obstacle negotiation”. In: *Unmanned Systems Technology XI*. Vol. 7332. International Society for Optics and Photonics. 2009, p. 733212.

[248] Junjun Pan, Junxuan Bai, Xin Zhao, Aimin Hao, and Hong Qin. "Real-time haptic manipulation and cutting of hybrid soft tissue models by extended position-based dynamics". In: *Computer Animation and Virtual Worlds* 26.3-4 (2015), pp. 321–335.

[249] George Papamakarios and Iain Murray. "Fast epsilon-free Inference of Simulation Models with Bayesian Conditional Density Estimation". In: *Advances in Neural Information Processing Systems*. Ed. by D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett. Vol. 29. Curran Associates, Inc., 2016.

[250] Pablo A Parrilo. *Semidefinite Optimization and Convex Algebraic Geometry*. SIAM, 2012.

[251] Peter Pastor, Mrinal Kalakrishnan, Ludovic Righetti, and Stefan Schaal. "Towards Associative Skill Memories". In: *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*. 2012, pp. 309–315. DOI: 10.1109/HUMANOIDS.2012.6651537.

[252] Christoph J Paulus, Lionel Untereiner, Hadrien Courtecuisse, Stéphane Cotin, and David Cazier. "Virtual cutting of deformable objects based on efficient topological operations". In: *The Visual Computer* 31.6 (2015), pp. 831–841.

[253] M. Peifer and J. Timmer. "Parameter estimation in ordinary differential equations for biochemical processes using the method of multiple shooting". en. In: *IET Systems Biology* 1.2 (2007), pp. 78–88. ISSN: 1751-8849, 1751-8857. DOI: 10.1049/iet-syb:20060067. (Visited on 06/06/2021).

[254] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. "DeepMimic: Example-guided Deep Reinforcement Learning of Physics-based Character Skills". In: *ACM Trans. Graph.* 37.4 (July 2018), 143:1–143:14. ISSN: 0730-0301. DOI: 10.1145/3197517.3201311.

[255] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. "Sim-to-real transfer of robotic control with dynamics randomization". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 1–8.

[256] Xue Bin Peng, Erwin Coumans, Tingnan Zhang, Tsang-Wei Edward Lee, Jie Tan, and Sergey Levine. "Learning Agile Robotic Locomotion Skills by Imitating Animals". In: *Robotics: Science and Systems*. July 2020. DOI: 10.15607/RSS.2020.XVI.064.

[257] V. Peterka. "Bayesian Approach to System Identification". In: *Trends and Progress in System Identification*. Ed. by Pieter Eykhoff. Pergamon, 1981, pp. 239–304. ISBN: 978-0-08-025683-2. DOI: https://doi.org/10.1016/B978-0-08-025683-2.50013-2.

[258] G. Peyré and M. Cuturi. *Computational Optimal Transport: With Applications to Data Science*. Foundations and trends in machine learning. Now Publishers, 2019. ISBN: 9781680835502.

[259] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically based rendering: From theory to implementation*. Morgan Kaufmann, 2016.

[260] John C Platt and Alan H Barr. "Constrained differential optimization for neural networks". In: *Advances in Neural Information Processing Systems* (1988).

[261] John C Platt and Alan H Barr. “Constrained differential optimization for neural networks”. In: (1988).

[262] Athanasios S. Polydoros and Lazaros Nalpantidis. “Survey of Model-Based Reinforcement Learning: Applications on Robotics”. In: *Journal of Intelligent & Robotic Systems* 86.2 (2017), pp. 153–173. ISSN: 1573-0409. DOI: 10.1007/s10846-017-0468-y.

[263] Lev Semenovich Pontryagin. *Mathematical theory of optimal processes*. Routledge, 2018.

[264] James A Preiss, Karol Hausman, and Gaurav S Sukhatme. “Learning a System-ID Embedding Space for Domain Specialization with Deep Reinforcement Learning”. In: *NeurIPS Workshop on Reinforcement Learning under Partial Observability* (2018).

[265] Song S. Qian, Craig A. Stow, and Mark E. Borsuk. “On Monte Carlo methods for Bayesian inference”. In: *Ecological Modelling* 159.2 (2003), pp. 269–277. ISSN: 0304-3800. DOI: https://doi.org/10.1016/S0304-3800(02)00299-5.

[266] Yi-Ling Qiao, Junbang Liang, Vladlen Koltun, and Ming C. Lin. “Scalable Differentiable Physics for Learning and Control”. In: *ICML*. 2020.

[267] Christopher Rackauckas, Yingbo Ma, Vaibhav Dixit, Xingjian Guo, Mike Innes, Jarrett Revels, Joakim Nyberg, and Vijay Ivaturi. “A Comparison of Automatic Differentiation and Continuous Sensitivity Analysis for Derivatives of Differential Equation Solutions”. In: *arXiv preprint arXiv:1812.01892* (2018).

[268] Maziar Raissi, Hessam Babaee, and Peyman Givi. *Deep Learning of Turbulent Scalar Mixing*. Tech. rep. 2018.

[269] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations”. In: *Journal of Computational Physics* 378 (2019), pp. 686–707.

[270] Janarthanan Rajendran, P. Prasanna, Balaraman Ravindran, and Mitesh M. Khapra. “ADAAPT: A Deep Architecture for Adaptive Policy Transfer from Multiple Sources”. In: *CoRR* abs/1510.02879 (2015). arXiv: 1510.02879. URL: http://arxiv.org/abs/1510.02879.

[271] Aravind Rajeswaran, Sarvjeet Ghotra, Balaraman Ravindran, and Sergey Levine. “EPOpt: Learning Robust Neural Network Policies Using Model Ensembles”. In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. 2017. URL: https://openreview.net/forum?id=SyWvgP5el.

[272] Kate Rakelly, Aurick Zhou, Deirdre Quillen, Chelsea Finn, and Sergey Levine. “Efficient Off-Policy Meta-Reinforcement Learning via Probabilistic Context Variables”. In: *arXiv preprint arXiv:1903.08254* (2019).

[273] Fabio Ramos and Lionel Ott. “Hilbert maps: scalable continuous occupancy mapping with stochastic gradient descent”. In: *The International Journal of Robotics Research* 35.14 (2016), pp. 1717–1730.

[274] Fabio Ramos, Rafael Carvalhaes Possas, and Dieter Fox. "BayesSim: adaptive domain randomization via probabilistic inference for robotics simulators". In: *Robotics: Science and Systems* (2019).

[275] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005. ISBN: 026218253X.

[276] Tomislav Reichenbach. "A dynamic simulator for humanoid robots". In: *Artificial Life and Robotics* 13.2 (2009), pp. 561–565. ISSN: 1614-7456. DOI: 10.1007/s10015-008-0508-6.

[277] Reinforcement Learning Working Group Contributors. *Garage: A toolkit for reproducible reinforcement learning*. https://github.com/rlworkgroup/garage. 2019.

[278] John D Co-Reyes, Yuxuan Liu, Abhishek Gupta, Benjamin Eysenbach, Pieter Abbeel, and Sergey Levine. "Self-Consistent Trajectory Autoencoder: Hierarchical Reinforcement Learning with Trajectory Embeddings". In: *ICML* (2018). arXiv: 1806.02813 [cs.LG]. URL: http://arxiv.org/abs/1806.02813.

[279] P. Rosenberger, M. Holder, M. Zirulnik, and H. Winner. "Analysis of Real World Sensor Behavior for Rising Fidelity of Physically Based Lidar Sensor Models". In: *2018 IEEE Intelligent Vehicles Symposium (IV)*. 2018, pp. 611–616. DOI: 10.1109/IVS.2018.8500511.

[280] Antoni Rosinol, Andrew Violette, Marcus Abate, Nathan Hughes, Yun Chang, Jingnan Shi, Arjun Gupta, and Luca Carlone. "Kimera: From SLAM to spatial perception with 3D dynamic scene graphs". In: *The International Journal of Robotics Research* 40.12-14 (2021), pp. 1510–1546. DOI: 10.1177/02783649211056674. eprint: https://doi.org/10.1177/02783649211056674.

[281] Jonas Rothfuss, Dennis Lee, Ignasi Clavera, Tamim Asfour, and Pieter Abbeel. "ProMP: Proximal Meta-Policy Search". In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=SkxXCi0qFX.

[282] Nikolaus Rott. "A multiple pendulum for the demonstration of non-linear coupling". In: *Zeitschrift für angewandte Mathematik und Physik ZAMP* 21.4 (1970), pp. 570–582.

[283] Reuven Y Rubinstein. "Optimization of computer simulation models with rare events". In: *European Journal of Operational Research* 99.1 (1997), pp. 89–112.

[284] Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. "The Earth Mover's Distance as a Metric for Image Retrieval." In: *International Journal of Computer Vision* 40.2 (2000), pp. 99–121.

[285] E. Rueckert, J. Mundo, A. Paraschos, J. Peters, and G. Neumann. "Extracting low-dimensional control variables for movement primitives". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 2015, pp. 1511–1518. DOI: 10.1109/ICRA.2015.7139390.

[286] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3rd. USA: Prentice Hall Press, 2009. ISBN: 978-0-13-604259-4.

[287] Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. "Progressive Neural Networks". In: *CoRR* abs/1606.04671 (2016). arXiv: 1606.04671. URL: http://arxiv.org/abs/1606.04671.

[288] Radu Bogdan Rusu and Steve Cousins. "3D is here: Point Cloud Library (PCL)". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China, 2011.

[289] Fereshteh Sadeghi and Sergey Levine. "CAD2RL: Real Single-Image Flight without a Single Real Image". In: *Robotics: Science and Systems(RSS)*. 2017.

[290] Steindór Sæmundsson, Katja Hofmann, and Marc Peter Deisenroth. "Meta Reinforcement Learning with Latent Variable Gaussian Processes". In: *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI 2018, Monterey, California, USA, August 6-10, 2018*. 2018, pp. 642–652. URL: http://auai.org/uai2018/proceedings/papers/235.pdf.

[291] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter W. Battaglia. *Learning to Simulate Complex Physics with Graph Networks*. 2020. arXiv: 2002.09405 [cs.LG].

[292] Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. "Graph Networks as Learnable Physics Engines for Inference and Control". In: *International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, 2018, pp. 4470–4479. URL: http://proceedings.mlr.press/v80/sanchez-gonzalez18a.html.

[293] Jason Sanders and Edward Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. en. Addison-Wesley Professional, 2010. ISBN: 978-0-13-218013-9.

[294] Stefan Schaal. "Is imitation learning the route to humanoid robots?" In: *Trends in Cognitive Sciences* 3.6 (1999), pp. 233–242. ISSN: 1364-6613. DOI: https://doi.org/10.1016/S1364-6613(99)01327-3.

[295] Stefan Schaal and Christopher G Atkeson. "Robot juggling: implementation of memory-based learning". In: *IEEE Control Systems Magazine* 14.1 (1994), pp. 57–71.

[296] Connor Schenck and Dieter Fox. "SPNets: Differentiable Fluid Dynamics for Deep Neural Networks". In: *Conference on Robot Learning* (2018).

[297] John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. "Gradient estimation using stochastic computation graphs". In: *NIPS*. 2015.

[298] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. "Trust region policy optimization". In: *International conference on machine learning*. PMLR. 2015, pp. 1889–1897.

[299] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal Policy Optimization Algorithms". In: *CoRR* abs/1707.06347 (2017). arXiv: 1707.06347. URL: http://arxiv.org/abs/1707.06347.

[300] Dale E Seborg, Duncan A Mellichamp, Thomas F Edgar, and Francis J Doyle III. *Process dynamics and control*. John Wiley & Sons, 2010.

[301] Aleksandr Segal, Dirk Haehnel, and Sebastian Thrun. "Generalized-ICP". In: *Robotics Science and Systems*. 2009.

[302] Christopher M. Seubert. "IR Reflectivity of Paint: Autonomy and CO_2 Emissions". In: *Detroit Colour Council* (2018).

[303] Jie Shan and Charles K Toth. *Topographic laser ranging and scanning: principles and processing*. CRC press, 2018.

[304] Eftychios Sifakis, Kevin G Der, and Ronald Fedkiw. "Arbitrary cutting of deformable tetrahedralized objects". In: *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*. 2007, pp. 73–80.

[305] Eftychios Sifakis, Tamar Shinar, Geoffrey Irving, and Ronald Fedkiw. "Hybrid simulation of deformable solids". In: *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*. 2007, pp. 81–90.

[306] Noah Simon, Jerome Friedman, Trevor Hastie, and Robert Tibshirani. "A sparse-group lasso". In: *Journal of computational and graphical statistics* 22.2 (2013), pp. 231–245.

[307] Breannan Smith, Fernando De Goes, and Theodore Kim. "Stable Neo-Hookean Flesh Simulation". In: *ACM Trans. Graph.* 37.2 (2018).

[308] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd. "OSQP: an operator splitting solver for quadratic programs". In: *Mathematical Programming Computation* 12.4 (2020), pp. 637–672. DOI: 10.1007/s12532-020-00179-2.

[309] David Stewart and Jeffrey C Trinkle. "An implicit time-stepping scheme for rigid body dynamics with coulomb friction". In: *IEEE International Conference on Robotics and Automation*. Vol. 1. IEEE. 2000, pp. 162–169.

[310] Jürgen Sturm, Cyrill Stachniss, and Wolfram Burgard. "A Probabilistic Framework for Learning Kinematic Models of Articulated Objects". In: *Journal of Artificial Intelligence Research* 41.2 (2011), pp. 477–526. ISSN: 1076-9757.

[311] Giovanni Sutanto, Austin Wang, Yixin Lin, Mustafa Mukadam, Gaurav Sukhatme, Akshara Rai, and Franziska Meier. "Encoding Physical Constraints in Differentiable Newton-Euler Algorithm". In: ed. by Alexandre M. Bayen, Ali Jadbabaie, George Pappas, Pablo A. Parrilo, Benjamin Recht, Claire Tomlin, and Melanie Zeilinger. Vol. 120. Proceedings of Machine Learning Research. The Cloud: PMLR, 2020, pp. 804–813. URL: http://proceedings.mlr.press/v120/sutanto20a.html.

[312] J. Tan, T. Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and V. Vanhoucke. "Sim-to-Real: Learning Agile Locomotion For Quadruped Robots". In: *ArXiv* abs/1804.10332 (2018).

[313] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy P. Lillicrap, and Martin A. Riedmiller. "DeepMind Control Suite". In: *CoRR* abs/1801.00690 (2018). arXiv: 1801.00690. URL: http://arxiv.org/abs/1801.00690.

[314] Brijen Thananjeyan, Animesh Garg, Sanjay Krishnan, Carolyn Chen, Lauren Miller, and Ken Goldberg. "Multilateral Surgical Pattern Cutting in 2D Orthotropic Gauze with Deep Reinforcement Learning Policies for Tensioning". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2017.

[315] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT Press, Cambridge, MA, 2005.

[316] T. Tieleman and G. Hinton. *Coursera: Neural netwroks for machine learning (lecture 6.5 - RMSProprop)*. 2012.

[317] Jo-Anne Ting, Aaron D'Souza, and Stefan Schaal. "Bayesian robot system identification with input and output noise". en. In: *Neural Networks* 24.1 (2011), pp. 99–108. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2010.08.011. (Visited on 08/20/2021).

[318] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. "Domain randomization for transferring deep neural networks from simulation to the real world". In: *International Conference on Intelligent Robots and Systems (IROS)*. IEEE/RSJ. 2017, pp. 23–30.

[319] E. Todorov, T. Erez, and Y. Tassa. "MuJoCo: A physics engine for model-based control". In: *International Conference on Intelligent Robots and Systems*. 2012, pp. 5026–5033. DOI: 10.1109/IROS.2012.6386109.

[320] Tina Toni, David Welch, Natalja Strelkowa, Andreas Ipsen, and Michael P.H Stumpf. "Approximate Bayesian computation scheme for parameter inference and model selection in dynamical systems". In: *Journal of The Royal Society Interface* 6.31 (2008), pp. 187–202.

[321] Marc Toussaint, Kelsey R Allen, Kevin A Smith, and Joshua B Tenenbaum. "Differentiable Physics and Stable Modes for Tool-Use and Manipulation Planning". In: *Robotics: Science and Systems* (2018), pp. 1–9.

[322] Eric Tzeng, Coline Devin, Judy Hoffman, Chelsea Finn, Xingchao Peng, Sergey Levine, Kate Saenko, and Trevor Darrell. "Towards Adapting Deep Visuomotor Representations from Simulated to Real Environments". In: *CoRR* abs/1511.07111 (2015). arXiv: 1511.07111. URL: http://arxiv.org/abs/1511.07111.

[323] Agricultural Research Service U.S. Department of Agriculture. *FoodData Central*. 2019. URL: fdc.nal.usda.gov.

[324] Kiwon Um, Yun Fei, Robert Brand, Philipp Holl, and Nils Thuerey. "Solver-in-the-Loop: Learning from Differentiable Physics to Interact with Iterative PDE-Solvers". In: *arXiv preprint arXiv:2007.00016* (2020).

[325] P.O. Vandanjon, M. Gautier, and P. Desbats. "Identification of robots inertial parameters by means of spectrum analysis". In: *Proceedings of 1995 IEEE International Conference on Robotics and Automation*. Vol. 3. 1995, 3033–3038 vol.3. DOI: 10.1109/ROBOT.1995.525715.

[326] Diederik Verscheure, Inna Sharf, Herman Bruyninckx, Jan Swevers, and Joris De Schutter. "Identification of Contact Parameters from Stiff Multi-point Contact Robotic Operations". In: *The International Journal of Robotics Research* 29.4 (2010), pp. 367–385. DOI: 10.1177/0278364909336805. eprint: https://doi.org/10.1177/0278364909336805.

[327] A Visser, N Dijkshoorn, M Van Der Veen, and R Jurriaans. "Closing the gap between simulation and reality in the sensor and motion models of an autonomous AR.Drone". In: *The International Micro Air Vehicles conference* (2011).

[328] Qing Wang, Sanjeev R. Kulkarni, and Sergio Verdu. "Divergence Estimation for Multidimensional Densities Via k-Nearest-Neighbor Distances". en. In: *IEEE Transactions on Information Theory* 55.5 (2009), pp. 2392–2405. ISSN: 0018-9448. DOI: 10.1109/TIT.2009.2016060. (Visited on 05/27/2021).

[329] Stephanie Wang, Mengyuan Ding, Theodore F. Gast, Leyi Zhu, Steven Gagniere, Chenfanfu Jiang, and Joseph M. Teran. "Simulation and Visualization of Ductile Fracture with the Material Point Method". In: *Proc. ACM Comput. Graph. Interact. Tech.* 2.2 (2019).

[330] Yongbo Wang, Huapeng Wu, and Heikki Handroos. "Markov Chain Monte Carlo (MCMC) methods for parameter estimation of a novel hybrid redundant robot". en. In: *Fusion Engineering and Design*. Proceedings of the 26th Symposium of Fusion Technology (SOFT-26) 86.9 (2011), pp. 1863–1867. ISSN: 0920-3796. DOI: 10.1016/j.fusengdes.2011.01.062. (Visited on 02/02/2021).

[331] Yuting Wang. "Virtual Node Algorithms for Simulating and Cutting Deformable Solids". PhD thesis. UCLA, 2014.

[332] Max Welling and Yee Whye Teh. "Bayesian learning via stochastic gradient Langevin dynamics". In: *Proceedings of the 28th International Conference on International Conference on Machine Learning*. ICML'11. Madison, WI, USA: Omnipress, 2011, pp. 681–688. ISBN: 978-1-4503-0619-5. (Visited on 06/18/2021).

[333] Turner Whitted. "An Improved Illumination Model for Shaded Display". In: *Commun. ACM* 23.6 (1980), pp. 343–349. ISSN: 0001-0782. DOI: 10.1145/358876.358882.

[334] L. Wijayarathne, Q. Sima, Z. Zhou, Y. Zhao, and F. L. Hammond. "Simultaneous Trajectory Optimization and Force Control with Soft Contact Mechanics". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020, pp. 3164–3171.

[335] 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*. 2016, pp. 1433–1440. DOI: 10.1109/ICRA.2016.7487277.

[336] Grady Williams, Nolan Wagener, Brian Goldfain, Paul Drews, James M. Rehg, Byron Boots, and Evangelos A. Theodorou. "Information theoretic MPC for model-based reinforcement learning". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 1714–1721. DOI: 10.1109/ICRA.2017.7989202.

[337] Joshuah Wolper, Yunuo Chen, Minchen Li, Yu Fang, Ziyin Qu, Jiecong Lu, Meggie Cheng, and Chenfanfu Jiang. "AnisoMPM: Animating Anisotropic Damage Mechanics". In: *ACM Trans. Graph.* 39.4 (2020).

[338] Joshuah Wolper, Yu Fang, Minchen Li, Jiecong Lu, Ming Gao, and Chenfanfu Jiang. "CD-MPM: Continuum Damage Material Point Methods for Dynamic Fracture Animation". In: *ACM Trans. Graph.* 38.4 (2019).

[339] C.T. Wu, Youcai Wu, John E. Crawford, and Joseph M. Magallanes. "Three-dimensional concrete impact and penetration simulations using the smoothed particle Galerkin method". In: *International Journal of Impact Engineering* 106 (2017), pp. 1–17.

[340] CT Wu, Y Guo, and W Hu. "An introduction to the LS-DYNA smoothed particle Galerkin method for severe deformation and failure analyses in solids". In: *International LS-DYNA Users Conference*. 2014, pp. 1–20.

[341] Jiajun Wu, Joseph J Lim, Hongyi Zhang, Joshua B Tenenbaum, and William T Freeman. "Physics 101: Learning physical object properties from unlabeled videos". In: *British Machine Vision Conference*. 2016.

[342] Jiajun Wu, Erika Lu, Pushmeet Kohli, Bill Freeman, and Josh Tenenbaum. "Learning to See Physics via Visual De-animation". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., 2017, pp. 153–164. URL: http://papers.nips.cc/paper/6620-learning-to-see-physics-via-visual-de-animation.pdf.

[343] Jiajun Wu, Ilker Yildirim, Joseph J. Lim, William T. Freeman, and Joshua B. Tenenbaum. "Galileo: Perceiving Physical Object Properties by Integrating a Physics Engine with Deep Learning". In: *Advances in Neural Information Processing Systems*. Montreal, Canada: MIT Press, 2015, pp. 127–135. URL: http://dl.acm.org/citation.cfm?id=2969239.2969254.

[344] Jun Wu, Rüdiger Westermann, and Christian Dick. "A survey of physically based simulation of cuts in deformable bodies". In: *Computer Graphics Forum* 34.6 (2015), pp. 161–187.

[345] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. *Detectron2*. https://github.com/facebookresearch/detectron2. 2019.

[346] Kai M Wurm, Armin Hornung, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. "OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems". In: *ICRA 2010 workshop on best practice in 3D perception and modeling for mobile manipulation*. Vol. 2. 2010.

[347] Jie Xu, Tao Chen, Lara Zlokapa, Michael Foshey, Wojciech Matusik, Shinjiro Sueda, and Pulkit Agrawal. "An End-to-End Differentiable Framework for Contact-Aware Robot Design". In: *Proceedings of Robotics: Science and Systems*. Virtual, 2021. DOI: 10.15607/RSS.2021.XVII.008.

[348] Zhenjia Xu, Jiajun Wu, Andy Zeng, Joshua B Tenenbaum, and Shuran Song. "DensePhysNet: Learning Dense Physical Object Representations via Multi-step Dynamic Interactions". In: *Robotics: Science and Systems* (2019).

[349] Akihiko Yamaguchi and Christopher G Atkeson. “Neural networks and differential dynamic programming for reinforcement learning problems”. In: *International Conference on Robotics and Automation*. IEEE. 2016, pp. 5434–5441.

[350] Kuan-Ting Yu, Maria Bauza, Nima Fazeli, and Alberto Rodriguez. “More than a million ways to be pushed. a high-fidelity experimental dataset of planar pushing”. In: *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2016, pp. 30–37.

[351] Wenhao Yu, Jie Tan, C Karen Liu, and Greg Turk. “Preparing for the unknown: Learning a universal policy with online system identification”. In: *Robotics: Science and Systems (RSS)* (2017).

[352] Cem Yuksel. “Sample Elimination for Generating Poisson Disk Sample Sets”. In: *Computer Graphics Forum* 34.2 (2015), pp. 25–32. ISSN: 1467-8659. DOI: 10.1111/cgf.12538.

[353] Andy Zeng, Shuran Song, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. “TossingBot: Learning to Throw Arbitrary Objects with Residual Physics”. In: (2019).

[354] T. Zhang, G. Kahn, S. Levine, and P. Abbeel. “Learning deep control policies for autonomous aerial vehicles with MPC-guided policy search”. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 2016, pp. 528–535. DOI: 10.1109/ICRA.2016.7487175.

[355] Shaojun Zhu, Andrew Kimmel, Kostas E. Bekris, and Abdeslam Boularias. “Fast Model Identification via Physics Engines for Data-Efficient Policy Search”. In: *International Joint Conferences on Artificial Intelligence*. 2018.

[356] Yuke Zhu, Ziyu Wang, Josh Merel, Andrei Rusu, Tom Erez, Serkan Cabi, Saran Tunyasuvunakool, János Kramár, Raia Hadsell, Nando de Freitas, and Nicolas Heess. “Reinforcement and Imitation Learning for Diverse Visuomotor Skills”. In: *Proceedings of Robotics: Science and Systems*. Pittsburgh, Pennsylvania, 2018. DOI: 10.15607/RSS.2018.XIV.009.