Transfer-Aware Kernels, Priors and Latent Spaces from Simulation to Real Robots

RIKA ANTONOVA

Doctoral Thesis
Stockholm, Sweden 2020

Division of Robotics, Perception and Learning
School of Electrical Engineering and Computer Science
TRITA-EECS-AVL-2020:54 KTH Royal Institute of Technology
ISBN 978-91-7873-669-0 SE-100 44 Stockholm, Sweden

Public defense:
Friday, November 20, 2020, 14.00
F3, KTH, Lindstedtsvägen 26, 114 28 Stockholm
Subject: Computer Science. Specialisation: Robotics, Perception and Learning.

Tryck: Universitetsservice US AB

Abstract

Consider challenging sim-to-real cases lacking high-fidelity simulators and allowing only 10-20 hardware trials. This work shows that even imprecise simulation can be beneficial if used to build transfer-aware representations.

First, the thesis introduces an informed kernel that embeds the space of simulated trajectories into a lower-dimensional space of latent paths. It uses a sequential variational autoencoder (sVAE) to handle large-scale training from simulated data. Its modular design enables quick adaptation when used for Bayesian optimization (BO) on hardware. The thesis and the included publications demonstrate that this approach works for different areas of robotics: locomotion and manipulation. Furthermore, a variant of BO that ensures recovery from negative transfer when using corrupted kernels is introduced. An application to task-oriented grasping validates its performance on hardware.

For the case of parametric learning, simulators can serve as priors or regularizers. This work describes how to use simulation to regularize a VAE's decoder to bind the VAE's latent space to simulator parameter posterior. With that, training on a small number of real trajectories can quickly shift the posterior to reflect reality. The included publication demonstrates that this approach can also help reinforcement learning (RL) quickly overcome the sim-to-real gap on a manipulation task on hardware.

A longer-term vision is to shape latent spaces without needing to mandate a particular simulation scenario. A first step is to learn general relations that hold on sequences of states from a set of related domains. This work introduces a unifying mathematical formulation for learning independent analytic relations. Relations are learned from source domains, then used to help structure the latent space when learning on target domains. This formulation enables a more general, flexible and principled way of shaping the latent space. It formalizes the notion of learning independent relations, without imposing restrictive simplifying assumptions or requiring domain-specific information. This work presents mathematical properties, concrete algorithms and experimental validation of successful learning and transfer of latent relations.

Sammanfattning

Betänk komplicerade fall av simulering-till-verklighet där det saknas simulatorer med hög precision och endast 10-20 hårdvaruförsök tillåts. Detta arbete visar att även oprecis simulering kan vara till nytta i dessa fall, om det används för att skapa överföringsbara representationer.

Avhandlingen introducerar först en informerad kärna som bäddar in rummet av simulerade trajektorier i ett lågdimensionellt rum med latenta banor. Denna använder en så kallad sekventiell variational autoencoder (sVAE) för att hantera storskalig träning utifrån simulerade data. Dess modulära design medför snabb anpassning till den nya domänen då den används för Bayesiansk optimering (BO) på verklig hårdvara. Avhandlingen och de inkluderade publikationerna visar att denna metod fungerar för flera olika områden inom robotik: rörelse och manipulation av objekt. Dessutom introduceras en variant av BO som garanterar återhämtning från negativ överföring om korrupta kärnor används. En tillämpning inom uppgiftsanpassade handgrepp bekräftar metodens prestanda på hårdvara.

När det gäller parametrisk inlärning, kan simulatorer tjäna som apriorifördelningar eller regulariserare. Detta arbete beskriver hur man kan använda simulering för att regularisera en VAEs avkodare för att koppla ihop det latenta VAE rummet till simuleringsparametrarnas aposteriorifördelning. I och med detta kan träning på ett litet antal verkliga banor snabbt anpassa aposteriorifördelningen till att återspegla verkligheten. Den inkluderade publikationen demonstrerar att detta tillvägagångssätt också kan hjälpa så kallad förstärkningsinlärning (RL) att snabbt överbrygga gapet mellan simulering och verklighet för en manipulationsuppgift på hårdvara.

En långsiktig vision är att skapa latenta rum utan att behöva förutsätta ett specifikt simuleringsscenario. Ett första steg är att lära in generella relationer som håller för sekvenser av tillstånd i en mängd angränsande domäner. Detta arbete introducerar en enhetlig matematisk formulering för att lära in oberoende analytiska relationer. Relationerna lärs in från källdomäner och används sedan för att strukturera det latenta rummet under inlärning i måldomänen. Denna formulering medger ett mer generellt, flexibelt och principiellt sätt att skapa det latenta rummet. Det formaliserar idén om inlärning av oberoende relationer utan att påtvinga begränsande antaganden eller krav på domänspecifik information. Detta arbete presenterar matematiska egenskaper, konkreta algoritmer och experimentell utvärdering av framgångsrik träning och överföring av latenta relationer.

Thanks to Ylva Jansson and Mårten Björkman for translating the abstract to Swedish.

To my father, Геннадий Антонов
твоя улыбка осталась

Acknowledgments

To *Danica Kragic*:
Thank you very much for advising, inspiring & supporting me at KTH.

To *Silvia Cruciani, Mia Kokic, Johannes Stork, Martin Hwasser, Anastasiia Varava & others@RPL*:
Thanks for collaborations at KTH.

To *Akshara Rai*:
Thank you for being an amazing long-term collaborator, despite different continents & time zones, always enthusiastic about our next robot learning adventure together!

To *Thomas Schön*:
Thanks for welcoming me to your group meetings at Uppsala.

To *Emma Brunskill & Chris Atkeson*:
Thanks for the best introduction to the world of RL & Robotics at CMU.

To *Ylva Jansson, Ioanna Mitsioni Diogo Almeida, João· Carvalho*:
Thanks for making my days in Sweden a bit warmer and brighter.

To *Matt Kretchmar & Jessen Havill, Ravi Sundaram & Guevara Noubir*:
Thanks for helping make my first steps in undergraduate and graduate CS.

To *Sam Devlin & Katja Hofmann, Cheng Zhang & Yingzhen Li, Kamil Ciosek & Sebastian Tschiatschek*:
Thanks for insightful discussions during my visit to MSR Cambridge and for your further support.

To *Sep Kamvar, Uygar Oztekin & search ranking team at Google; Ray Smith & OCR+Books+ StreetView teams*:
Thanks for the best work environment an engineer could dream of; these memories gave me positive energy to last a lifetime.

To *Maksim Maydanskiy*:
Thanks for all that math :-]

To *Irina Antonova*:
Thank you for being a wonderful mother – it is the hardest role!

To *Carlos Ponguillo*:
Thanks for supporting my initial East → West journey & further help.

To the grading committee:
Marc Deisenroth,
Joelle Pineau,
Sebastian Trimpe:
Thanks for examining the defense.

To *Jens Kober*:
Thanks for taking part in the defense as the opponent.

List of Papers

This thesis is based on the following papers:

Bayesian Optimization in Variational Latent Spaces with Dynamic Compression.
R. Antonova[1], A. Rai[1], T. Li, D. Kragic
In Conference on Robot Learning (CoRL),
Proceedings of Machine Learning Research (PMLR) 100:456-465, 2019.

Using Simulation to Improve Sample-Efficiency of Bayesian Optimization for Bipedal Robots.
A. Rai[1], **R. Antonova**[1], F. Meier, C. Atkeson.
In Journal of Machine Learning Research (JMLR), PMLR 20(49):1-24, 2019.

Deep Kernels for Optimizing Locomotion Controllers.
R. Antonova[1], A. Rai[1], C. Atkeson.
In Conference on Robot Learning (CoRL), PMLR 78:47-56, 2017.

Bayesian Optimization Using Domain Knowledge on the ATRIAS Biped.
A. Rai[1], **R. Antonova**[1], S. Song, W. Martin, H. Geyer, C. Atkeson.
In IEEE International Conference on Robotics and Automation (ICRA), 2018.

Global Search with Bernoulli Alternation Kernel for Task-oriented Grasping Informed by Simulation.
R. Antonova[1], M. Kokic[1], J. A. Stork, D. Kragic.
In Conference on Robot Learning (CoRL), PMLR 87:641-650, 2018.

Variational Auto-Regularized Alignment for Sim-to-Real Control.
M. Hwasser, D. Kragic, **R. Antonova**.
In IEEE International Conference on Robotics and Automation (ICRA), 2020.

Analytic Manifold Learning: Unifying and Evaluating Representations for Continuous Control.
R. Antonova, M. Maydanskiy, D. Kragic, S. Devlin, K. Hofmann.
In arXiv:2006.08718, 2020.

[1]Equal contribution

The following works have also been produced during the PhD study period (but are not included in this thesis):

Benchmarking Bimanual Cloth Manipulation.
I. Garcia-Camacho, M. Lippi, M.C. Welle, H. Yin, **R. Antonova**, A. Varava, J. Borras, C. Torras, A. Marino, G. Alenya, D. Kragic.
In IEEE Robotics and Automation Letters (RA-L) 2020.

Unlocking the potential of simulators: Design with RL in mind.
R. Antonova[1], S. Cruciani[1].
Conference on Reinforcement Learning and Decision Making (RLDM) 2017.
Presentation at RLDM was based on results from the following work:
Reinforcement Learning for Pivoting Task.
R. Antonova[1], S. Cruciani[1], C. Smith, D Kragic.
In arXiv:1703.00472, 2017.

Statement of Contributions

Bayesian Optimization in Variational Latent Spaces with Dynamic Compression.
R. Antonova[1], A. Rai[1], T. Li, D. Kragic, CoRL 2019

Rika Antonova: formulated the algorithm for embedding trajectories into latent distributions; constructed sequential VAE architecture based on time-convolutions and experimented with alternative NN architectures; set up training and simulation experiments for locomotion and manipulation; set up and ran hardware experiments for manipulation

Akshara Rai: provided Daisy hexapod simulator and controllers; set up and ran hardware experiments for locomotion; provided access to large-scale compute infrastructure for simulation experiments

T. Li. helped with collecting hardware experiment data for locomotion

D. Kragic: gave advice for manipulation tasks and literature

Deep Kernels for Optimizing Locomotion Controllers.
R. Antonova[1], A. Rai[1], C. Atkeson. CoRL 2017

Bayesian Optimization Using Domain Knowledge on the ATRIAS Biped.
A. Rai[1], R. Antonova[1], S. Song, W. Martin, H. Geyer, C. Atkeson. ICRA 2018

Using Simulation to Improve Sample-Efficiency of Bayesian Optimization for Bipedal Robots.
A. Rai[1], R. Antonova[1], F. Meier, C. Atkeson. JMLR 2019

Rika Antonova: formulated and constructed NN-based kernels, conducted simulation experiments; derived equations for interpreting mismatch as part of the kernel; provided BO background and literature; wrote approach descriptions and justification from the learning perspective; compared with sparse GP baselines

Akshara Rai: set up ATRIAS simulator and controller; did hardware experiments; constructed domain-specific DoG kernels; compared with IT&E approach; provided locomotion literature; experimented with robustness to incorrect dynamics

S. Song: helped with Neuromuscular controller in simulation

W. Martin: helped with setting up and repairing ATRIAS hardware

F. Meier: gave advice for Akshara at MPI; proofread JMLR draft, gave suggestions on organization & figures
H. Geyer & C. Atkeson: advising for Akshara at CMU

Global Search with Bernoulli Alternation Kernel for Task-oriented Grasping Informed by Simulation.
R. Antonova[1], M. Kokic[1], J. A. Stork, D. Kragic. CoRL 2018

Rika Antonova: formulated the BO-BAK algorithm and provided its analysis; constructed BO kernel from CNN; provided BO background, theory and evaluation; set up and ran hardware experiments for BO; demonstrated BO-BAK success on hardware for challenging objects and showed recovery with severely degraded kernels

Mia Kokic: set up CNN architecture and training with grasp stability and task suitability scores; converted Kinect input to voxel grid representation; ran simulation and hardware experiments for the 'top-k' approach based on CNN (without BO)

J. A. Stork: helped with paper writing and organization

D. Kragic: helped with paper organization; advising for Rika & Mia

Variational Auto-Regularized Alignment for Sim-to-Real Control.
M. Hwasser, D. Kragic, R. Antonova. ICRA 2020

Rika Antonova: formulated the initial version of DET2STOC algorithm; implemented comparisons with likelihood-free methods; formulated hardware tasks that allowed investigating recovery from sim-to-real mismatch; wrote ICRA2020 submission
[& supervised Martin's MS thesis work that included initial implementation and simulation experiments]

Martin Hwasser: implemented DET2STOC and developed an effective training procedure; created advanced simulation environments for evaluation; set up comparisons with CVAE baseline; set up and ran hardware experiments

D. Kragic: advising for Rika

Analytic Manifold Learning: Unifying and Evaluating Representations for Continuous Control.
R. Antonova, M. Maydanskiy, D. Kragic, S. Devlin, K. Hofmann. arXiv 2020

Rika Antonova: formulated the unifying approach as a generalization of learning with auxiliary losses; developed a benchmark suite for analyzing VAEs training on non-stationary data stream; constructed NN-based algorithm using the mathematical formalism; developed VI-based algorithm suitable for sim-to-real and transfer learning settings

M. Maydanskiy: suggested formalism from abstract algebra and analytic & differential geometry for rigorous definitions of independence; provided theorem proofs; helped formalize non-triviality of learned relations

D. Kragic: gave suggestions for robotics tasks and feedback on paper organization

S. Devlin, K. Hofmann: hosted Rika at MSR Cambridge (supported the start of the project); provided Azure compute resources, detailed discussions about paper writing & help with communicating the work to the learning community

Contents

Part I

Main

Chapter 1

Introduction: Towards Transfer-aware Methods

"All models are wrong, but some are useful" [1]. In robotics, precise scenario-specific simulation and models have been widely used since the inception of the field. However, leveraging imprecise general-purpose simulators is an open problem. We can consider this problem in the context transfer learning, with simulation as the source domain and real-world hardware as the target domain. While some approaches from the general field of transfer learning can be applicable, in the context of robotics we face a unique combination of challenges. Hence, the term *sim-to-real* has been established to concisely express this combination. The main challenges are: the need for data efficiency when training on hardware and the need to close the *sim-to-real gap*. Prior work that aimed to tackle these challenges most often focused on only one aspect at a time. In contrast, the work presented in this thesis offers a unified view and proposes a set of *transfer-aware* methods that are both data-efficient and non-restrictive in terms of simulation quality needed for successful sim-to-real transfer.

This work defines a *transfer-aware* paradigm for constructing sim-to-real algorithms. The core idea of this paradigm is that training on a source domain (simulation) should be done with the foresight of the need to adjust the resulting structures/representations on the target domain (reality) with only a few hardware trials/episodes. The algorithms presented in this thesis demonstrate that this transfer-aware paradigm can be used to construct methods that leverage simulation in various ways: by constructing informed kernels; by using simulators as regularizers; by learning to describe a simulation-induced data manifold as a set of independent relations, which can be imposed to structure the latent space during training on target (real) data. Hence, the overall result is the toolbox of sim-to-real methods, where each roboticist could hope to find a tool that fits their needs and preferences. For those who prefer to use structured parametric controllers: the proposed kernel-based methods for Bayesian optimization (BO) would be the best fit (Chapter 3). For those favoring model-free deep reinforcement learning (RL) and variational inference (VI): the proposed approach to use simulation as a regularizer would help obtaining flexible posteriors over simulation parameters and help deep RL recover from sim-to-real mismatch with few hardware trials (Chapter 4). For those aiming to make minimal assumptions regarding the source domain: the proposed approach to automatically encode the latent properties of the source domain in a set of (non-linearly) independent relations would give most freedom, while helping to improve data-efficiency on a target domain (Chapter 5).

The proposed methods aim to incorporate components from different sub-fields of machine learning. Despite this variety, the unifying theme of these algorithms is that they are constructed with the goal to remove restrictive assumptions about the quality of the simulation. Representations and structures learned by these methods are designed to be quickly adjusted from few hardware samples in order to close the sim-to-real gap in a data-efficient way. For the case of BO, for example, this yields ultra-data efficient methods that can benefit from as few as 10 hardware trials/episodes. Experiments presented in this thesis (and in the included publications) show that previous work, which used generic ways to update representations/structures, could not achieve such data efficiency for modern higher-dimensional controllers and state spaces. Hence, the thesis demonstrates the need and the benefit of adhering to the transfer-aware paradigm, instead of simply hoping that making all components differentiable would be enough to handle the sim-to-real mismatch effectively.

The Structure of this Thesis

- Chapter 1 (this one) gives an overview of the thesis.
- Chapter 2 states challenges & opportunities of the sim-to-real problem, then outlines the background from the fields of machine learning relevant to this thesis.
- Chapter 3 presents the proposed kernel-based methods. First, it provides justification for using simulation-informed kernels for Gaussian processes (GPs) within the framework of Bayesian optimization (BO). Policy optimization is formulated as BO on the space of *structured parametric controllers*. Successful application to bipedal locomotion on the ATRIAS robot is summarized from [2, 3, 4]. Then, a more general domain-agnostic approach is presented: BO-SVAE-DC introduces a modular sequential variational autoencoder (sVAE) used to embed the space of simulated trajectories into a lower-dimensional space of latent paths in an unsupervised way. This yields an encoder used to construct a simulation-informed kernel. The method also allows to further compress parts of the space containing undesirable regions. Experiments (summarized from [5]) demonstrate that using the resulting kernels yields significant improvements over uninformed BO, with only 10 hardware trials to close the sim-to-real gap. The generality of this approach is demonstrated by hardware experiments in two different areas of robotics: locomotion (on HEBI Robotics Daisy hexapod) and manipulation (ABB Yumi robot). The kernels for these are built using the same sVAE architecture (same sizes and parameters of the underlying neural networks), and the same BO hyperparameters. Next, a

BO-BAK method is proposed for cases with highly imprecise or severely degraded kernels. The thesis describes hardware experiments with recovering from negative transfer, in the setting of task-oriented grasping introduced in [6].

- Chapter 4 shows how to use simulators as regularizers to infer flexible simulator parameter posteriors from few hardware trajectories. The proposed DET2STOC approach regularizes a VAE decoder to simulation, with latent space bound to a subset of simulator parameters, yielding (multimodal) parameter posteriors aligned to hardware data. Hardware experiments (summarized from [7]) on a non-prehensile task with an ABB Yumi robot show ability to help RL overcome severe sim-to-real mismatch. DET2STOC is aimed to be useful for the part of the community that favors *unstructured neural network policies*, e.g. those learned by recently popularized model-free deep RL algorithms.

- Chapter 5 first acknowledges the limitations of unsupervised approaches, such as VAEs, when handling distribution shift. This has direct negative implications for sim-to-real. To combat this, previous lines of work proposed imposing hand-constructed latent relations based on domain knowledge or algorithmic insights (e.g. expecting/ensuring continuity between consecutive latent states, consistency with a known forward or inverse model structure, etc). This thesis presents a unifying mathematical formulation for automatically learning (non-linearly) independent relations from the latent data manifold. The proposed Analytic Manifold Learning (AML) obtains analytic relations on source domains (e.g. simulation), then uses these relations to help structure the latent space when learning on target domains. Experiments (summarized from [8]) show initial success in transfer of relations learned from source domains with simple geometric shapes to target (simulated) domains that contain objects with real textures and 3D scanned meshes. The generality of AML goes beyond being useful for sim-to-real. Hence, this thesis presents the general formulation and highlights its potential for areas like continual and lifelong learning, leaving hardware sim-to-real experiments for future work.

- Chapter 6 presents conclusion and future directions. First, it summarizes the main contributions offered by the work conducted for this thesis, both from the algorithmic perspective (i.e. new concrete algorithms that the thesis describes) and from the conceptual perspective (for example: how the work and arguments from publications associated with this thesis changed the views and attitudes of the community towards themes like using simulation-informed kernels). The chapter concludes by discussing how the proposed algorithms, methods and mathematical formulations can enable further progress in more challenging sim-to-real settings, such as manipulation with deformable objects and lifelong learning.

- The above chapters constitute Part I: ‘Main’ part of the thesis. Part II contains the publications included in this thesis. These are provided in almost exactly the same form as published, with minor edits to accommodate the thesis paper size and format.

Note on the Thesis Format: Compilation vs Monograph

Formally, this thesis has the format of a ‘Compilation thesis’ (also known as ‘Cumulative thesis’, ‘Thesis by published works’, ‘Article thesis’), which is encouraged in the Nordic countries. The compilation thesis format is defined as follows: it starts with a ‘Kappa’: a summary (15-35 pages) that provides an overview of the thesis work as a whole and briefly summarizes each work included as a ‘thesis publication’. Then, the list of published (or submitted) papers is provided, along with a statement of thesis author’s contributions to each paper.

Some parts of the international community are more used to the ‘Monograph’ format and might find it challenging to evaluate the views and contributions of the thesis author if they are all embedded in the joint publications. Moreover, the compilation format encourages breadth, since publications do not necessarily build on a single algorithm/idea – this can be challenging for the readers due to lack of a unified mathematical notation and the need for a broad prior background. To address these points, I wrote this thesis in an extended format. The ‘Main’ part does start by giving an overview (the ‘Introduction’ above), but is then extended beyond what would be usually included in a summary/overview ‘Kappa’.

First: I provided the common mathematical notation, algorithmic background and literature review in Chapter 2.

Second: from the included publications, I selected a subset of approaches for which I was the main contributor of algorithmic ideas and that exemplify the principle of being ‘transfer-aware’. Chapters 3,4,5 present these algorithms. Chapters 3,4 are mostly self-contained, so the readers would only need to look at the included publications to get in-depth details. Chapter 5 gives an overview of the main ideas presented in [8], however, readers unfamiliar with abstract algebra and differential geometry would need to refer to the additional explanations in the Appendix in [8] for a better understanding.

Third: while some paragraphs in this thesis contain text from the included publications, a large part of the text is new (more appropriate for a higher-level discussion as opposed to reporting minor details). Furthermore, Chapter 3 contains several new illustrations, mathematical derivations and hardware experiments (not contained in the included publications).

Fourth: I included descriptions of some of the hardware experiments I ran in our lab at KTH. These present examples of my experimental hardware platform. However, I would not have been satisfied with only one person+platform for the validation of the proposed algorithms. Hence, I collaborated closely with other students to design experiments that validated the proposed approaches further. In the ‘Main’ part of the thesis I only briefly summarize these experiments, focusing on the results and implications rather than details (the included publications contain the details of the hardware setups and experiments).

Firth: the goal of the ‘Main’ part of this thesis is to explain how the algorithms proposed in the included publications address sim-to-real challenges from the perspective of active learning. I view sim-to-real problem from a perspective that is closer to the learning community, which seeks ways to remove dependence on hand-constructed representations, task-specific structures and assumptions. Hence, the ‘Main’ part of the thesis includes detailed analysis of the ‘active’ aspects of the learning process (e.g. Bayesian optimization on hardware, adjusting simulator posterior and reinforcement learning policies using data from real trajectories, etc). The details of offline training or domain/task-specific aspects are available in the included publications, but I do not discuss them in detail in the ‘Main’ part.

Finally: I hope that this extended format would help the audience to quickly grasp the main ideas of this thesis work, while still allowing the interested readers to find all the further details within the included publications.

Chapter 2

Background

2.1 Sim-to-Real: Problem Statement and Challenges

Generally speaking, *sim-to-real* defines a class of transfer learning problems, with simulation being the source and reality/hardware being the target domain. Methods tackling the *sim-to-real* problem aim to leverage simulation to improve efficiency of learning from real data. The need for data efficiency arises since running experiments on hardware can be costly, both in terms of time and in terms of wear-and-tear costs (e.g. research-grade hardware usually contains components that fail in case of prolonged operation).

Closing the *sim-to-real gap* is a challenging problem if we do not want to limit ourselves to utilizing only high-fidelity simulators. Medium- and low-fidelity simulators are constructed without bounds on how much simulation can deviate from reality. This implies that utilizing imprecise simulation can cause negative transfer. *Negative transfer* occurs when the use of knowledge from the source/prior domain hurts the learning progress on the target domain. There have been recent (but limited) attempts to examine this notion formally in the supervised deep learning [9] and the RL communities [10]. However, it is challenging to formalize and guard against negative transfer for transfer learning in general, as well as for sim-to-real in particular. Hence, the vast majority of prior works do not discuss the possibility of negative transfer and don't explicitly test what happens as the quality of the simulator degrades. They side-step this issue by simply assuming that the source and target domains are 'related enough' such that incorporating information from the source domain in any form is ultimately useful. This thesis work does not make such assumption, and instead conducts simulation and hardware experiments that aim to shed light on this important issue.

In recent years, sim-to-real methods have been experimentally shown to work on tasks in various areas of robotics, for example motion planning [11], navigation [12], locomotion [13, 14, 15] and manipulation [16, 17]. Early works either relied on designing domain-specific features and controllers [13] or utilized basic techniques, such as domain randomization [11, 14, 16, 18]. A scalable example of the latter demonstrated solving an advanced in-hand manipulation task with a model-free deep RL policy trained from high-dimensional observations [19]. The downside of basic domain randomization is that it can yield suboptimal policies. Moreover, randomizing too aggressively leads to failing to solve advanced tasks even in simulation, while randomizing too little leads to learning policies that fail on hardware. Hence, there is still a debate within the community as to when sim-to-real using general-purpose simulators is warranted [20]. The basic argument that could be put forth is that traditional approaches, such as system identification and building explicit models from hardware data, should be used when such modeling is tractable, while sim-to-real should be used when learning a precise and concise model from real data is intractable. In such cases we could hope to benefit from domain knowledge, but this knowledge could be in a 'black-box' form of a general-purpose simulator.

Building on the initial success of domain randomization, one line of more recent approaches aims to adapt simulation parameter posteriors using hardware data. A scalable version of this demonstrated that with a highly parallelized simulation it is possible to benefits from as little as 10 hardware trajectories [17]. However, this work was limited to learning unimodal simulation parameter posteriors. More recent developments proposed approaches that could yield multimodal mixture posteriors [21], though initially without demonstrating hardware results. The work presented in Chapter 4 of this thesis proposes an approach capable of producing mixture posteriors, shows ability to align with reality using 10 hardware trials and helps an RL policy to close a large sim-to-real gap [7].

Despite succeeding on some domains, the approach of learning simulator parameter posteriors is not always tractable. This could be either because, for advanced scenarios, the simulator might fail to come close to reality with *any* setting of simulation parameters or because finding a well-performing parameter distribution requires a prohibitive amount of hardware data. In such cases, more direct methods are needed to extract domain knowledge from simulation in flexible ways. One such family of methods could be obtained by considering data-efficient approaches like Bayesian optimization (BO). BO could be used to search for well-performing controller parameters directly, instead of optimizing a model or a simulator to align with reality. However, BO 'from scratch' is still not data efficient enough for optimizing higher-dimensional controllers or solving advanced robotics tasks. Hence, this thesis argues for constructing informed kernels from simulation to enhance data efficiency and representational power of BO (Chapter 3).

2.2 Bayesian Optimization

Background and Mathematical Formulation

Bayesian optimization (BO) is a framework for online, black-box, gradient-free global search; [22] and [23] provide a comprehensive introduction. The problem of optimizing controllers can be interpreted as finding controller parameters $\boldsymbol{x}^*$

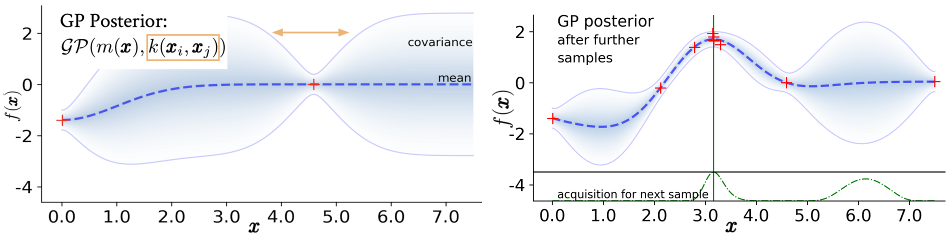


Figure 2.1: Illustration of Bayesian optimization for an example 1D problem. The objective is to find $\boldsymbol{x}$ with maximal $f(\boldsymbol{x})$. Gaussian Process (GP) models the posterior for $f(\boldsymbol{x})$. GP has two main components: the mean function $m(\boldsymbol{x})$ and kernel (covariance) function $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$. The kernel determines how far the influence of the previously evaluated points (red pluses) extends. Left: GP posterior after the first two samples. Right: GP posterior after further samples. The acquisition function (dashed green line on the bottom) uses posterior mean and covariance to balance exploration and exploitation. BO achieves data efficiency by sampling more points close to the optimum; BO explores in the rest of the search space only enough to ensure that a better solution is unlikely (not aiming to decrease uncertainty uniformly across the search space).

that optimize some objective function $f(\boldsymbol{x})$. In the context of this thesis work, $\boldsymbol{x}$ represents a vector that contains parameters of a pre-structured policy. For brevity, 'controller $\boldsymbol{x}$' is used denote a controller with parameters $\boldsymbol{x}$. The objective $f(\boldsymbol{x})$ is a function of the trajectory induced by controller parameters $\boldsymbol{x}$; it expresses how well a controller is able to solve a given task.

BO can be used to find controller $\boldsymbol{x}^*$ that maximizes an objective function f:

$$f(\boldsymbol{x}^*) = \max_{\boldsymbol{x}} f(\boldsymbol{x})$$

Some works use costs instead of objective/reward functions. The optimization process is analogous in such cases: the same code can be used with just a sign of the cost negated to do objective maximization instead of cost minimization.

BO is initialized with a prior that expresses *a priori* uncertainty over $f(\boldsymbol{x})$ and helps keep track of the *posterior* of f. A widely used representation for the objective function f is a Gaussian process (GP):

$$f(\boldsymbol{x}) \sim \mathcal{GP}(m(\boldsymbol{x}), k(\boldsymbol{x}_i, \boldsymbol{x}_j))$$

The prior mean function $m(\cdot)$ is set to zero when no domain-specific knowledge is given. The kernel function $k(\cdot, \cdot)$ encodes similarity between inputs. If $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is large for inputs $\boldsymbol{x}_i, \boldsymbol{x}_j$, then $f(\boldsymbol{x}_i)$ strongly influences $f(\boldsymbol{x}_j)$. One of the most widely used kernel functions is the Squared Exponential (SE):

$$k_{SE}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sigma_k^2 \exp\big(-\tfrac{1}{2}(\boldsymbol{x}_i - \boldsymbol{x}_j)^T \operatorname{diag}(\boldsymbol{\ell})^{-2}(\boldsymbol{x}_i - \boldsymbol{x}_j)\big), \tag{1}$$

where σ_k^2, $\boldsymbol{\ell}$ are signal variance and a vector of length scales respectively. σ_k^2, $\boldsymbol{\ell}$ are referred to as 'hyperparameters' and can be optimized automatically by maximizing GP marginal likelihood (see [22], Section V-A).

The posterior mean and covariance for any point $\boldsymbol{x}_*$ can be computed with:

$$\mathbb{E}[f(\boldsymbol{x}_*)] = \bar{f}_* = \boldsymbol{k}_*^T(K + \sigma_n^2 I)^{-1}\boldsymbol{y} \tag{2}$$

$$Var[f(\boldsymbol{x}_*)] = \mathbb{V}[f_*] = k(\boldsymbol{x}_*, \boldsymbol{x}_*) - \boldsymbol{k}_*^T(K + \sigma_n^2 I)^{-1}\boldsymbol{k}_* \tag{3}$$

Here: $K = K(X, X)$, meaning K is a matrix $\in \mathbb{R}^{n \times n}$ that has $k(x_i, x_j)$ as ij-th entries and is computed using all pairs of points evaluated in trials/episodes $\{1, ..., n\}$ that have been completed so far; $\boldsymbol{k}_* = k(X, \boldsymbol{x}_*)$ is a vector $\in \mathbb{R}^n$ that captures the similarity between a given point $\boldsymbol{x}_*$ and each of the n points from the completed trials.

To propose the point/controller $\boldsymbol{x}$ that should be evaluated next, BO optimizes an auxiliary function called *acquisition function.* Two most commonly used options for the acquisition function are: Expected Improvement (EI) [24] and Upper Confidence Bound (UCB) [25]. While some works report results being sensitive to the choice of the acquisition function, the algorithms presented in this thesis showed similar performance with EI and UCB. Hence, UCB was used in the most recent part of the work, since UCB is intuitive to understand and has regret bound guarantees [25]. The acquisition function uses GP posterior that incorporates all the data available so far to balance exploration vs exploitation. It selects points for which the posterior estimate of the objective f is promising, taking into account both posterior mean and (co)variance. For example, UCB acquisition function selects the next $\boldsymbol{x}$ using:

$$\boldsymbol{x}_{UCB} = \arg\max_{\boldsymbol{x} \in \mathcal{X}} \mathbb{E}[f(\boldsymbol{x})] + \beta\ Var[f(\boldsymbol{x})]$$

β can be determined by theoretical considerations to ensure theoretical regret bounds [25] or could be chosen as higher/lower if more/less exploration is desired for a particular domain. See Figure 2.1 for basic GP posterior and acquisition function visualizations.

BO ensures data efficiency by keeping track of uncertainty across the search space and leaving unpromising parts of the search space under-explored. This approach is well-suited for cases with a small budget of hardware trials (<100). One straightforward way to incorporate simulation information could be to add 'fake' prior points obtained from simulated trials to the GP. However, in this case computational complexity of GPs may be a deterrent. Computing GP posterior mean, covariance and marginal likelihood is usually accomplished with algorithms that involve Cholesky factorization, which has the computational complexity of $\frac{n^3}{6}$. The most expensive operation involved is the matrix inversion $(K + \sigma_n^2 I)^{-1}$, which has a smaller asymptotic constant ($n^{2.373}$ with some recent methods). However, approaches utilizing Cholesky factorization are considered more numerically stable, hence are commonly used in GP libraries [26, 27]. See [28] for further details.

To improve scalability of GPs, a number of sparse approximations have been proposed. Inducing points methods use a small set of m inducing points instead of forming a full covariance matrix [29, 30]. Such methods can reduce the computational complexity to $O(nm^2)$. Some versions use approximate inference to compute approximations to the posterior [31, 32]. Such methods can scale to 10K+ points, and experiments in Chapter 3 demonstrate they can be useful for populating GPs with prior ‘fake’ points from simulation. However, Chapter 3 also shows that this prior-based way to utilize simulation is not robust in case of using low-fidelity simulations.

Gaining intuition about GP mean is easier than understanding the effects of the kernel. Nonetheless, it is especially important to appreciate that kernel choices have a large impact on BO, since they shape the search space by imposing a similarity metric on it. The SE kernel from Equation 1 belongs to a broader class of Matérn kernels, which in general have more free parameters. One common parameter choice yields Matérn$_{5/2}$: $k_{\text{Matérn}_{5/2}}(\boldsymbol{r}) = \big(1 + \frac{\sqrt{5}\boldsymbol{r}}{\boldsymbol{\ell}} + \frac{5\boldsymbol{r}^2}{3\boldsymbol{\ell}^2}\big) \exp\big(-\frac{\sqrt{5}\boldsymbol{r}}{\boldsymbol{\ell}}\big)$. In some cases carefully choosing kernel parameters improves performance of BO [33]. However, manually constructed domain-informed kernels can easily out-perform even well-tuned Matérn kernels [13]. SE and Matérn kernels are stationary: $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ depend only on $r = \boldsymbol{x}_i - \boldsymbol{x}_j \ \forall \boldsymbol{x}_{i,j}$, and not on individual $\boldsymbol{x}_i, \boldsymbol{x}_j$. Stationarity allows avoiding commitment to domain-specific assumptions, which helps generality, but can be detrimental to data efficiency and flexibility. This is because all regions of the search space have to be treated in an equivalent way. Chapter 4 of [28] provides a number of other choices for the kernel functions. Some uninformed kernels can improve BO if their assumptions match the needs of a particular domain, e.g. periodic kernels for domains with cyclic patterns. However, such choice requires domain knowledge about the properties of the target domain/task. Chapter 3 of this thesis shows that the need for such expertise can be replaced by leveraging general-purpose simulation to build informed kernels automatically.

Prior Work in Bayesian Optimization

Gaussian Processes (GPs) have been widely used in robotics for learning models, for example for reinforcement learning for control [34, 35, 36], motion planing [37], manipulation [38, 39] and active perception [40, 41]. GPs have also been used as key structures in active learning algorithms, such as Bayesian optimization. BO without the use of simulation has shown initial success in a number of areas of robotics. For example, BO for locomotion has been shown to succeed for snake robots [42], AIBO quadrupeds [43], and hexapods [13]. BO and continuous Multi-armed bandit approaches have been useful for grasping: a problem where vision alone is usually not sufficient to inform about important inertial and frictional properties of objects [44, 45, 46]. However, the above results have been achieved either with lower-dimensional controllers (as in the case for locomotion) or with simple objectives, e.g. grasping an object in any way to avoid slips, without considering how the object would be used for a subsequent task. Hence, further research was warranted to improve scalability, data efficiency, and flexibility of BO by using further domain knowledge.

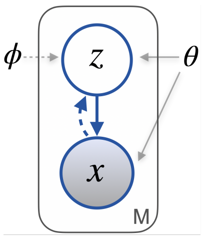
Domain knowledge from simulation can be incorporated into Gaussian Process prior used in BO, for example as done in [13]. However, as will be shown in Chapter 3, prior-based approaches require carefully tuning the influence of points added from simulation vs hardware points, especially for the case of imprecise simulators. When multi-fidelity simulators are available, approaches such as [47, 48] can be used to trade off computation vs simulation accuracy to select the fidelity level or for the next trial/evaluation (with real world being the highest-fidelity source). In contrast, this thesis considers a different setting: a single simulator (with an unknown fidelity level) and an extremely small number of experiments on a real robot. Hence the work in this thesis benefits from ability to take a two step approach: learning kernel transforms in the 1st stage, then running BO on a real robot in the 2nd stage.

Recently, several works proposed using neural networks (NNs) within GP kernels [49], [50]. This offered improvements for some challenging aspects that arise in robotics, e.g. [50] showed ability to successfully handle discontinuous objectives. However, these approaches did not address the problem of incorporating simulation directly. These prior works aimed to jointly update the NN by propagating gradients from the GP updates. This could succeed with ample data on the target domain. However, these prior works did not discuss the challenges of updating such kernels effectively with a small of data available from hardware BO experiments.

Behavior Based Kernel (BBK) introduced by [51] aimed to enhance GP kernels with trajectory information. BBK computes an estimate of Jensen-Shannon distance (a symmetrized version of KL divergence) between trajectories induced by two controllers, then uses this estimate as a kernel distance metric. However, obtaining such estimates requires obtaining samples for each controller $\boldsymbol{x}_i, \boldsymbol{x}_j$ whenever $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is needed. This is impractical, since it requires an evaluation of every controller considered when doing the internal BO computations to optimize the acquisition function. The authors propose using BBK in conjunction with a model-based method. However, as discussed in the previous section, we are particularly interested in challenging cases when building an accurate model from hardware data is intractable, either due to a limited budget of hardware trials or because of the complexity of the problem, e.g. contact-rich tasks, higher-dimensional controllers, etc.

An alternative approach that aims to benefit from simulated trajectories has been proposed in [13]. This approach defines a behavior metric specific to hexapod locomotion and collects an ‘elite’ set of points that perform well in simulation. The behavior metric is used to guide BO in finding walking controllers on hardware with few trials, and can even cope with damage to the robot. BO on hardware is done in this hexapod ‘behavior’ space, but it is limited to pre-selected ‘elite’ points from simulation. Hence, if an optimal point is not pre-selected, BO cannot propose it during optimization on hardware.

The work described in this thesis utilizes trajectories from simulation to build feature transforms that can be incorporated into the GP kernel. This direction is related, in part, to input space warping [52], but goes beyond simply applying a transform given in a explicit form. Instead, the central part of the work is to incorporate information from simulation while ensuring that the overall algorithm facilitates closing the sim-to-real gap. The aim is also to accomplish that in a more domain-agnostic and scalable manner than prior attempts.



prior : $p(\boldsymbol{z})$; *likelihood* : $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$
marginal likelihood : $p(\boldsymbol{x}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})d\boldsymbol{z}$
posterior : $p(\boldsymbol{z}|\boldsymbol{x}) = p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})/p(\boldsymbol{x})$
approximate variational posterior : $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$
(approximates intractable posterior $p(\boldsymbol{z}|\boldsymbol{x})$)

Figure 2.2: Left: a generic graphical model for illustrating VI principles (similar to [55]). Right: a summary of notation used in VI literature. It is common to use $\boldsymbol{\theta}, \boldsymbol{\phi}$ to denote parameters of the distributions, e.g. mean and variance vectors, or alternatively: weights of a neural network (NN) that outputs mean and variance estimates. For VAEs: $\boldsymbol{\phi}$ denotes weights of an encoder NN (that takes input data $\boldsymbol{x}$ and produces parameters of q, e.g. mean and covariance if using Gaussian distributions); $\boldsymbol{\theta}$ denotes weights of a decoder NN (that can decode a latent sample $\tilde{\boldsymbol{z}}$ obtained from $q(\cdot|\boldsymbol{x})$ into a 'reconstruction' $\hat{\boldsymbol{x}}$).

2.3 Variational Inference and VAEs

Background and Notation for Variational Autoencoders

Variational inference (VI) is a class of efficient methods for inference in graphical models. VI can be used effectively to quickly determine an approximation to a model's posterior given data/evidence. VI approaches first select a parametric family of distributions, then optimize its parameters. [53] provides an extensive introduction into VI and [54] gives a recent overview.

Consider data $\boldsymbol{x} = \{\boldsymbol{x}^{(i)}\}_{i=1}^{M}$ that consist of M iid (independent and identically distributed) samples. This constitutes the *observed* data, indicated by a circle with gray background in Figure 2.2. We assume that the data is generated by a random process that involves an *unobserved* (latent) variable $\boldsymbol{z}$, illustrated by a circle with white background in Figure 2.2. The data generation process consists of 2 steps:

- $\boldsymbol{z}^{(i)}$ is generated from a *prior* distribution $p_{\boldsymbol{\theta}^{prior}_{true}}(\boldsymbol{z})$
- $\boldsymbol{x}^{(i)}$ is generated from a conditional distribution $p_{\boldsymbol{\theta}^{lik}_{true}}(\boldsymbol{x}|\boldsymbol{z})$, called *likelihood*

It is assumed that: the $\boldsymbol{\theta}_{true}$ parameters are unknown; integrating *marginal likelihood* $p(\boldsymbol{x}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})d\boldsymbol{z}$ is intractable; finding the exact *posterior* density $p(\boldsymbol{z}|\boldsymbol{x}) = p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})/p(\boldsymbol{x})$ is also intractable.

Variational autoencoders (VAEs) [55] utilize VI principles to learn an approximate posterior $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ that serves as approximation to $p(\boldsymbol{z}|\boldsymbol{x})$ and do so in a scalable and unsupervised manner. VAEs leverage reparametrization trick (see [55, 56]) to allow learning $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ from data instead of assuming it is given or estimated separately. Hence, VAEs solve both the learning and the inference problem. $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$, $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ are parameterized by neural networks (NNs). NNs are trained using all of the available data and $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ is represented by a NN that can be used for any input point $\boldsymbol{x}^{(i)}$, which means inference is amortized. NN weights are learned via gradient ascent on the objective called *evidence lower bound (ELBO)*, which is derived below.

Maximum likelihood estimation suggests optimizing parameters by maximizing observed data likelihood (evidence), or equivalently: maximizing data log likelihood $\log p(x) = \log \int p(x,z)dz$. To make this tractable, one can instead maximize a lower bound on $\log p(x)$, i.e. the ELBO, which can obtained by re-writing $\log p(x)$ as $\log p(x) = \log \int \frac{q(z|x)}{q(z|x)} p(x,z)dz = \log\Big(\mathbb{E}_{q(z|x)}\Big[\frac{p(x,z)}{q(z|x)}\Big]\Big)$ & applying Jensen's inequality:

$$\underbrace{\mathbb{E}_{q(z|x)}\Big[\log \frac{p(x,z)}{q(z|x)}\Big]}_{ELBO} \leq \underbrace{\log\Big(\mathbb{E}_{q(z|x)}\Big[\frac{p(x,z)}{q(z|x)}\Big]\Big)}_{\log p(x)} \tag{4}$$

The justification for maximizing ELBO can also be derived from the perspective of minimizing the KL divergence between approximate and true posterior: $\min_{\boldsymbol{\phi}} KL\big(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}|\boldsymbol{x})\big)$. This KL can be decomposed into $\log p(x) - ELBO$:

$$\begin{aligned} KL\big(q(z|x) \;||\; p(z|x)\big) &= \int q(z|x) \log \frac{q(z|x)}{p(z|x)} dz = \int q(z|x) \log \frac{q(z|x)}{p(z,x)/p(x)} dz \\ &= \int q(z|x) \log p(x) \frac{q(z|x)}{p(z,x)} dz \\ &= \int q(z|x) \log p(x) dz + \mathbb{E}_{q(z|x)}\Big[\log \frac{q(z|x)}{p(z,x)}\Big] \\ &= \log p(x) - ELBO \end{aligned}$$

The optimization problem reduces to minimizing $-ELBO$ (so maximizing ELBO), since $\log p(x)$ is constant w.r.t parameters $\boldsymbol{\phi}$.

When ELBO is used as an optimization objective for learning NNs parameterized by $\boldsymbol{\theta}, \boldsymbol{\phi}$, it is often written in the following form:

$$ELBO_{VAE} = \underbrace{\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\big]}_{\text{reconstruction: data log lik.}} - \underbrace{KL\big(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \;||\; p(\boldsymbol{z})\big)}_{\text{regularization: diverg. from prior}} \tag{5}$$

This highlights the two parts of the objective: reconstruction and regularization.

Intuition for the reconstruction part can be obtained by noting the connection to non-variational autoencoders. If the dimensionality of $\boldsymbol{z}$ is chosen to be much smaller than that of $\boldsymbol{x}$, then VAEs can be seen as a probabilistic generative version of deterministic autoencoders. The latter learn to reconstruct a given input $\boldsymbol{x}$ by passing it through a bottleneck that restricts representational capacity to obtain a reconstruction $\hat{\boldsymbol{x}}$. For VAEs, $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ can be interpreted as an *encoder* that maps $\boldsymbol{x} \in \mathcal{X}$ into a lower-dimensional $\boldsymbol{z} \in \mathcal{Z}$. $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ can be seen as a *decoder*: it decodes $\tilde{\boldsymbol{z}} \sim q_{\phi}(\cdot|\boldsymbol{x})$ into a reconstruction $\hat{\boldsymbol{x}} \sim p_{\boldsymbol{\theta}}(\cdot|\tilde{\boldsymbol{z}})$. The first term in Equation 5 rewards making the output $\hat{\boldsymbol{x}}$ close to the given input $\boldsymbol{x}$.

The intuition for the regularization part of ELBO is that this term encourages the distribution of the encoder outputs to stay close to the prior $p(\boldsymbol{z})$. The prior is usually chosen as a parameterless standard distribution, e.g. $\mathcal{N}(0,1)$. More sophisticated priors have also been proposed [57, 58]. Ultimately, structured variational inference approaches advocate learning the parameters of the prior as well, leaving only the structure as fixed, hence providing regularization via structural assumptions. For example, disentangled sequential autoencoder (DSA) [59] postulates a sequential Markov structure that separates the effect of static (time-independent) and dynamic aspects. DSA models the prior using recurrent networks and then uses a sequential version of ELBO to train weights of NNs that parameterize the prior. Chapter 5 presents descriptions and experiments with several other sequential and structured VAE variants.

Conditional variational autoencoder (CVAE) [60] is a useful VAE variant that can condition on auxiliary input. CVAE defines an encoder $q_{\boldsymbol{\phi}}(\boldsymbol{z} \,|\, \boldsymbol{y}, \boldsymbol{x})$, a prior $p_{\boldsymbol{\theta}_{pr}}(\boldsymbol{z} \,|\, \boldsymbol{x})$ and a decoder $p_{\boldsymbol{\theta}_{dec}}(\boldsymbol{y} \,|\, \boldsymbol{z}, \boldsymbol{x})$. ELBO for CVAE with the output variable $\boldsymbol{y}$ is reformulated as:

$$ELBO_{\mathrm{CVAE}}(\boldsymbol{\phi}, \boldsymbol{\theta}_{dec}, \boldsymbol{\theta}_{pr}) = \log p_{\boldsymbol{\theta}_{dec}}(\boldsymbol{y} \,|\, \boldsymbol{z}, \boldsymbol{x}) - KL\big(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{y}, \boldsymbol{x}) || p_{\boldsymbol{\theta}_{pr}}(\boldsymbol{z}|\boldsymbol{x})\big) \quad (6)$$

Chapter 4 of this thesis introduces an approach that builds on the CVAE ideas, but instead of using a fixed prior as a regularizer, it uses simulation to 'regularize' the decoder. This allows to use trainable $p_{\boldsymbol{\theta}_{pr}}(\boldsymbol{z}|\boldsymbol{x})$ to represent a flexible (mixture) distribution and to re-interpret it as posterior distribution over simulation parameters. This is motivated by the ideas similar to structured variational inference, which also allows to adapt parameters of the priors. Here, the prior gains an implicit structure from being 'bound' to express the posterior over simulation parameters.

VAEs have been used extensively in recent robotics research to learn low-dimensional state representations. A recent survey on state representation learning cites a number of works that developed VAE variants and applied them to robotics scenarios [61]. However, a significant drawback of VAEs is that these methods are not particularly data-efficient. Hence, these require either using a large number of real samples or training from simulated observations, then solving the sim-to-real problem. To address this issue, Chapter 4 proposes to a novel training procedure that can utilize ample simulation data for decoder training, and can quickly shift the approximate posterior expressed by the encoder to incorporate real observations in a more data-efficient way. Chapter 5 proposes an approach that aims to improve data efficiency when training on the target domain by retaining latent space structure from a source domain.

Chapter 3

Bayesian Optimization with Informed Kernels

Bayesian optimization is particularly promising for robotics, since it provides a data-efficient way to learn from hardware trials. However, early BO experiments on hardware mostly involved optimizing low-dimensional controllers. To scale up, BO needs to incorporate prior knowledge.

This thesis first presents an approach that allows incorporating information from simulations into GP kernels. This is achieved by using a neural network (NN) to learn an informed similarity metric from simulated trajectory summaries. This is used to construct a simulation-informed kernel. Experiments on the ATRIAS bipedal robot demonstrate that using this kernel during BO on hardware significantly outperforms uninformed BO using only 10 hardware trials.

Next, the thesis presents comparisons between kernel-based vs prior-based ways of utilizing simulation data, showing that kernel-based methods can cope with low simulation fidelity more effectively.

To allow building simulation-informed kernels in a domain-agnostic way, the thesis presents BO-SVAE-DC: an algorithm that trains from full trajectories. These could be sampled at high frequency, hence recording a large amount of data per trajectory. BO-SVAE-DC proposes a model and architecture for a sequential variational autoencoder that embeds the space of simulated trajectories into a lower-dimensional space of latent paths in an unsupervised way. BO-SVAE-DC also allows to further compress the search space for BO by reducing exploration in parts of the state space that are undesirable, without requiring explicit constraints on controller parameters. This approach is validated with hardware experiments on a Daisy hexapod robot and an ABB Yumi manipulator. These experiments show that BO-SVAE-DC outperforms uninformed BO using 10 hardware trials and confirm that the same learning procedure succeeds for different areas of robotics: locomotion and manipulation.

The modular design of SVAE used for BO-SVAE-DC allows updating the latent components of SVAE from hardware observations. This can be accomplished by optimizing GP marginal likelihood and propagating the resulting gradients through the NNs. Allowing to update only the latent components is key, since the limited amount of data from hardware trials would be insufficient to significantly alter larger NNs that work with full trajectories. However, updating simulation-based kernels might be insufficient to close a severe sim-to-real gap if given a limited budget of trials. Moreover, some cases might call for additional care to guard against negative transfer, for example when simulation data could be corrupted or misleading. To address this, BO-BAK is introduced at the end of this chapter. This approach proposes to sample the choice of a kernel (simulation-based vs uninformed) at each BO trial. With this, simulation-informed kernels can help BO to quickly discover promising regions, but corrupted kernels do not degrade the performance of BO. The benefits of BO-BAK are demonstrated with a Yumi robot performing task-oriented grasping. The simulation-informed kernel is constructed via incorporating NN that maps high-dimensional point cloud input into grasp stability and task suitability metrics. These experiments demonstrate that BO can be formulated to benefit from high-dimensional camera input, while successfully utilizing low-fidelity and degraded simulation kernels.

3.1 Simulation-informed Kernels from Trajectory Summaries

Initial success of incorporating simulation information into GP kernels for BO was achieved by extracting task-specific features [13, 62]. While such features can be useful and robust, constructing them requires domain expertise. Moreover, earlier approaches could only search over a limited number of points/controllers, since they required pre-computing the features for each controller that could be evaluated during hardware trials.

This thesis presents an approach that resolves both issues by first proposing to train neural networks on trajectory summaries from simulation[1]. The summaries can be constructed by simply sub-sampling trajectory readings at fixed intervals. For example, if the simulator records the state of the robot by keeping track of the position, velocity and tilt of the torso, one could sub-sample these measurements to create a low-dimensional vector that roughly characterizes this trajectory. Then, a neural network (NN) can be trained to output such trajectory summaries given controller parameters as input. This NN can serve as a powerful function approximator for learning to represent the mapping between the space of controller parameters and the space of trajectory summaries. During BO this NN can be used to compute similarities between any controllers, removing the need to pre-compute these from simulation. An additional benefit is that this approach can be cost-agnostic, since it defines similarity between trajectory summaries instead of paying attention to features that could be more directly tied to a particular cost or objective.

[1]Work presented in this section was published by the thesis author in [2, 4]

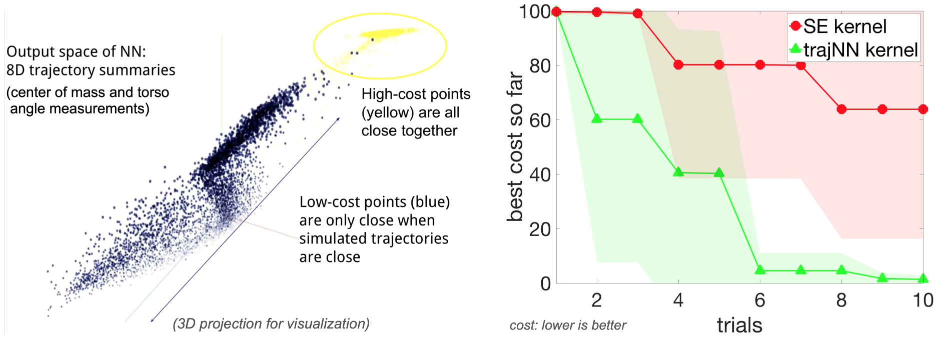


Figure 3.1: Left: A visualization of ϕ_{trajNN} output (i.e. approximate trajectory summaries) given a range of inputs (i.e. controllers with various parameters). 8D trajectory summaries from simulation (using controller from Section 4.4 in [4]) are projected into 3D for visualization. The color indicates cost (from low:blue to yellow:high). The high-cost (yellow) points appear close together, since the robot falls quickly with failing controllers (start-tilt-fall trajectories). Right: results for BO on hardware with a 9D reactive-stepping controller; shaded regions indicate 1 st. dev. (the right plot is from Section 5.1.2 of [4]).

Algorithm trajNN outlines the steps for NN training. First, a dataset for NN to fit is obtained from simulation: $\mathcal{D}_{sim} = \{(\boldsymbol{x}^{(i)}, \boldsymbol{\xi}_{\boldsymbol{x}}^{(i)})\}$, where $\boldsymbol{x}$ is a vector of controller parameters of a parametric controller, $\boldsymbol{\xi}_{\boldsymbol{x}}$ is a trajectory summary obtained when running simulation and using $\boldsymbol{x}$ for control. Then NN is trained using a standard gradient descent on a commonly used L2 loss (L1 can also be used and often yields faster training). The resulting NN can be seen as a function $\phi_{\text{trajNN}}(\cdot)$ that outputs approximate trajectory summary $\hat{\boldsymbol{\xi}}_{\boldsymbol{x}}$ for an given input controller $\boldsymbol{x}$. Hence, ϕ_{trajNN} can be used as a kernel transform and BO can use a simulation-informed kernel $k_{traj_{\text{NN}}}$:

Algorithm trajNN: Train ϕ_{trajNN}

```
// construct dataset $\mathcal{D}_{sim}$
$\mathcal{D}_{sim} \leftarrow \{\}$; $M \leftarrow$ desired dataset size
for $i = 1, ..., M$ do
    sample controller parameters $\boldsymbol{x}^{(i)}$
            // (e.g. from a Sobol grid)
    run simulation using $\boldsymbol{x}^{(i)}$ for control
    summary $\boldsymbol{\xi}_{\boldsymbol{x}}^{(i)} \leftarrow$ readings every $k^{th}$ step
            // (e.g. CoM, torso angle, etc)
    $\mathcal{D}_{sim} \leftarrow \mathcal{D}_{sim} \cup \{(\boldsymbol{x}^{(i)}, \boldsymbol{\xi}_{\boldsymbol{x}}^{(i)})\}$
// train $\phi_{\text{trajNN}}$: NN with inp. $\boldsymbol{x}$, outp. $\hat{\boldsymbol{\xi}}_{\boldsymbol{x}}$
while not converged do
    grad. descent on minibatches from $\mathcal{D}_{sim}$
    using $Loss_{NN} = \frac{1}{2}\sum_{i=1}^{N} ||\hat{\xi}_{\boldsymbol{x}_i} - \xi_{\boldsymbol{x}_i}||^2$
    // NN output $\hat{\boldsymbol{\xi}}_{\boldsymbol{x}} = \phi_{\text{trajNN}}(\boldsymbol{x})$
    // is the 'reconstructed' traj. summary
```

$$k_{trajNN}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sigma_k^2 \exp\left(-\tfrac{1}{2}\boldsymbol{t}_{ij}^T \operatorname{diag}(\boldsymbol{\ell})^{-2}\boldsymbol{t}_{ij}\right) \; ; \; \boldsymbol{t}_{ij} = \phi_{trajNN}(\boldsymbol{x}_i) - \phi_{trajNN}(\boldsymbol{x}_j) \quad (7)$$

Figure 3.1 shows key results obtained for hardware experiments with the ATRIAS robot (right plot). Section 3.1.2 & Appendix A in [4] give further details regarding the data collection and NN training, Section 4 in [4] gives details regarding the hardware setup and controllers used, Section 5.1.3 in [4] describes the experiments.

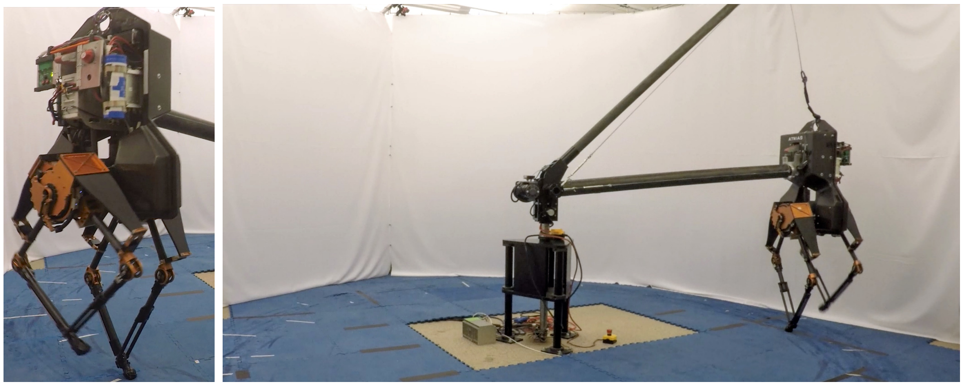


Figure 3.2: ATRIAS robot during our BO trials at CMU. ATRIAS is a human-scale biped [63]. This experimental platform was used for experiments in [2, 3, 4].

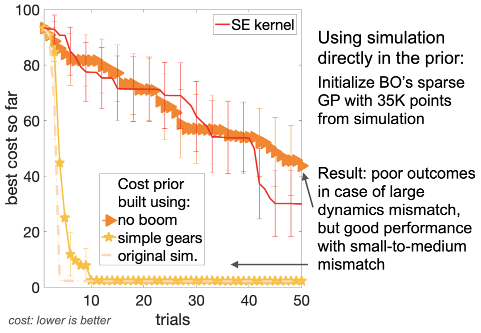
While the informed BO algorithm that uses k_{trajNN} is easy to describe, it takes a few additional insights to understand why this approach offers a significant improvement over uninformed BO. The next section provides comparisons of this kernel-centric approach with alternative prior-based approaches. Discussion in the further sections, which propose more advanced kernel-based approaches, gives intuition as to why kernel-centric methods can yield ultra data-efficient BO.

3.2 Informing Prior Mean vs Kernels

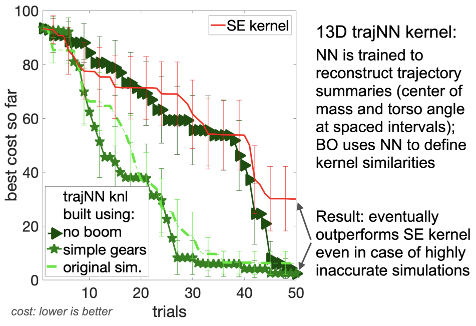
GPs consist of two main components: the mean function and the kernel. Specifying a prior mean function has been a common way to incorporate prior knowledge. When a prior mean function could not be constructed manually, the next default has been to incorporate prior (simulated) observations into a GP as 'fake' data. Then, this GP would be used to further learn from true data on the target (real) domain. This thesis work argues that embedding prior knowledge into GP kernels instead provides a more flexible way to capture simulation-based information.

A classic book on GPs for machine learning [28] gives advice on shaping the prior mean function (Section 2.7 in [28]). It shows that incorporating a fixed deterministic mean function is straightforward and also gives examples of how to express a prior mean as a linear combination of a given set of basis functions. This approach has been used as early as 1975, e.g. with polynomial features $h(\boldsymbol{x}) = (1, \boldsymbol{x}, \boldsymbol{x}^2, ...)$ [64].

Modern approaches seek more flexibility. One direction is to initialize GPs with points from simulated trials directly. This can be formulated as a multi-fidelity problem, with different fidelities for simulated vs real points [47, 48]. The main issue is that one needs to carefully weigh the contributions from simulated vs real trials, since 'fake' data from inaccurate simulations can overwhelm the effects of the real data. This can be done if simulation fidelity is known, but is more challenging otherwise. Another issue arises if simulation is cheap and the number of simulated/fake points is too large to be handled by exact GPs. Sparse GPs can



(a) BO with 'cost prior'



(b) BO with trajNN kernel

Figure 3.3: Experiments with varying simulator fidelity for BO with a 50D Virtual Neuromuscular Controller on a bipedal robot model. Plots show best cost so far for mean over 50 runs for each algorithm, 95% CIs. See Section 5.3 in [4] for further details.

be used in such cases ([26, 27] implement several versions), however this may cause loss in precision due to approximate inference.

Figure 3.3a illustrates the effects of simulation fidelity on such 'cost prior' formed by adding 35K simulated points $\boldsymbol{x}$ and the corresponding simulation-based results $cost_{sim}(\boldsymbol{x})$ to a Sparse GP. Then, this GP is used for BO with a 50-dimensional virtual neuromuscular controller (VNMC) [65]. These experiments are conducted in simulation, with a high-fidelity simulator of a bipedal robot used as a surrogate for reality (labeled 'original sim' in the legend). A medium-fidelity simulator is obtained by replacing the original high-fidelity gear model with a commonly used approximation for geared systems (this version is called 'simple gears' in the legend). A low-fidelity simulator is obtained by using simple gears and omitting the computation of the forces exerted by the boom on the robot (labeled 'no boom' in the legend). For high- and medium-fidelity simulators the performance of BO with 'cost prior' is excellent. However, for the low fidelity the result is worse than the baseline (uninformed) BO. This sheds light on why a part of the learning and control communities might be quite sympathetic to the 'cost prior' methods: if experiments are conducted on synthetic data that matches the target domain fairly well, then adding 'fake' points to the prior offers a good solution. The robotics community has to deal with real data, and in recent years has become interested in advanced scenarios for which simple models and accurate simulators are not (yet) tractable to construct. For such cases 'cost prior' approaches could be detrimental when simulation fidelity is unknown.

Figure 3.3b shows results when using simulation-informed trajNN kernel described in the previous section. The approach is able to benefit from high- and medium-fidelity simulations, and eventually significantly outperforms uninformed BO when using low-fidelity simulation as well. Section 5.3 in [4] provides further comparisons with several other methods. These experiments indicate that kernel-based approaches have the potential for being robust to a large sim-to-real gap.

3.3 Bayesian Optimization in Variational Latent Spaces

The kernel-based approach described in the previous section required sub-sampling trajectories. An alternative is to embed full trajectories into the kernel. Conceptually, this could be similar to Behavior Based Kernel (BBK) [51]. It showed success on low-dimensional analytic benchmarks. BBK used a particular form of symmetrized KL $\big(KL_{\mathrm{BBK}}(p,q){:=}KL(p||q)^{1/2}{+}KL(q||p)^{1/2}\big)$ as a distance metric for the kernel: $k_{\mathrm{BBK}} = \exp\big(\text{-}\alpha KL_{\mathrm{BBK}}\big(p(\boldsymbol{\xi}|\boldsymbol{x}_i), p(\boldsymbol{\xi}|\boldsymbol{x}_j)\big)\big)$. However, the above would not scale for trajectories from modern full-scale robotic systems with higher-dimensional controllers and state spaces. Here, 'higher-dimensional' does not indicate pixel or point cloud input, since utilizing robot's joint angles & velocities and objects' poses already presents challenge, especially if trajectories are sampled at high frequencies (for example: $500Hz - 1kHz$ commonly used for modern manipulation systems).

This section presents an approach for embedding the space of simulated trajectories into a lower-dimensional space of latent paths in an unsupervised way[2]. The proposed modular sequential variational autoencoder (sVAE) allows learning the relationship between controller parameters and the latent representation of trajectories. Furthermore, the approach also allows to achieve 'dynamic' compression of the search space for BO by reducing exploration in parts of the state space that are undesirable, without requiring explicit constraints on controller parameters. Figure 3.4 gives an overview of the proposed approach.

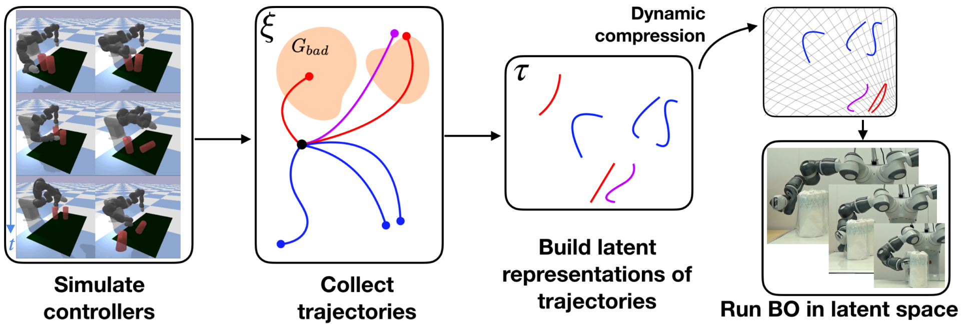


Figure 3.4: An overview of BO-SVAE-DC approach: Start by simulating controllers and collecting their trajectories $\boldsymbol{\xi}$, along with the fraction of time spent in undesirable regions given by G_{bad}. Then, learn to embed trajectories into a lower-dimensional a space of latent paths $\boldsymbol{\tau}$. Next, 'dynamic' compression is used to scale distances between latent paths based on their desirability. This dynamically compressed latent space is used for BO on hardware. Trajectory data $\boldsymbol{\xi}$ consists of high-frequency readings of robot joint angles and object position/velocity estimates (the framework can accommodate vision-based data in the future, but this direction is not explored in this part of the work).

[2]Work presented in this section was published by the thesis author in [5]. Some paragraphs reuse the text from [5], which is permitted by CC-BY 4.0 license.

SVAE-DC: Graphical Model and ELBO

The problem of learning low-dimensional trajectory embeddings $\boldsymbol{\tau}$ is formulated as a joint variational inference (VI) problem: learning to reconstruct trajectories $\boldsymbol{\xi}$ with a VAE-like component, while at the same time learning to associate controller parameters $\boldsymbol{x}$ with their corresponding probability distributions over the latent paths $p(\boldsymbol{\tau}|\boldsymbol{x})$. The training is guided by the Evidence Lower Bound (ELBO) derived directly from the modeling assumptions and doesn't require any auxiliary objectives. The following list summarizes the notation used in this section:

$\boldsymbol{x}$: a shorthand for denoting controller/policy with parameters $\boldsymbol{x}, \boldsymbol{x} \in \mathbb{R}^D$; policies can be either deterministic or stochastic; $\boldsymbol{x}$ could contain gains for structured controllers, NN weights for NN-based controllers, or any other continuous parameters

$\boldsymbol{\xi} \equiv \boldsymbol{\xi}_{1:T}$: an original trajectory with high-frequency sensor readings (T steps)

$\boldsymbol{\tau} \equiv \boldsymbol{\tau}_{1:K}$: a latent 'path' (embedding of a trajectory; can set $K \ll T$)

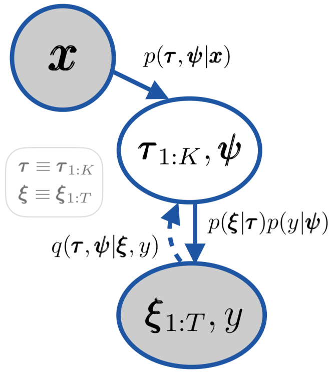
$p(\boldsymbol{\xi}_{1:T}|\boldsymbol{x})$: a conditional probability distribution over the trajectories induced by controller $\boldsymbol{x}$; the relationship between the controller and trajectories could be probabilistic either because the controller is stochastic, or because the simulator environment is stochastic, or both

$p(\boldsymbol{\tau}_{1:K}|\boldsymbol{x})$: a conditional probability distribution over the space of latent paths induced controller by $\boldsymbol{x}$

$G_{bad} : S \rightarrow \{0, 1\}$ a map denoting whether an observation $\boldsymbol{\xi}_t \in S$ is within an undesirable region ($G_{bad} : S \rightarrow [0, 1]$ can be used to give probabilities, but for users it is easier to make rough thresholded estimates, so $G_{bad} : S \rightarrow \{0, 1\}$ is the default)

y : a fraction of time that $\boldsymbol{\xi}$ spends in the undesirable regions; $\boldsymbol{\psi}$ would express the analogous notion for a latent path $\boldsymbol{\tau}$ (after training)

The goal is to learn $p(\boldsymbol{\tau}, \boldsymbol{\psi}|\boldsymbol{x})$, since this is the main component that will be used for the BO kernel. $p(\boldsymbol{\tau}|\boldsymbol{x})$ is analogous to $p(\boldsymbol{\xi}|\boldsymbol{x})$, but the paths $\boldsymbol{\tau}$ are encoded in a lower-dimensional latent space. As a measure of trajectory 'quality', y keeps track of how long each trajectory spends in undesirable regions. For latent paths the analogous notion is encoded in $\boldsymbol{\psi} = \boldsymbol{\psi}_{1:K}$ (this additional modularity can enable fast online updates). G_{bad}, which is used to compute y, can be specified approximately, since the optimization process used in this work would not impose any hard constraints. Figure 3.5 summarizes the graphical model (it is a 'sketch', since some independencies not shown explicitly).



$\boldsymbol{x}$
$p(\boldsymbol{\tau}, \boldsymbol{\psi}|\boldsymbol{x})$
$\boldsymbol{\tau}_{1:K}, \boldsymbol{\psi}$
$\boldsymbol{\tau} \equiv \boldsymbol{\tau}_{1:K}$
$\boldsymbol{\xi} \equiv \boldsymbol{\xi}_{1:T}$
$p(\boldsymbol{\xi}|\boldsymbol{\tau})p(y|\boldsymbol{\psi})$
$q(\boldsymbol{\tau}, \boldsymbol{\psi}|\boldsymbol{\xi}, y)$
$\boldsymbol{\xi}_{1:T}, y$

Figure 3.5: A sketch of the generative and inference model.

The backbone of the model (considering random variables $\boldsymbol{x}, \boldsymbol{\xi}, \boldsymbol{\tau}$, omitting $y, \boldsymbol{\psi}$) is inspired by the hierarchical models in [66, 67]. However, these works considered supervised and semi-supervised settings, where a discrete label was associated with each high-dimensional data point (e.g. a label for an image). The model presented in this work handles sequential trajectory data instead, so the internal structure of the data is different. Nonetheless, for the moment let us think about each trajectory as a point in some high-dimensional space. The idea is to interpret controllers $\boldsymbol{x}$ as continuous 'labels' for trajectories $\boldsymbol{\xi}$. In this work $\boldsymbol{x}$ is observed, because we know which controller is executed when obtaining a trajectory $\boldsymbol{\xi}$ in simulation. Hence, there is no further uncertainty about $\boldsymbol{x}$ and the terms involving $p(\boldsymbol{x}), q(\boldsymbol{x}|\boldsymbol{\tau})$ would not appear. Such formulation treats $\boldsymbol{x}$ and $\boldsymbol{\xi}$ as observed data, which is in fact what is available from simulation. The controllers can be sampled at random during data collection, since this work assumes access to a relatively inexpensive simulator (in a sense that it is viable to simulate 100K+ trajectories for offline kernel construction). The ELBO derivation for this backbone part of the model is as follows:

$$\text{Backbone Generative model: } p(\boldsymbol{\tau}, \boldsymbol{\xi} \mid \boldsymbol{x}) = p(\boldsymbol{\tau}_{1:K}|\boldsymbol{x}) \textstyle\prod_{t=1}^{T} p(\boldsymbol{\xi}_t|\boldsymbol{\xi}_{t-1}, \boldsymbol{\tau}_{1:K})$$
$$\text{Approximate posterior: } q(\boldsymbol{\tau}|\boldsymbol{\xi}) := q(\boldsymbol{\tau}_{1:K}|\boldsymbol{\xi}_{1:T})$$

$$\log p(\boldsymbol{\xi}|\boldsymbol{x}) = \log \int_{\boldsymbol{\tau}} p(\boldsymbol{\xi}, \boldsymbol{\tau}|\boldsymbol{x}) d\boldsymbol{\tau} = \log \int_{\boldsymbol{\tau}} \frac{q(\boldsymbol{\tau}|\boldsymbol{\xi})}{q(\boldsymbol{\tau}|\boldsymbol{\xi})} p(\boldsymbol{\xi}, \boldsymbol{\tau}|\boldsymbol{x}) d\boldsymbol{\tau} = \log \mathbb{E}_{q(\boldsymbol{\tau}|\boldsymbol{\xi})}\left[\frac{p(\boldsymbol{\xi}, \boldsymbol{\tau}|\boldsymbol{x})}{q(\boldsymbol{\tau}|\boldsymbol{\xi})}\right]$$
$$\underbrace{\mathbb{E}_{q(\boldsymbol{\tau}|\boldsymbol{\xi})}\left[\log \frac{p(\boldsymbol{\xi}, \boldsymbol{\tau}|\boldsymbol{x})}{q(\boldsymbol{\tau}|\boldsymbol{\xi})}\right]}_{ELBO_{\text{backbone}}} \leq \underbrace{\log \mathbb{E}_{q(\boldsymbol{\tau}|\boldsymbol{\xi})}\left[\frac{p(\boldsymbol{\xi}, \boldsymbol{\tau}|\boldsymbol{x})}{q(\boldsymbol{\tau}|\boldsymbol{\xi})}\right]}_{\log p(\boldsymbol{\xi}|\boldsymbol{x})}$$

$$\begin{aligned} ELBO_{\text{backbone}} &= \mathbb{E}_{q(\boldsymbol{\tau}|\boldsymbol{\xi})}\left[\log \frac{p(\boldsymbol{\xi}|\boldsymbol{\tau}, \boldsymbol{x}) p(\boldsymbol{\tau}|\boldsymbol{x})}{q(\boldsymbol{\tau}|\boldsymbol{\xi})}\right] = \mathbb{E}_{q(\boldsymbol{\tau}|\boldsymbol{\xi})}\left[\log \frac{p(\boldsymbol{\xi}|\boldsymbol{\tau}) p(\boldsymbol{\tau}|\boldsymbol{x})}{q(\boldsymbol{\tau}|\boldsymbol{\xi})}\right] \\ &= \mathbb{E}_{q(\boldsymbol{\tau}|\boldsymbol{\xi})}\left[\log p(\boldsymbol{\xi}|\boldsymbol{\tau}) + \log p(\boldsymbol{\tau}|\boldsymbol{x}) - \log q(\boldsymbol{\tau}|\boldsymbol{\xi})\right] \end{aligned}$$

The above can be written explicitly as a function of the parameters of the variational approximation $\boldsymbol{\phi} = [\boldsymbol{\phi}_{\tau}]$ and the parameters of the generative model $\boldsymbol{w} = [\boldsymbol{w}_{\tau}, \boldsymbol{w}_{\xi}]$:

$$\begin{aligned} &ELBO_{\text{backbone}}(\boldsymbol{w}, \boldsymbol{\phi}|\boldsymbol{\xi}, \boldsymbol{x}) = \\ &\quad = \mathbb{E}_{\tilde{\boldsymbol{\tau}} \sim q_{\boldsymbol{\phi}_{\tau}}(\boldsymbol{\tau}|\boldsymbol{\xi})}\Big[\log p_{\boldsymbol{w}_{\xi}}(\boldsymbol{\xi}|\tilde{\boldsymbol{\tau}}) + \log p_{\boldsymbol{w}_{\tau}}(\tilde{\boldsymbol{\tau}}|\boldsymbol{x}) - \log q_{\boldsymbol{\phi}_{\tau}}(\tilde{\boldsymbol{\tau}}|\boldsymbol{\xi})\Big] \end{aligned} \tag{8}$$

In this work, $\boldsymbol{\phi}, \boldsymbol{w}$ are weights of deep neural networks. It is customary to drop the subscripts indicating NN weight parameters and write q, p for a shorthand notation.

The derivation for $ELBO_{\text{SVAE-DC}}$ follows a similar logic and includes $y, \boldsymbol{\phi}$:

$$\text{Generative model: } p(\boldsymbol{\tau}, \boldsymbol{\psi}, \boldsymbol{\xi}, y \mid \boldsymbol{x}) := p(\boldsymbol{\tau}_{1:K}, \boldsymbol{\psi}|\boldsymbol{x}) p(y|\boldsymbol{\psi}) \textstyle\prod_{t=1}^{T} p(\boldsymbol{\xi}_t|\boldsymbol{\xi}_{t-1}, \boldsymbol{\tau}_{1:K})$$
$$\text{Approximate posterior: } q(\boldsymbol{\tau}, \boldsymbol{\psi}|\boldsymbol{\xi}, y) := q(\boldsymbol{\tau}_{1:K}, \boldsymbol{\psi}|\boldsymbol{\xi}_{1:T}, y)$$

$$\begin{aligned} \log p(\boldsymbol{\xi}, y|\boldsymbol{x}) &= \log \int_{\boldsymbol{\tau}, \boldsymbol{\psi}} p(\boldsymbol{\xi}, y, \boldsymbol{\tau}, \boldsymbol{\psi}|\boldsymbol{x}) d\boldsymbol{\tau} d\boldsymbol{\psi} = \log \int_{\boldsymbol{\tau}, \boldsymbol{\psi}} \frac{q(\boldsymbol{\tau}, \boldsymbol{\psi}|\boldsymbol{\xi}, y)}{q(\boldsymbol{\tau}, \boldsymbol{\psi}|\boldsymbol{\xi}, y)} p(\boldsymbol{\xi}, y, \boldsymbol{\tau}, \boldsymbol{\psi}|\boldsymbol{x}) d\boldsymbol{\tau} d\boldsymbol{\psi} \\ &= \log \mathbb{E}_{q(\boldsymbol{\tau}, \boldsymbol{\psi}|\boldsymbol{\xi}, y)}\left[\frac{p(\boldsymbol{\xi}, y, \boldsymbol{\tau}, \boldsymbol{\psi}|\boldsymbol{x})}{q(\boldsymbol{\tau}, \boldsymbol{\psi}|\boldsymbol{\xi}, y)}\right] \end{aligned}$$

$$\underbrace{\mathbb{E}_{q(\boldsymbol{\tau},\boldsymbol{\psi}|\boldsymbol{\xi},y)}\Big[\log\frac{p(\boldsymbol{\xi},y,\boldsymbol{\tau},\boldsymbol{\psi}|\boldsymbol{x})}{q(\boldsymbol{\tau},\boldsymbol{\psi}|\boldsymbol{\xi},y)}\Big]}_{ELBO_{\text{SVAE-DC}}} \leq \underbrace{\log\mathbb{E}_{q(\boldsymbol{\tau},\boldsymbol{\psi}|\boldsymbol{\xi},y)}\Big[\frac{p(\boldsymbol{\xi},y,\boldsymbol{\tau},\boldsymbol{\psi}|\boldsymbol{x})}{q(\boldsymbol{\tau},\boldsymbol{\psi}|\boldsymbol{\xi},y)}\Big]}_{\log p(\boldsymbol{\xi},y|\boldsymbol{x})}$$

$$\begin{aligned}ELBO_{\text{SVAE-DC}} &= \mathbb{E}_{q(\boldsymbol{\tau},\boldsymbol{\psi}|\boldsymbol{\xi},y)}\Big[\log\frac{p(\boldsymbol{\xi}|\boldsymbol{\tau},\boldsymbol{x})p(\boldsymbol{\tau}|\boldsymbol{x})p(y|\boldsymbol{\psi})}{q(\boldsymbol{\tau},\boldsymbol{\psi}|\boldsymbol{\xi},y)}\Big]\\ &= \mathbb{E}_{q(\boldsymbol{\tau},\boldsymbol{\psi}|\boldsymbol{\xi},y)}\Big[\log\frac{p(\boldsymbol{\xi}|\boldsymbol{\tau})p(\boldsymbol{\tau}|\boldsymbol{x})p(y|\boldsymbol{\psi})}{q(\boldsymbol{\tau},\boldsymbol{\psi}|\boldsymbol{\xi},y)}\Big]\\ &= \mathbb{E}_{q(\boldsymbol{\tau},\boldsymbol{\psi}|\boldsymbol{\xi},y)}\big[\log p(\boldsymbol{\xi}|\boldsymbol{\tau})+\log p(y|\boldsymbol{\psi})+\log p(\boldsymbol{\tau}|\boldsymbol{x})-\log q(\boldsymbol{\tau},\boldsymbol{\psi}|\boldsymbol{\xi},y)\big]\end{aligned}$$

Finally, we can write the above as a function of parameters $\boldsymbol{w},\boldsymbol{\phi}$ explicitly:

$$\begin{aligned}ELBO_{\text{SVAE-DC}}(\boldsymbol{w},\boldsymbol{\phi}|\boldsymbol{x},\boldsymbol{\xi},y) = \mathbb{E}_{\tilde{\boldsymbol{\tau}},\tilde{\boldsymbol{\psi}}\sim q(\boldsymbol{\tau},\boldsymbol{\psi}|\boldsymbol{\xi},y)}\Big[&\\ \log p_{\boldsymbol{w}_\xi}(\boldsymbol{\xi}|\tilde{\boldsymbol{\tau}})+\log p_{\boldsymbol{w}_y}(y|\tilde{\boldsymbol{\psi}})+\log p_{\boldsymbol{w}_\tau}(\tilde{\boldsymbol{\tau}},\tilde{\boldsymbol{\psi}}|\boldsymbol{x})-\log q_{\boldsymbol{\phi}_\tau}(\tilde{\boldsymbol{\tau}},\tilde{\boldsymbol{\psi}}|\boldsymbol{\xi},y)\Big]&\end{aligned} \tag{9}$$

The 4 terms inside the expectation in Equation 9 above are the 4 neural networks whose parameters will be optimized by gradient ascent to maximize $ELBO_{\text{SVAE-DC}}$.

The above formulation is agnostic to whether controllers are stochastic or deterministic, and to whether the simulators used to collect samples are stochastic or deterministic. This is especially convenient, since deterministic controllers and simulators are used widely in robotics, while the reinforcement learning community frequently considers stochastic policies and environments. With SVAE-DC model: the stochasticity of either the controllers or the environment (or both) will be encoded in $p(\boldsymbol{\tau}|\boldsymbol{x})$. The model is applicable even in the case of deterministic controllers and environment (i.e. a deterministic relationship between $\boldsymbol{x}$ and $\boldsymbol{\xi}$), because of the bottleneck $\boldsymbol{\tau}$ and randomness coming from the sampling of $\boldsymbol{x}$ during data collection.

One question could be: why learn an embedding into a lower-dimensional space of paths jointly with learning $p(\boldsymbol{\tau}|\boldsymbol{x})$, instead of decomposing the problem into separate dimensionality reduction and $p(\boldsymbol{\tau}|\boldsymbol{x})$ modeling stages. For an arbitrary space of paths (either low-dimensional, or the original higher-dimensional space of trajectories) learning the relationship between $\boldsymbol{x}$ and the corresponding probability distribution over the paths would be challenging. This is because it involves the controller properties and the dynamics of the physical environment, both of which are usually highly non-trivial. In the joint SVAE-DC model: $p(\boldsymbol{\tau}|\boldsymbol{x})$ term can be seen as a 'regularization' part of the ELBO. It encourages the latent encoding to be well suited for modeling the relationship between $\boldsymbol{x}$ and $\boldsymbol{\tau}$. The terms pertaining to 'reconstructing' original trajectories are the encoder $q(\boldsymbol{\tau}|\boldsymbol{\xi})$ and decoder $p(\boldsymbol{\xi}|\boldsymbol{\tau})$. The learning progress for these is fast if there is sufficient capacity in the bottleneck $\boldsymbol{\tau}$. However, if these make fast progress, but produce encodings that are not easy to relate to the space of controllers – then $p(\boldsymbol{\tau}|\boldsymbol{x})$ will drop. Hence, the ELBO will be lower for this 'inconvenient' representation of $\boldsymbol{\tau}$, and this would encourage to seek alternatives. Consequently, the joint learning offered by SVAE-DC allows not only to embed the space of trajectories in a lower-dimensional space, but also encourages to find an embedding scheme that simplifies the problem of learning $p(\boldsymbol{\tau}|\boldsymbol{x})$.

SVAE-DC Kernel for Bayesian Optimization

The beginning of this chapter outlined the motivation of encoding trajectory information into the kernel via symmetrized KL between trajectory distributions $p(\boldsymbol{\xi}|\boldsymbol{x})$ induced by the corresponding controllers. Using SVAE-DC, one can instead define a kernel in the space of lower-dimensional latent paths[3]:

$$k_{\text{latent}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\Big(\text{-}\alpha KL_{sym}\big(p(\boldsymbol{\tau}|\boldsymbol{x}_i)\ ,\ p(\boldsymbol{\tau}|\boldsymbol{x}_j)\big)\Big) \tag{10}$$

$$KL_{sym}(p, q) := KL(p||q) + KL(q||p)\ ;\ \text{defined in [68]} \tag{11}$$

One problem is that variational inference tends to under-estimate variances in theory [69, 70] and in practice [71, 72]. This underestimation could negatively impact the practical performance of such kernel. As a consequence, it is more practical to work with the latent means $\bar{\boldsymbol{\tau}}_{\boldsymbol{x}}, \bar{\boldsymbol{\psi}}_{\boldsymbol{x}} = E\big[p(\boldsymbol{\tau}, \boldsymbol{\psi}|\boldsymbol{x})\big]$ directly. Hence, this work starts with k_{latent} as a target form, makes a common choice of representing the latent distributions by diagonal Gaussians and derives k_{SVAE} as follows:

$$p(\boldsymbol{\tau}|\boldsymbol{x}_i) = \overbrace{\mathcal{N}(\boldsymbol{\mu}_i, \Sigma_i = \text{diag}(\boldsymbol{v}_i))}^{\mathcal{N}_i}\ ;\ p(\boldsymbol{\tau}|\boldsymbol{x}_j) = \overbrace{\mathcal{N}(\boldsymbol{\mu}_j, \Sigma_j = \text{diag}(\boldsymbol{v}_j))}^{\mathcal{N}_j}\ ;\ \boldsymbol{\mu}, \boldsymbol{v} \in \mathbb{R}^D$$

$$KL(\mathcal{N}_i||\mathcal{N}_j) = \tfrac{1}{2}\Big[tr(\Sigma_j^{-1}\Sigma_i) + \log\frac{|\Sigma_j|}{|\Sigma_i|} - D + (\boldsymbol{\mu}_j - \boldsymbol{\mu}_i)^T \Sigma_j^{-1}(\boldsymbol{\mu}_j - \boldsymbol{\mu}_i)\Big]$$

If using a common $\boldsymbol{v}$ for $p(\boldsymbol{\tau}|\boldsymbol{x})$, so $\mathcal{N}_i = \mathcal{N}(\boldsymbol{\mu}_i, \text{diag}(\boldsymbol{v})); \mathcal{N}_j = \mathcal{N}_j(\boldsymbol{\mu}_j, \text{diag}(\boldsymbol{v}))$ we get:

$$\begin{aligned}
KL_{\boldsymbol{v}}(\mathcal{N}_i||\mathcal{N}_j) &= \tfrac{1}{2}\Big[tr(I) + \log 1 - D + (\boldsymbol{\mu}_j - \boldsymbol{\mu}_i)^T\, \text{diag}(\boldsymbol{v})^{-1}(\boldsymbol{\mu}_j - \boldsymbol{\mu}_i)\Big] \\
&= \tfrac{1}{2}(\boldsymbol{\mu}_j - \boldsymbol{\mu}_i)^T\, \text{diag}(\boldsymbol{v})^{-1}(\boldsymbol{\mu}_j - \boldsymbol{\mu}_i) \\
\implies \exp\big(\text{-}\alpha KL_{\boldsymbol{v}}(\mathcal{N}_i||\mathcal{N}_j)\big) &= \exp\big(-\tfrac{\alpha}{2}(\boldsymbol{\mu}_j - \boldsymbol{\mu}_i)^T\, \text{diag}(\boldsymbol{v})^{-1}(\boldsymbol{\mu}_j - \boldsymbol{\mu}_i)\big) \\
&= \underbrace{1}_{\substack{\text{BO hyper} \\ \text{param } \sigma_k}} \cdot \exp\big(-\tfrac{1}{2}(\boldsymbol{\mu}_j - \boldsymbol{\mu}_i)^T\, \text{diag}(\underbrace{\sqrt{\alpha \boldsymbol{v}}}_{\substack{\text{BO hyper} \\ \text{param } \boldsymbol{\ell}}})^{-2}(\boldsymbol{\mu}_j - \boldsymbol{\mu}_i)\big)
\end{aligned}$$

$$\text{Hence, define: } k_{SVAE}(\boldsymbol{x}_i, \boldsymbol{x}_j) := \sigma_k^2 \exp\big(-\tfrac{1}{2}\boldsymbol{r}_\tau^T\, \text{diag}(\boldsymbol{\ell})^{-2}\boldsymbol{r}_\tau\big)\ ;\ \boldsymbol{r}_\tau := \bar{\boldsymbol{\tau}}_{\boldsymbol{x}_i} - \bar{\boldsymbol{\tau}}_{\boldsymbol{x}_j} \tag{12}$$

The above shows that we can define k_{SVAE} by utilizing SE as a 'shell' function applied to the differences between the latent means: using $\bar{\boldsymbol{\tau}}_{\boldsymbol{x}_j} - \bar{\boldsymbol{\tau}}_{\boldsymbol{x}_i}$ for $\boldsymbol{\mu}_j - \boldsymbol{\mu}_i$. During BO, σ_k and length scale hyperparameters $\boldsymbol{\ell}$ are optimized via automatic relevance determination (see Section V-A in [22]). In practice, this direct optimization is better than using variances $\Sigma_i = \text{diag}(\boldsymbol{v}_i)$ for $p(\boldsymbol{\tau}|\boldsymbol{x}_i)$ obtained from VI, because Σ_i estimates are often brittle due to variance underestimation problems of VI. Making a diagonal Gaussian assumption often works well in practice. Alternatively, using diagonal Laplace distributions performs well with noisy objectives and was used for

[3][51] cites [68] for justifying KL symmetrization with KL_{BBK}. However, in these papers, neither KL_{BBK} nor KL_{sym} from [68] are proven to yield a valid kernel in general. Nonetheless, KL_{sym} from [68] does yield a valid kernel in the Gaussian case, so it is used here to define k_{latent}.

several experiments in this work. In this case L1 norm replaces L2 norm (though the connection to k_{latent} is not as direct).

To achieve 'dynamic' compression the latent representations $\bar{\boldsymbol{\tau}}_{\boldsymbol{x}}$ can be scaled by $\textsc{dc}(\bar{\boldsymbol{\psi}}_{\boldsymbol{x}}) := 1 - \mathbb{E}\big[y|\bar{\boldsymbol{\psi}}_{\boldsymbol{x}}\big]$. With that, $\bar{\boldsymbol{\tau}}_{\boldsymbol{x}}$ that correspond to controllers frequently visiting undesirable parts of the space are scaled down. Hence undesirable controllers are brought closer together. This allows BO to reduce the number of samples from the undesirable parts of the space. 'Dynamic compression' here means this search space transformation is applied after the offline SVAE-DC training phase. This means it is obtained during BO on hardware in addition to the compression obtained by dimensionality reduction obtained by working with $\bar{\boldsymbol{\tau}}_{\boldsymbol{x}}$ instead of considering original trajectories $\boldsymbol{\xi}$. Hence, $k_{\text{SVAE-DC}}$ kernel is defined as follows:

$$\begin{aligned} &\boldsymbol{r}_{\tau\text{-DC}} = \textsc{dc}(\bar{\boldsymbol{\psi}}_{\boldsymbol{x}_i})\bar{\boldsymbol{\tau}}_{\boldsymbol{x}_i} - \textsc{dc}(\bar{\boldsymbol{\psi}}_{\boldsymbol{x}_j})\bar{\boldsymbol{\tau}}_{\boldsymbol{x}_j} \quad ; \quad \textsc{dc}(\bar{\boldsymbol{\psi}}_{\boldsymbol{x}}) := 1 - \mathbb{E}\big[y|\bar{\boldsymbol{\psi}}_{\boldsymbol{x}}\big] \\ &k_{\text{SVAE-DC}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sigma_k^2 \exp\big(-\tfrac{1}{2}\boldsymbol{r}_{\tau\text{-DC}}^T \operatorname{diag}(\boldsymbol{\ell})^{-2}\boldsymbol{r}_{\tau\text{-DC}}\big) \end{aligned} \tag{13}$$

The above formulation is convenient in practice, since the form of Equation 13 allows applying existing machinery for optimizing kernel hyperparameters $\sigma_k^2, \boldsymbol{\ell}$. The scaling can be made non-linear with $\textsc{dc}(\bar{\boldsymbol{\psi}}_{\boldsymbol{x}}) := \textit{sigmoid}\big(\alpha(\mathbb{E}[y|\bar{\boldsymbol{\psi}}_{\boldsymbol{x}}] - c)\big)$. This can help achieving aggressive compression in settings with a very small budget of trials. The additional parameters α, c, as well as $p(\boldsymbol{\tau}, \boldsymbol{\psi}|\boldsymbol{x}), p(y|\boldsymbol{\psi})$ can be optimized online in the same way as BO hyperparameters.

Overall, SVAE-DC yields a fully automatic way of learning latent trajectory embeddings in an unsupervised way. For domains where G_{bad} is easy to specify: further dynamic compression of the latent space can yield ultra data-efficient BO. All the components used during BO can be optimized online via the same methods already implemented for automatically adjusting BO hyperparameters.

BO-SVAE-DC Experiments

Sections 6 & 7 in [5] present detailed descriptions of experiments conducted to validate the proposed BO-SVAE-DC approach. The main results are briefly summarized here, with a focus on a set of experiments conducted on ABB Yumi robot at KTH.

BO-SVAE-DC has been developed with a goal to generalize beyond domain-specific kernel methods. To demonstrate this generalization, experiments were conducted for tasks from different areas of robotics: hexapod locomotion and non-prehensile manipulation. The kernels were build using the same general-purpose simulator PyBullet [73] (with 500K trajectories for NN training for each robot). SVAE-DC NNs used the same architecture and training parameters (see Section 5 in [5] for more details). BO was also run with the same settings for all experiments.

Hexapod locomotion: For hardware experiments, we used a Daisy robot from Hebi robotics [74]. Simulation for constructing the kernel used a basic robot model and free-space motion of individual joints generally transferred to hardware well. However, the overall behavior of the robot revealed a large sim-to-real gap due to

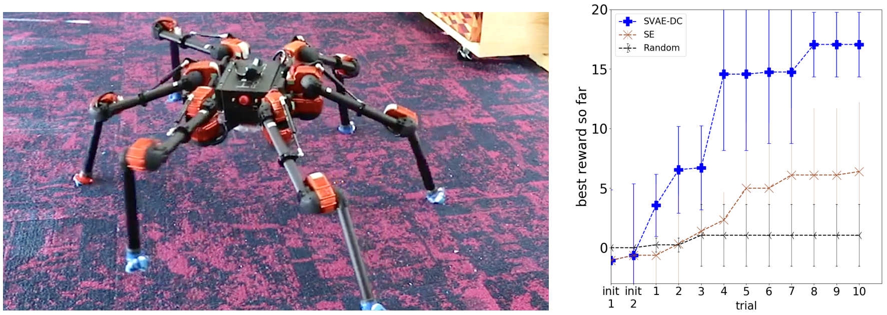


Figure 3.6: Left: Daisy hexapod during BO trials. Right: Results for BO on Daisy hardware (means of 5 runs, 90% CIs). Video of experiments: `https://youtu.be/2SvdwGZNrvY`. Note: unlike previous sections that reported experiments with costs (lower was better), starting from this section the thesis reports objective (reward) maximization (so higher is better).

difficulty of modeling contacts with the ground. Moreover, BO trials were done on a shallow carpet, without attempts to estimate friction and contact dynamics specific to this kind of surface.

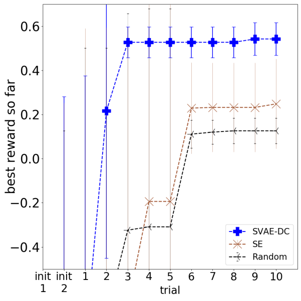
For control, we used Central Pattern Generators (CPGs) from [75], yielding a 27D controller for simulation experiments, 11D for hardware experiments. Figure 3.6 shows results of BO with an objective to walk forward. BO with SVAE-DC kernel found walking controllers in all 5/5 runs, yielding controllers that could walk up to 1.5m in 25sec. In contrast, uninformed BO with SE kernel found forward walking controllers only in 2/5 runs. Best controllers found by BO with SVAE-DC kernel were also able to walk on a different surface: an outdoor wooden patio.

Manipulation: ABB Yumi robot was used for the hardware experiments (Figure 3.7a). Additional simulation experiments were conducted with ABB Yumi and Franka Emika robot models. The manipulation task was to push two objects from one side of the table to another without tipping them over. Compared to 'push-to-target' task, this task had two different challenges. The objects were likely to come into contact with each other (not only the robot arm). Moreover, they could easily tip over, especially if forces were applied above an object's center of mass. Reward was given only at the end of the task: the distance each upright object moved in the desired direction minus a penalty for objects that tipped over: $f(\boldsymbol{x}) = \sum_i \big[(y^{obj_i}_{final} - y^{obj_i}_{start})\mathbb{1}_{obj_i \in Up} - y_{max}\mathbb{1}_{obj_i \in Tipped}\big]$, with $y_{max} =$ table width.

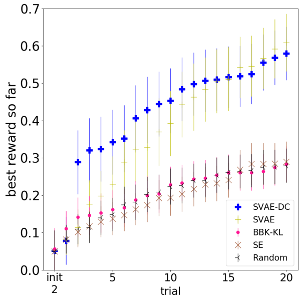
Two types of controllers were tested: 1) joint velocity controller suitable for robots like ABB Yumi and 2) torque controller suitable for robots like Franka Emika. These yielded 42D and 48D parametric controllers (see Section 7 in [5] for details). Simulated trajectories for learning SVAE-DC kernel contained robot joint & object poses at each time step (1K steps per trajectory). A step t on a trajectory $\boldsymbol{\xi}$ was marked 'undesirable' $\big(G_{bad}(\boldsymbol{\xi}_t) = 1\big)$ if: any object tipped over or was pushed off the table; robot collided with the table; the end effector was outside of main workspace.

ABB Yumi robot available to for these experiments could operate effectively only at low velocities ($\frac{1}{5}$ of simulation maximum). Hence, high-velocity trajectories

(a) 'Stable push' task with ABB Yumi robot at KTH



(b) BO on Yumi hardware (5 runs, 90% CIs)



(c) BO on Yumi simulation (50 runs, 90% CIs)

Figure 3.7: Experiments for a non-prehensile manipulation task to evaluate BO-SVAE-DC.

successful in simulation yielded different results on hardware. To prevent Yumi from shutting down due to high load, the execution was stopped if the robot's arm extended too far outside the main workspace or if it was about to collide with the table (yielding $-2y_{max}$ reward in such cases). These factors caused a large sim-real gap. Nonetheless, BO with SVAE-DC kernel was still able to significantly outperform BO with SE (Figure 3.7b). Even when controllers successful in simulation yielded very different outcomes on hardware, BO with SVAE-DC kernel was still able to find well-performing alternatives (more conservative, yet successful on hardware).

Further experiments in simulation were performed to analyze SVAE-based kernel without dynamic compression, compare to BBK-KL (an approach that used KL between original trajectories $\boldsymbol{\xi}$) and compare to using Matèrn kernel function for BO. The 'sim-to-real' gap was emulated by sampling different object properties (mass, friction, restitution) at the start of each BO run. Results in Figure 3.7c show that BO on Yumi with SVAE-DC kernel yielded substantial improvement over all baselines. BO in the latent space of SVAE (without dynamic compression) was also able to substantially outperform all baselines. See Section 7 in [5] for further details and plots for Franka Emika simulation experiments.

3.4 Alternation Kernel Robust to Negative Transfer

As discussed in Section 2.1, one danger of using low-fidelity simulations for sim-to-real is that this could cause negative transfer. This section presents a variant of BO that alternates between using informed and uninformed kernels[4]. This Bernoulli Alternation Kernel (BAK) ensures that discrepancies between simulation and reality do not hinder adapting robot control policies online during BO. The resulting approach, BO-BAK, is applied to a challenging real-world problem of task-oriented grasping with novel objects. With a large number of hardware experiments (>1.2K trials conducted on an ABB Yumi robot overall), this section of the thesis

[4]Work presented in the first part of this section was published by the thesis author in [6]. Some paragraphs reuse the text and figures from [6], which is permitted by CC-BY 4.0 license.

demonstrates[5] that BO-BAK can obtain large improvements over baseline BO when using imprecise simulations; moreover, BO-BAK can quickly recover from negative transfer even when the simulation-informed kernels are purposefully degraded.

Bayesian Optimization with Bernoulli Alternation Kernel

One way to avoid failures due to a corrupted or highly imprecise simulation-based kernel is to combine it with an uninformed kernel. For this, using a sum of kernels could seem appropriate at first. Let $k_{sum}(\boldsymbol{x}, \boldsymbol{x}') = k_{SE}(\boldsymbol{x}, \boldsymbol{x}') + k_\phi(\boldsymbol{x}, \boldsymbol{x}')$ be a kernel comprised of k_{SE} (Equation 1 in Section 2.2) and $k_\phi = k_{SE}(\phi(\boldsymbol{x}), \phi(\boldsymbol{x}'))$, where $\phi(\cdot)$ is akin to a warping function.

Recall that $\boldsymbol{x}$ denotes a vector of control parameters. $\phi(\boldsymbol{x})$ can be obtained by executing controls $\boldsymbol{x}$ in simulation and outputting relevant characteristics of the result. It is useful to embed ϕ into the kernel, as demonstrated in Section 3.1. This can help collapse the space of unsuccessful controls: i.e. failed points/controls could be similar to each other, but dissimilar from successful regions of control parameters. Section 3.1 demonstrates this for locomotion trajectory summaries, but a similar effect can be achieved in other areas of robotics. For example, if $\boldsymbol{x}$ points represent grasping controllers, then $\phi(\boldsymbol{x})$ could output grasp stability scores computed in simulation. In this case, the failed points/controls would correspond to those with poor stability scores. BO can then quickly learn to neglect the non-promising regions, even if they are far away from each other in the original space of control parameters.

Section 4.1 in [6] demonstrates that k_{sum} could be adversely impacted by the k_ϕ component if ϕ fails to provide high-quality information. To offer a more robust alternative, this thesis work proposes BO with Bernoulli Alternation Kernel (BO-BAK), which was first presented in Section 3 in [6] and is summarized here. BO-BAK takes contributions from both k_{SE} and k_ϕ, but ensures that BO cannot be permanently damaged by a poor choice of ϕ. At each BO iteration/trial, the choice of whether k_{SE} or k_ϕ is used is randomized. For this, BO-BAK defines a probability distribution over kernels and draws a kernel function to be used at each BO trial independently. Hence, k_{BAK} kernel is defined by the following sampling approach:

$$k_{BAK}(\boldsymbol{x}, \boldsymbol{x}') \sim \mathbb{1}_{\{\theta \leq 0.5\}} k_{SE}(\boldsymbol{x}, \boldsymbol{x}') + \mathbb{1}_{\{\theta > 0.5\}} k_\phi(\boldsymbol{x}, \boldsymbol{x}'); \quad \theta \sim Uniform(0, 1) \qquad (14)$$

Note that after each iteration/trial n, the data for computing the GP posterior consists of all the points sampled so far: $\boldsymbol{D}_n = \{(\boldsymbol{x}_i, f(\boldsymbol{x}_i)) | i = 1, ..., n\}$. In BO, posterior is usually re-computed after each new sample. This aspect of BO allows to make a choice of the kernel function for each iteration separately. So, after n iterations/trials, BO-BAK first picks a kernel function k_n using Equation 14, then computes GP posterior mean and covariance in a standard manner. For this, Equation 2 from Section 2.2 is used with $\boldsymbol{y}$ being a vector of evaluations for the sampled points: $[\boldsymbol{y}_n]_i = f(\boldsymbol{x}_i)$. Algorithm BO-BAK gives a concise summary.

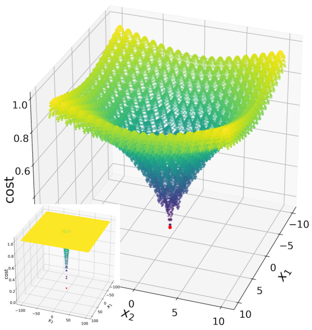
[5]Experiments presented at the end of this section are newer, hence are not in [6].

For an intuitive insight, note that points sampled for trials when k_ϕ is used could provide fast guidance towards useful parts of the search space. These points are included in the data for subsequent posterior computations, enabling even trials in which k_{SE} is used to propose better next choices. This is important if probability of discovering a promising region is small, which is frequent in robotics. If k_ϕ is misleading, choices made on the trials that use k_{SE} are not impacted by poor ϕ, since with BO-BAK the acquisition function makes its decisions using only one of the kernels at a time.

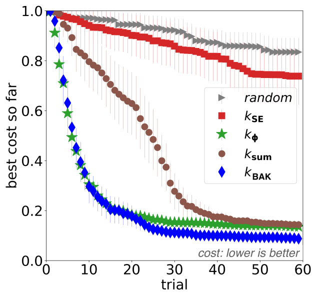
Algorithm BO-BAK

```
sample $\boldsymbol{x}_1$ randomly, get $y_1 = f(\boldsymbol{x}_1)$ from real world
initialize: $\boldsymbol{D}_1 = \{(\boldsymbol{x}_1, y_1)\}$
for $n = 1, 2, ...$ do
    sample kernel function $k_n$ using Equation 14
    get posterior GP mean & cov using $\boldsymbol{D}_n$ & Eq.2
    select $\boldsymbol{x}_{n+1}$ by optimizing acquisition function:
        $\boldsymbol{x}_{n+1} = \arg\max_{\boldsymbol{x}} \alpha(\boldsymbol{x}; \boldsymbol{D}_n)$
    get $y_{n+1} = f(\boldsymbol{x}_{n+1})$ from real world
    augment data $\boldsymbol{D}_{n+1} = \boldsymbol{D}_n \cup \{(\boldsymbol{x}_{n+1}, y_{n+1})\}$
```

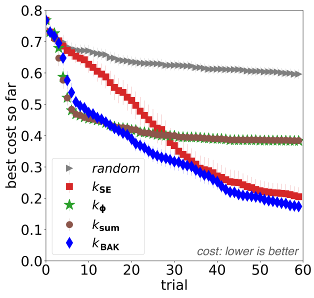
Figure 3.8 shows comparisons of BO variants on 2D and 10D version of a synthetic benchmark used as a challenging test case – the Ackley function [76]: $f_{AC}(x) = \text{-}a \cdot \exp\left(\text{-}b\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(cx_i)\right) + a + \exp(1)$, with $a = 20, b = 0.2, c = 2\pi, x \in \mathbb{R}^d$. The following ϕ emulates a case where simulation could provide coarse guidance, but also could be limiting when searching for the global optimum: $[\phi(x)]_1 = \text{-}b\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}$; $[\phi(x)]_2 = \frac{1}{d}\sum_{i=1}^{d}\cos(cx_i)$. ϕ has components that capture information from two first addends of the Ackley function. This gives a kind of 'collapsing' of the relevant features, akin to simulation that aims to capture most salient features needed to approximate real-world output. Figure 3.8b shows that in 2D using ϕ gives a significant advantage to all informed



(a) Ackley function



(b) Ackley 2D on [-100, 100]



(c) Ackley 10D on [-10, 10]

Figure 3.8: Ackley function presents challenges similar to those in optimization for robotics: shallow local optima, small low cost region and sharp cost changes, deep local optima. Left: Ackley function in 2D, with values (costs) normalized to $[0, 1]$; a small insert shows Ackley on a larger domain $[-100, 100]$: only 1% of the space is < 0.9. Middle and right: plots of BO on 2D and 10D versions of Ackley (mean over 40 runs for each kernel type, 95% CIs).

kernels: k_ϕ, k_{sum} and k_{BAK}. When the low-cost region is small, the performance of uninformed BO with SE kernel degrades to random search, while the informed versions get close to the optimum with <40 trials. However, the hint of limitations induced by using ϕ is already visible: for k_ϕ and k_{sum} the improvement stagnates after 50 trials. This stagnation is more striking in 10D (Figure 3.8c). There, k_ϕ and k_{sum} stagnate at a high cost. In contrast, k_{BAK} outperforms both informed and uninformed kernels. See Section 4.1 in [6] for other illustrations with synthetic benchmarks.

The simple design of BO-BAK makes make it easy to suggest that the method retains the optimality guarantees of the conventional BO. Intuitively, this is because in expectation BO-BAK performs $n/2$ trials using only SE to choose the next point. This is no worse than using conventional BO with half of the trials, and it is easy to anticipate that satisfactory regret bounds could be easily obtained as well. In future work it would be interesting to investigate whether the guarantees can hold for an adaptive method, where θ from Equation 14 is adapted based on the estimates of the severity of the sim-to-real gap. Preliminary investigations suggest that such guarantees would be difficult to obtain without making restrictive assumptions regarding simulator fidelity. Hence, BO-BAK might ultimately provide the best balance, offering theoretical simplicity and practical benefits.

Task-oriented Grasping Experiments with BO-BAK

This section describes evaluation of BO-BAK on a real-world setting of task-oriented grasping with an ABB Yumi robot. First, the setup and experiments from [6] are summarized. Then, a new set of hardware experiments to illustrate robustness to negative transfer are presented.

For the offline (kernel construction) part: a CNN was trained on a large number of simulated grasps (605 objects, 4.5K gasps for each) to output grasp stability & task suitability scores, given a desired task, a voxelized representation of an object, and grasp parameters. Grasp parameter vectors $\boldsymbol{x}$ were comprised of: a desired point $\boldsymbol{p}$ to approach (on the object's surface), a surface normal vector, gripper roll & distance to $\boldsymbol{p}$. Grasp stability scores were computed (by OpenRave [77] simulator) and included metrics commonly used in the literature [78]. Hence, $\phi(x)$ for the informed part of the BO-BAK kernel was comprised of the CNN output; the uninformed part was the standard SE kernel.

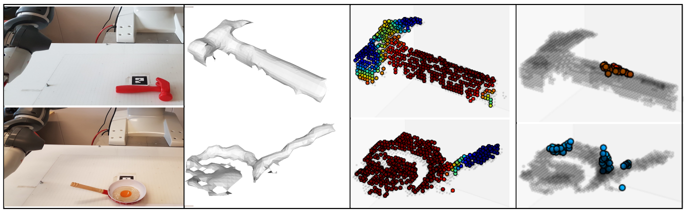


Figure 3.9: Left to right: object in the workspace; partial mesh; task scores ζ; 'CoM' stability metric for top 100 grasps with $\zeta > 0.5$.

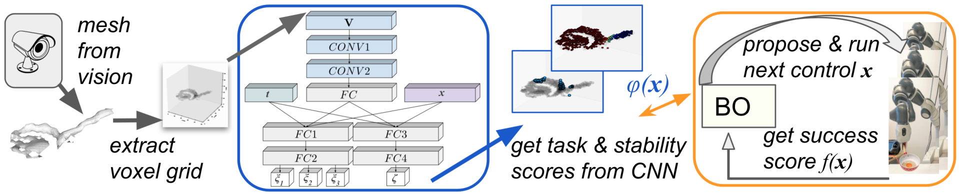


Figure 3.10: An overview of using BO-BAK for task-oriented grasping. Blue highlights components trained offline; orange highlights online learning components that guide real-time decisions made by the robot. A vision system is used to first construct a mesh from raw point cloud. Then, a set of grasping points, voxelized mesh and task identity are passed through the network to obtain task suitability and grasp stability scores. These are used to construct informed kernel for BO. Then, the robot executes online search with BO to find the best task-appropriate grasp. Video of the hardware setup and experiments: `https://youtu.be/MlCabBaYw7E`.

For the online (BO) part: a set of small everyday objects from seven categories aimed for six different tasks was used for the hardware experiments. The selection of objects was constrained by limitations of the robot: maximum payload of 500g (lower in practice), rigid plastic parallel gripper, no force-torque or tactile sensing. At the start of each trial an object was placed on the table and Microsoft Kinect camera was used to get a dense point cloud of the scene and segment out the object. Then, a mesh representation was generated from the partial point cloud. It was then voxelized, and together with grasp and task representation fed through the CNN to get stability and task suitability score estimates (see Figure 3.9 for a visualization). These were utilized for the informed part of the BO-BAK kernel (k_ϕ). Then, BO-BAK would suggest the next controller $\boldsymbol{x}$ to execute and MoveIt [79] was used for motion planning to execute the grasp. The grasp execution was assigned a result $\in [0, 4]$, i.e. the objective $f(\boldsymbol{x})$. Low values indicated MoveIt planning failures, mid-range values indicated failures to grasp an object in a way that would be appropriate for the task, the highest value indicated that the grasp was stable and appropriate for the task. After that, BO would suggest a controller for the next trial, and so on. Figure 3.10 gives an overview.

Figure 3.11 shows BO on objects with a large sim-to-real mismatch. These were identified by first executing top 3 grasp approaches suggested by simulation. This involved generating 4.5K grasps randomly, obtaining estimates for their task suitability and grasp stability scores from the CNN; then retaining only those with task suitability $\zeta > 0.5$ and executing top 3 with highest stability score predictions. For the objects where the grasps failed using this 'Best 3 sim' approach: BO was ran for the further 10 trials. Figure 3.12 summarizes these results, showing

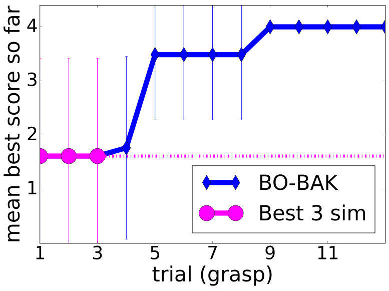
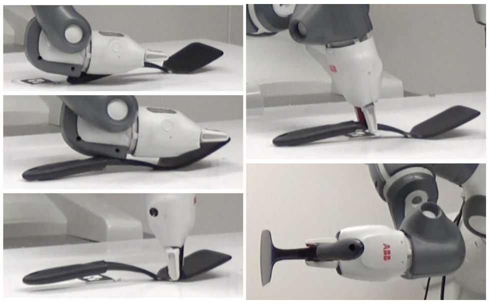


Figure 3.12: BO-BAK for challenging objects, where 'Best 3 sim' grasps failed (mean over 6 runs, 90% CIs).


(a) Left: top choices for handover task from 3 stability metrics; Right: the next BO trial.

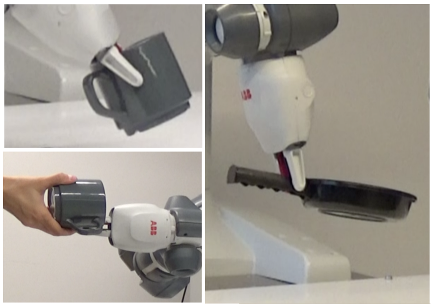

(b) Left: A BO trial for mug handover task. Right: A BO trial for pan support task.

Figure 3.11: BO on challenging objects, where grasps selected using stability score estimates returned by CNN failed (i.e. testing BO-BAK robustness when simulation-informed kernel quality would be poor either due to a larger sim-to-real gap or due to NN approximation).

quick significant improvement of BO over the initial top choices from the 3 stability metrics. Qualitatively, the benefits we observed from using BO were: 1) exploring various parts of the object systematically and efficiently; 2) sampling a variety of successful controllers that further improve over a merely acceptable controller.

To show ability of BO-BAK to recover from negative transfer, this thesis presents new experiments that involve further BO 27 runs with various object-task pairs (hardware setup is same as the experiments from [6] summarized above). Figure 3.13 shows 3 lines: the black line illustrates the performance of baseline (uninformed) BO with SE kernel (BO-SE does not use any simulation information). The blue line shows the performance of BO-BAK after executing 'Best 3 sim' grasps (selection procedure for these was described in the previous paragraph). While 'Best 3 sim'

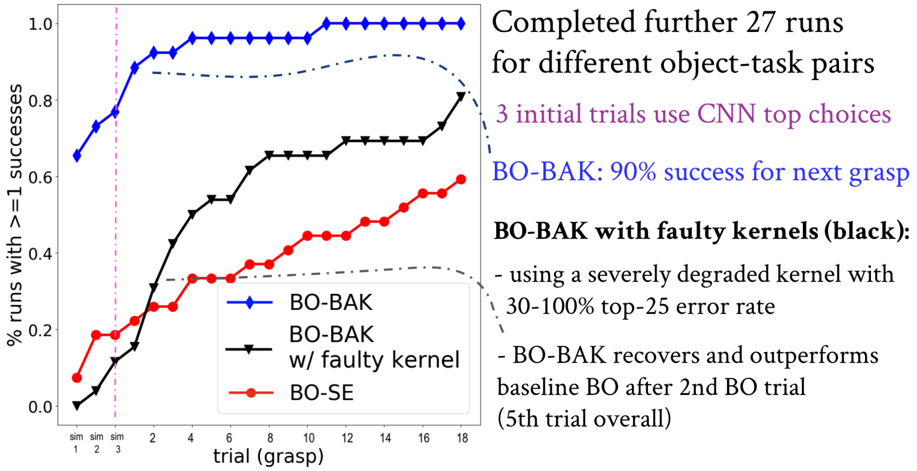


Figure 3.13: Experiments for: BO-BAK using simulation-based k_ϕ (blue line); BO-BAK with a faulty kernel (black line); BO-SE that does not use any simulation data (red line).

is enough for many object-task pairs to obtain a successful grasp, BO-BAK is instrumental in quickly improving this further, yielding 100% success rate after further 8 trials. The black line in Figure 3.13 shows the key result: ability of BO-BAK to recover from negative transfer. Here, BO-BAK is given a faulty kernel for the k_ϕ part. This is done by degrading k_ϕ used in the earlier experiments by introducing severe noise. After this degradation, top 25 grasps selected by k_ϕ have 30-100% error rate. This is computed by attempting to execute 'top 25' grasps for each object, which is feasible because many suggested grasps now fail at the MoveIt planning stage, with fewer full executions needed on the robot. As intended, this degradation causes negative transfer: the black line is below the red line for trials 1-4, which implies that simulation-based trials are worse than uninformed BO. However, on the 5th trial BO-BAK recovers from this negative transfer (black line above red line). Since trials 1-3 execute top choices from CNN, 5th trial overall means 2nd trial suggested by BO-BAK, which is notable. Furthermore, BO-BAK proceeds to benefit from the faint simulation-based signal that remains in the degraded kernel and ultimately achieves a significant gain over uninformed BO: >80% success rate using a degraded k_ϕ vs <60% with uninformed BO.

Chapter 4

Variational Alignment for Sim-to-Real

This Chapter presents a novel way to use simulators as regularizers[1]. The approach leverages a small amount of hardware data to convert a given deterministic simulator to an approximate stochastic model, hence the name: DET2STOC. This is achieved by regularizing a decoder of a variational autoencoder to a black-box simulation, with the latent space bound to a subset of simulator parameters. This yields a data-efficient way to align the result with hardware observations. In addition to producing flexible simulation parameter posteriors, this approach can be used to help Reinforcement Learning (RL) overcome the sim-to-real gap. For example, model-free RL policies can be fine-tuned using the resulting stochastic simulations. Hence, DET2STOC provides an alternative for those who prefer to work with NN-based model-free policies (instead of using structured parametric controllers discussed in the previous chapter).

4.1 The DET2STOC Algorithm

Modern general-purpose simulators can be fast, however, some aspects remain compute-intensive (e.g. contact-rich dynamics). Moreover, most simulators do not explicitly model uncertainty. In contrast, modern generative modeling incorporates uncertainty and uses fast neural networks. Hence, a hybrid solution is desirable: combining deterministic simulators with learnable, stochastic neural network models.

To construct a generative model one could train a CVAE using real data (recall $ELBO_{\text{CVAE}}$ from Equation 6 in Section 2.3. However, this is not data-efficient. Furthermore, approaches based on variational inference are prone to overfitting and mode collapse even with medium-to-large datasets. An option of using strong uninformed regularization penalties (e.g. increasing the weight of KL term) is

[1]Work presented in this section was published in [7]. Some paragraphs reuse the text from the accepted version of the article, which is permitted by the IEEE copyright terms.

problematic. It frequently prevents from learning sharp posteriors, and such over-regularized models fail to ensure enough precision for advance control policies.

DET2STOC utilizes simulators as 'informed' regularizers. Instead of starting from random encoder and decoder weights, the decoder function f^{dec} is aligned with the output of an existing general-purpose simulator $\mathfrak{f}^{sim}$. This implies a soft restriction on the class of models for the decoder neural network function. Hence, the search space is reduced to a subspace that aligns more closely with the class of functions captured by the simulator.

First, let's define notation:

$\boldsymbol{\psi}$: simulation parameters to infer (a subset of all simulator parameters available)

$\boldsymbol{\xi}^{sim}_{\boldsymbol{\psi}}$: simulated observations from a setting that roughly matches the target real-world setting

$\boldsymbol{\xi}^{real}$: real observations (e.g. positions and velocities of the robot joints and the relevant objects)

$\boldsymbol{\phi}_{\mu,\sigma}$: posterior produced by encoder activations

$\boldsymbol{\phi}_{mix}$: a learnable mixture 'prior' that can be used as aggregate posterior (instead of using encoder activations)

Algorithm DET2STOC

```
$\boldsymbol{\psi}_0 \leftarrow$ initialize sim parameters ; $\pi \leftarrow$ random policy
$\boldsymbol{\xi}^{real} \leftarrow$ get initial real trajectories using $\pi$
for $i = 0, 1, ..., n$ do
    $\boldsymbol{\xi}^{sim}_{\boldsymbol{\psi}_i} \leftarrow$ get sim trajectories from $\mathfrak{f}^{sim}_{\boldsymbol{\psi}_i}$ using $\pi$
    $f^{dec} \leftarrow$ pre-train decoder on $\boldsymbol{\xi}^{sim}_{\boldsymbol{\psi}_i}$
    $f^{enc} \leftarrow$ train CVAE on $\boldsymbol{\xi}^{real}$ (w. pre-trained $f^{dec}$)
    $\boldsymbol{\phi}_{\mu,\sigma} \leftarrow$ CVAE posterior $q_{\boldsymbol{\phi}}$   ($f^{enc}$ given $\boldsymbol{\xi}^{real}$)
    update posterior sim params: $\boldsymbol{\psi}_{i+1} \leftarrow \boldsymbol{\phi}_{\mu,\sigma}$
    if training RL then
        $\pi \leftarrow RL(\mathfrak{f}^{sim}_{\boldsymbol{\psi}_{i+1}})$
        $\boldsymbol{\xi}^{real} \leftarrow$ get more real trajectories using $\pi$
```

Algorithm DET2STOC gives an outline of the approach. First, an initial distribution for simulation parameters is chosen. This can be a uniform or a wide truncated Gaussian. $\boldsymbol{\psi}_0$ denotes the parameters of this distribution. Then, initial hardware trajectories $\boldsymbol{\xi}^{real}$ are collected. At the beginning of each DET2STOC iteration i, trajectories $\boldsymbol{\xi}^{sim}_{\boldsymbol{\psi}_i} = \{[..., \boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}, ...]\}^{1:N}$ are obtained by running simulations parameterized by $\boldsymbol{\psi}_i$. This constitutes the data for aligning decoder f^{dec} with simulation. The data for training f^{dec} is comprised of the tuples $(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1})$. The decoder learns a forward model: it is given $\boldsymbol{s}_t, \boldsymbol{a}_t$ as input and is trained to output $\boldsymbol{s}_{t+1}$. The training is done using a standard negative log likelihood as a supervised loss. This implies that f^{dec} is pre-trained to be aligned with simulation that corresponds to the current parameter posterior at each DET2STOC iteration. Then, variational posterior $q_{\boldsymbol{\phi}}$ is trained on $\boldsymbol{\xi}^{real}$ using $ELBO_{\text{DET2STOC}}$ defined below:

$$ELBO_{\text{DET2STOC}} = \log p_{\boldsymbol{\theta}}(\boldsymbol{s}_{t+1} \,|\, \boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{z}) - KL\big(q_{\boldsymbol{\phi}}(\boldsymbol{z} \,|\, \boldsymbol{s}_{t+1}, \boldsymbol{s}_t, \boldsymbol{a}_t) \parallel p_{\boldsymbol{\phi}_{mix}}(\boldsymbol{z})\big) \quad (15)$$

$ELBO_{\text{DET2STOC}}$ can be seen as an extension of $ELBO_{\text{CVAE}}$ aimed to support a learnable latent mixture prior/posterior $p_{\boldsymbol{\phi}_{mix}}(\boldsymbol{z})$. The parameters of $\boldsymbol{\phi}_{mix}$ are optimized via KL gradients from $ELBO$. At the end of each iteration, DET2STOC updates simulation parameter distribution $\boldsymbol{\psi}_{i+1}$ to match the learned prior/posterior $\boldsymbol{\phi}_{mix}$. Such learnable 'prior' still provides regularization, since it is informed by all of the data in aggregate (while the likelihood term aims to optimize likelihood for each point individually). An alternative is to use the real data accumulated so far as input to the encoder and compute the mean and standard deviation of activations: $\boldsymbol{\phi}_{\mu,\sigma}$. This approach works for the case of unimodal posteriors. Finally, in case DET2STOC is used to jointly train a control policy π (e.g. with model-free RL), π can be fine-tuned using simulation with the updated posterior $\boldsymbol{\psi}_{i+1}$.

A single iteration of DET2STOC can be enough to obtain a good initial fit, even with a small number of real trajectories. This high data efficiency arises because decoder gradients are informative immediately after the pre-training step. Hence, encoder can be trained from few real samples and still meaningfully shift the posterior. Further iterations are beneficial for more complex scenarios. For example, 1-20 iterations were ran for various experiments reported in [7]. These used DET2STOC in a 'batch' mode: first collecting a set of hardware trajectories, then using them for all iterations. DET2STOC can also be used in a fully incremental mode: updating the posterior as soon as new hardware data arrives, potentially even after every timestep.

4.2 Experiments on Benchmarks

Formally, DET2STOC takes a choice of a likelihood model $p(x|z)$. However, the decoder is parameterized by neural network weights θ, which are learned during optimization. This yields $p_\theta(x|z)$: a highly flexible function of the latent input z. As expressivity of the neural network increases, distributional assumptions become less consequential. Hence, comparison to likelihood-free approaches becomes relevant. BayesSim [21] is a very recent approach that offers likelihood-free and Bayesian treatment for the problem of estimating posterior for simulator parameters. It also provides a mixture posterior, unlike previous unimodal approaches, such as [17].

Table 4.1 provides a comparison to BayesSim [21] on a set of standard analytic RL benchmarks. To evaluate DET2STOC posterior, each environment setting is run 6 times with different 'true' parameters; log predicted probabilities (densities) are computed for each run. As in [21], we obtain 10 surrogate 'real' trajectories/episodes, 200 steps each. The environments and parameter ranges are taken exactly as in [21] to enable the comparison. The results show that DET2STOC offers significantly sharper densities than the previously proposed methods. It recovers the true parameters with ease on the standard benchmarks of CartPole, Pendulum and Mountain Car. On the more challenging Acrobot benchmark, DET2STOC significantly outperforms both variants of BayesSim using the same number of mixture components.

Problem	Parameter	BayesSim RFF	BayesSim NN	DET2STOC k=1	DET2STOC k=5
CartPole	pole length	-0.609±0.39	-0.657±0.25	**4.0284±0.6269**	3.1474±0.9581
	pole mass	0.973±0.26	0.633±0.52	**4.2682±1.1935**	3.4865±0.7018
Pendulum	dt	3.192±0.30	3.199±0.17	**7.0521±0.3946**	3.7471±2.9817
Moutn Car	power	3.863±0.52	3.901±0.2	**6.9967±0.574**	6.8696±0.2072
Acrobot	link mass 1	2.046±0.37	1.331±0.22	1.0218±0.1687	**2.2006±0.6511**
	link mass 2	0.321±1.85	1.513±0.39	1.6456±0.1137	**3.2091±0.3015**
	link length 1	2.072±0.76	1.856±0.18	1.2212±0.2053	**2.5353±0.1816**
	link length 2	-0.148±0.19	-0.672±0.09	**0.1688±0.2601**	-0.1276±1.081

Table 4.1: Comparison of DET2STOC with likelihood-free methods. The results for BayesSim are from Table 1 in [21], where BayesSim is shown to outperform Rejection ABC and ϵ-Free methods. DET2STOC k=1 uses one mixture component; k=5 uses 5 (same as BayesSim). The table shows mean and standard deviation of log predicted probabilities (densities).

4.3 Hardware Experiments

For hardware experiments we used a dual-arm ABB Yumi robot (2x7DoF) and a Microsoft Kinect to track a CheezIt box object from the YCB dataset [80]. We investigated the ability of DET2STOC to identify mass distribution and friction. A heavy block-in-box insert was placed inside the CheezIt box. We varied the position of the block-in-box within the CheezIt box from one long side (the area with CheezIt logo) to the other. This gave center of mass (CoM) changes from -0.08m to 0.08m off-center from the middle of the CheezIt box. Varying CoM yielded significantly altered dynamics, especially when the block-in-box insert was placed inside at the CheezIt logo area (appears on the left when the box is lying on its side).

To measure friction we performed an inclined plane test to find approximate static and kinetic friction coefficients for several cardboard boxes and our table surface. The results fell in a rather wide range of [0.2, 0.4], with dynamic friction closer to 0.2. To experiment with alternative friction settings we padded a side of the CheezIt box with a thin foam pad to obtain a much higher friction.

We then investigated two key questions: whether DET2STOC could provide a reasonable alignment to real-world measurements and whether it could help a policy trained in simulation to recover from a severe sim-to-real gap.

Posterior Alignment

We obtained 10 trajectories from hardware episodes of Yumi moving and orienting the CheezIt box. We placed block-in-box insert inside to match the CheezIt logo area. The top left plot in Figure 4.1 shows that DET2STOC was able to correctly identify the shifted CoM. The friction range was also identified correctly: a wide

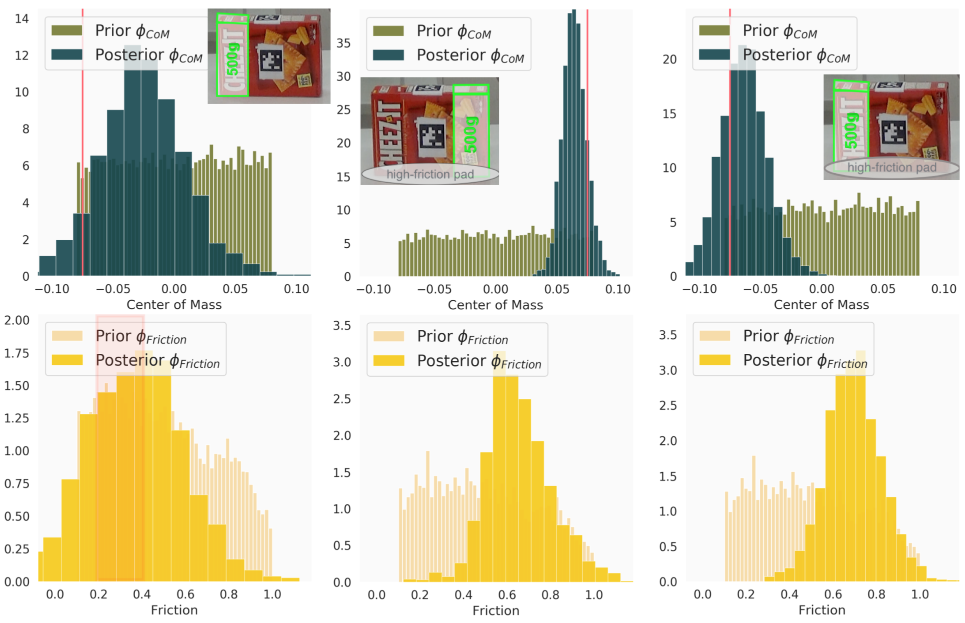


Figure 4.1: DET2STOC posterior learned from 10 hardware trajectories. Left: a heavy block-in-box insert was placed on the left (at CheezIt logo). Middle: block-in-box on the right; high friction. Right: block-in-box on the left left; high-friction. Top row: CoM; thin red line shows block-in-box insert placement. Bottom row: friction; parameter range from real-world estimates shown as a shaded pink region for the low-friction condition (left).

distribution with a mode covering the range we obtained from our measurements. Middle and right plots in Figure 4.1 show results for the high-friction setting. Top plots show that DET2STOC could identify the CoM changes, bottom plots show that higher friction was also detected correctly.

Overall, DET2STOC was able to recover the latent parameter means and variances well. One limitation of our setup was the fact that we used a single combined value to describe friction. The high variance for our manual real-world estimates indicates that a more granular description with several parameters and more advanced friction models would be beneficial.

Reinforcement Learning with DET2STOC Alignment

DET2STOC formulation could help control algorithms benefit from simulators and overcome the sim-to-real gap. The decoder learns a forward model $p(\boldsymbol{s}_{t+1} \,|\, \boldsymbol{s}_t, \boldsymbol{a}_t)$, which can be used for model-based RL and model predictive control (MPC). A forward pass on a neural network is one of the most optimized operations, and could support MPC-based methods for a wide range of problems. However, joint-space control of dual-arm systems could present a challenge for MPC methods. Since direct MPC on higher-dimensional action spaces (14D in our case) is currently not very common, the direction of combining DET2STOC with MPC is left for future

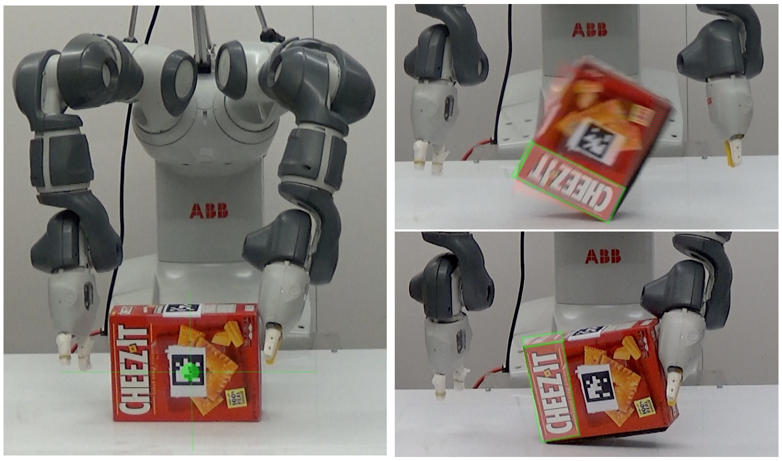


Figure 4.2: Left: Yumi moving and orienting CheezIt box to a goal pose using 'best PPO' policy, trained on simulation with medium friction & empty box. Top right: frequent failure mode when CoM is moved away from the pushing arm and the box is on a high-friction pad. Bottom right: same policy fine-tuned on simulation with parameter posterior from DET2STOC; the resulting policy adopts a more careful strategy to avoid flips.

work. Instead, we tested a currently common practice of training RL methods in simulation, then transferring RL policies to hardware.

We utilized two different classes of RL algorithms: on-policy PPO [81] and off-policy HER [82]. Using MuJoCo simulator [83] we performed several training runs on a range of simulation parameters that generally matched our task of moving & orienting a box to a desired goal position & orientation. We allowed for dual-arm policies to emerge, so both arms could be active. However, we did not use any reward shaping for whether/when the arms should be engaged. This learning problem proved to be quite challenging. PPO and HER have been the leading methods in recent RL-based sim-to-real works. Nevertheless, HER could not learn a useful policy. The commonly used practice of uninformed domain randomization also proved detrimental for both HER and PPO. PPO was able to learn a number of successful policies when trained on non-randomized parameters. It was especially successful on 'empty box' setting, with a low friction set to 0.1 in simulation. Several policies trained on this setting transferred well to hardware. One of these policies proved particularly versatile, we refer to it as 'best-PPO' policy below.

We ran 'best-PPO' policy on the hardware setting that matched its simulated training data (similar friction coefficient, same box mass). The policy was able to reliably move and orient the box to the goal. The policy also performed well with a heavier box, as well as with a lower friction (a smooth plate attached underneath). On the latter setting the dual-arm aspect became prominent: if the left arm pushed the box beyond the goal, the right arm was able to re-position it to the target.

In the further challenging set of experiments we used a high-friction pad and moved the block-in-box insert (an hence the CoM of the box) to the 'CheezIt' logo area. 'Best-PPO' policy was not successful on this setting: it either tipped the box, or missed the box at the start (went past the box but failed to move it), getting 0% success rate on the test runs on hardware (see Table 4.2).

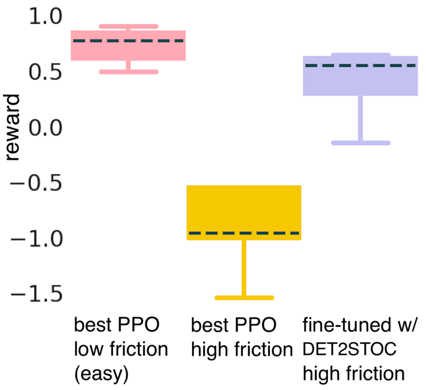


Figure 4.3: Closing the sim-to-real gap.

To evaluate the capability of DET2STOC for helping RL handle this severe sim-to-real gap we fine-tuned 'best-PPO' using CoM identified by DET2STOC. We performed 20 test runs of the resulting policy on hardware and found that the adjusted policy could move the box close to the goal position and orientation in 60% of the episodes for this very challenging setting. Table 4.2 summarizes this comparison. Figure 4.3 shows that using DET2STOC helped RL recover most of its performance in terms of reward as well. The above result was achieved by using only 10 hardware trajectories as training data for DET2STOC. The rest of the data for training PPO came from simulation. This is notable, since it demonstrates that a dual-arm system could be trained to make a good progress on a challenging version of a given task with only ≈5 minutes of hardware data collection. As simulation-based computation becomes less and less expensive, this opens new possibilities for faster learning and recovery from the sim-to-real gap.

Result state	'best PPO'	'best PPO' fine-tuned w/ DET2STOC
a) flipped	17	6
b) missed on start	3	2
c) close to goal	0	12

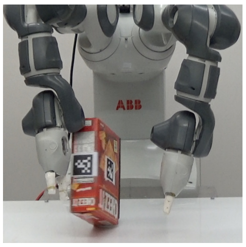
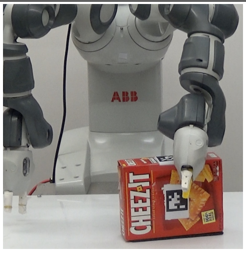
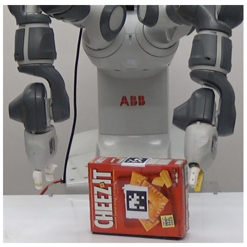
a)  b)  c) 

Table 4.2: Results comparing 'best PPO' policy before and after fine-tuning using simulation adjusted with DET2STOC posterior. Video of experiments: `https://youtu.be/zgaAEJf9Oc4`

Chapter 5

Analytic Manifold Learning for Modular Latent Space Transfer

Recent popularity of unsupervised learning methods, such as VAEs, yields a promise of automatically learning low-dimensional representations from high dimensional observations (e.g. RGB images, point clouds). However, the success of these approaches has been, in large part, shown either on dataset-oriented benchmarks or on visually simple domains. This presents a significant challenge for making such methods applicable to robotics. The shortcomings of the dataset-oriented view have been recently pointed out by some of the prominent robotics researchers [84]. Work conducted for this thesis also revealed a set of challenges when VAE-based approaches attempt to learn from non-stationary steams generated during RL training [8]. These shortcomings have negative implications for sim-to-real: methods that cannot handle distribution shift (simulated data $\rightarrow$ real data) would not be successful in quickly closing the sim-to-real gap.

Prior works attempted to address some of these challenges by manually constructing and imposing known/desirable relationships on the structure on the latent space, the survey in [61] gives an overview. In the context of sim-to-real, this could help retain latent space properties learned from ample data in simulation during the hardware adaptation/transfer stage. However, it would be tedious and error prone to construct a comprehensive set of heuristics that would capture the desired properties of the latent space for various domains and tasks. This chapter proposes Analytic Manifold Learning (AML) that can discover such relations automatically.[1] AML learns to encode the dynamics properties of simulated/source domains in a set of independent analytic relations that hold on sequences of low-dimensional simulation/latent states. We formalize the notion of learning non-linearly indepen-

[1]The work presented in this chapter is described in [8]. This chapter starts by presenting AML using the text and figures from sections 3 & 4 in [8], then provides extended explanations for the points most relevant to the main themes of this thesis. For further mathematical background, proofs and prior work: the readers are encouraged to see Appendix B in [8].

dent relations, without imposing restrictive simplifying assumptions or requiring domain-specific information. Defining non-linear independence rigorously allows us to obtain a valid modular representation and provide guarantees that each relation captures a new aspect of the dynamics. Relations learned by AML can be imposed on the latent space when learning on the target domain (e.g. reality) to improve data efficiency and transfer desirable latent space properties. This formulation enables a general and flexible way of shaping the latent space.

5.1 Motivation for Learning Latent Relations

Let x_t denote a high-dimensional (observable) state at time t and s_t denote the corresponding low-dimensional or latent state. x_t could be an RGB image of a scene with a robot & objects, while s_t could contain robot joint angles, object poses, and velocities. Consider an example of a latent relation: the continuity (slowness) principle [85, 86]. It postulates continuity in the latent states, implying that sudden changes are unlikely. It imposes a loss $L_{cont}(\mathcal{D}_x, \phi) = \mathbb{E}\big[||s_{t+1} - s_t||^2\big]$, with $D_x{=}\{x_t, x_{t+1}, ...\}$ and encoder $\phi(x){=}s$. A related heuristic from [87] maximizes mutual information between parts of consecutive latent states. Such approaches may be viewed as postulating concrete latent relations: $g(s_t, s_{t+1}) = c_\epsilon$, where g is the squared distance between s_t and s_{t+1} for L_{cont}, and a more complicated relation for [87]. Ultimately, all these are heuristics coming from intuition or prior knowledge. However, only a subset of them might hold for a given class of domains. Moreover, it would be tedious and error-prone to manually compose and incorporate a comprehensive set of such heuristics into the overall optimization process.

We take a broader perspective. Let $g(\mathcal{D}_\tau){=}0$ define a relation that holds on a set of sequences $\mathcal{D}_\tau{=}\{\tau^{(i)}\}_{i=1}^M$. $\mathcal{D}_\tau$ could contain state sequences $\tau{=}[s_t, ..., s_{t+T}]$ from a set of source domains. We start by learning a relation g_1; then learn g_2 that differs from g_1; then learn g_3 different from $\{g_1, g_2\}$ and so on. Overall, we aim to learn a set of relations that are (approximately) independent, and we define independence rigorously. To understand why rigor is important here, recall the significance of the definition of independence in linear algebra: it is central to the theory and algorithms in that field. Extending the notion of independence to our more general nonlinear setting is not trivial, since naive definitions can yield unusable results. Our contribution is developing rigorous definitions of independence, and ensuring the result can be analyzed theoretically & used for practical algorithms.

5.2 Mathematical Formulation for Non-linear Independence

Let $\mathbb{R}^N$ be the ambient space of all possible latent state sequences τ (of some fixed length). Let $\mathcal{M}$ be the submanifold of actual state sequences that a dynamical system from one of our domains could generate (under any control policy). A common view of discovering $\mathcal{M}$ is to learn a mapping that produces only plausible sequences as output (the 'mapping' view). Alternatively, a submanifold can be specified by describing all equations (i.e. relations) that have to hold for points in the submanifold.

We are interested in finding relations that are in some sense independent. In linear algebra, a dependency is a linear combination of vectors with constant coefficients. In our nonlinear setting the analogous notion is that of *syzygy.* A collection of functions $\mathfrak{f}^{\ddagger}=\{f_1,...,f_k\}$ is called a *syzygy* if $\sum_{j=0}^{k} f_j g_j$ is zero. Observe that this sum is a linear combination of relations $g_1,...,g_k$ with coefficients in the ring of functions. If there is no syzygy $\mathfrak{f}^{\ddagger}$ s.t. $\sum_{j=0}^{k} f_j g_j=0$, then $g_1,...,g_k$ are independent. However, this notion of independence is too general for our case, since it deems any g_1,g_2 dependent: $g_1\cdot g_2 - g_2\cdot g_1 = 0$ holds for any g_1,g_2. Hence, we define *restricted syzygies.*

Definition 5.2.1 (Restricted Syzygy)**.** *Restricted syzygy for relations $g_1,...,g_k$ is a syzygy with the last entry f_k equal to -1, i.e. $\mathfrak{f} = \{f_1,...,f_{k-1},f_k=-1\}$ with $\sum_{j=1}^{k} f_j g_j=0$.*

Definition 5.2.2 (Restricted Independence)**.** *g_k is independent from $g_1,...,g_{k-1}$ in a restricted sense if the equality $\sum_{j=1}^{k} f_j g_j=0$ implies $f_k\neq -1$, i.e. if there exists no restricted syzygy for $g_1,...,g_k$.*

For $\mathfrak{f}=\{f_1,...,f_{k-1},f_k=-1\}$ we denote $\sum_{j=1}^{k} f_j(\tau)g_j(\tau)$ by $\mathfrak{f}(\tau,g_1,...,g_k)$. Using the above definitions, we construct a practical algorithm (Section 5.3) for learning independent relations. The overall idea is: while learning g_ks, we are also looking for restricted syzygies $\mathfrak{f}(\tau,g_1,...,g_k)=0$. Finding them would mean g_ks are dependent, so we augment the loss for learning g_k to push it away from being dependent. We proceed sequentially: first learning g_1, then g_2 while ensuring no restricted syzygies appear for $\{g_1,g_2\}$, then learning g_3 and so on. For training g_ks we use *on-manifold* data: τ sequences from our dynamical system. Restricted syzygies $\mathfrak{f}$ are trained using *off-manifold* data: $\tau_{off}=\{s_{off_t},s_{off_{t+1}},...,s_{off_T}\}$, because we aim for independence of g_ks on $\mathbb{R}^N$, not restricted to $\mathcal{M}$ (on $\mathcal{M}$ g_ks should be zero). τ_{off} do not lie on our data submanifold and can come from thickening of on-manifold data or can be random (when $\mathbb{R}^N$ is large, the probability a random sequence satisfies equations of motion is insignificant). Independence in the sense of Definition 5.2.2 is the same as saying that g_k does not lie in the *ideal* generated by $(g_1,...,g_{k-1})$, with *ideal* defined as in abstract algebra (see Appendix B.1 in [8]). Hence, the ideal generated by $(g_1,...,g_{k-1},g_k)$ is strictly larger than that generated by $(g_1,...,g_{k-1})$ alone, because we have added at least one new element (the g_k). We prove that in our setting the process of adding new independent g_ks will terminate (proof in Appendix B.1 in [8]):

Theorem 5.2.1. *When using Definition 5.2.2 for independence and real-analytic functions to approximate gs, the process of starting with a relation g_1 and iteratively adding new independent g_ks will terminate.*

If $\mathcal{M}$ is real-analytic (i.e. is cut out by a finite set of equations of type $h(\tau)=0$ for some finite set of real-analytic hs), then after the process terminates, the set where all relations $g_1, .., g_k$ hold will be precisely $\mathcal{M}$. Otherwise, the process will still terminate, having learned all possible analytic relations that hold on $\mathcal{M}$. By a theorem of Akbulut and King [88] any smooth submanifold of $\mathbb{R}^N$ can be approximated arbitrarily well by an analytic set, so in practice the differences would be negligible.

To ensure that each new relation decreases the data manifold dimension, we could additionally prohibit $g_1, ..., g_k$ from having any syzygy $\{f_1, ..., f_k\}$ in which f_k itself is not expressible in terms of $g_1, ..., g_{k-1}$. With such definition (below) we could guarantee that a sequence of independent relations $g_1, ..., g_k$ restricts the data to a submanifold of codimension at least k (Theorem 5.2.2, which we prove in Appendix B.1 in [8]).

Definition 5.2.3 (Strong Independence). *g_k is strongly independent from $g_1, ..., g_{k-1}$ if the equality $\sum_{j=1}^{k} f_j g_j = 0$ implies that f_k is expressible as $f_k = h_1 \cdot g_1 + ... + h_{k-1} \cdot g_{k-1}$.*

Theorem 5.2.2. *Suppose $g_1, \ldots, g_k$ is a sequence of analytic functions on B, each strongly independent of the previous ones. Denote by $\mathcal{M}_{\mathring{B}} = \{x \in \mathring{B} | g_j(x) = 0 \text{ for all } j\}$ the part of the learned data manifold lying in the interior of B. Then dimension of $\mathcal{M}_{\mathring{B}}$ is at most $N - k$.*

In addition, we construct an alternative approach with similar dimensionality reduction guarantees, which ensures that the learned relations differ to first order. For this we use a notion of independence based on *transversality*, with the following definition and lemmas (with proofs in Appendix B.1 in [8]):

Lemma 5.2.1. *Dependence as in Definition 5.2.2 implies $\nabla_\tau g_k$ and $\nabla_\tau g_1, ..., \nabla_\tau g_{k-1}$ are dependent.*

Definition 5.2.4 (Transversality). *If for all points $\tau^{(i)} \in \mathcal{M}$ the gradients of $g_1, .., g_k$ at τ, i.e. $\nabla_\tau g|_{\tau^{(i)}}$, are linearly independent, we say that g_k is transverse to the previous relations: $g_k \pitchfork g_1, ..., g_{k\text{-}1}$.*

Using transversality, we deem g_k to be independent from $g_1, ..., g_{k-1}$ if the gradients of g_k do not lie in the span of gradients of $g_1, ..., g_{k-1}$ anywhere on $\mathcal{M}$. With this, g_k that only differs from previous relations in higher-order terms would be deemed as 'not new'. This formulation is natural from the perspective of differential geometry. Let H_{g_j} be the hypersurface defined by g_j: the set of points where $g_j = 0$. Each $H_{g_1}, ..., H_{g_k}$ contains $\mathcal{M}$. If gradients of g_k are linearly independent from gradients of $g_1, ..., g_{k-1}$, then the corresponding hypersurfaces intersect transversely along $\mathcal{M}$.

Lemma 5.2.2. *For once differentiable $(g_1, .., g_k)$ s.t. H_{g_j}s are transverse along their common intersection H, this intersection H is a submanifold of $\mathbb{R}^N$ of dimension $N - k$.*

The notion of independence defined via transversality is infinitesimal and symmetric w.r.t. permuting g_ks. This is useful in settings where many relations could be discovered, because it is then better to find relations whose first order behavior differs. In cases where guaranteed decrease in dimension is not needed, using restricted syzygies could allow a flexible search for more expressive relations.

5.3 Learning Latent Relations with Neural Networks

Here we describe the algorithm with relations g_k and restricted syzygies $\mathfrak{f}$ approximated by neural networks. Each g is represented by a neural network (NN) that takes a sequence of latent/low-dimensional states $\tau = [s_t, s_{t+1}, ..., s_T], \tau \in \mathbb{R}^N$ as input. The output of g is a scalar. We use g to denote both the relation and the NN used to learn it. If g outputs 0 for on-manifold data, this implies g has learned a function $g(\tau) = 0$, which captures a relation between states of the underlying dynamical system. g is trained on minibatches of size b of on-manifold data points $\tau^{(i)}$ using loss gradients: $\nabla L = \sum_{i=1}^{b} \nabla_g \left[L(\tau^{(i)})\right]$, where ∇_g means gradient w.r.t NN weights of g. We need to make $g \to 0$ for on-manifold data, while avoiding trivial relations (e.g. all NN weights ≈ 0). Hence, in the loss we minimize $d_g(\tau) = \frac{|g(\tau)|}{\|v\|}$, where v is the gradient of g with respect to input points $\tau^{(i)}$: $v = \nabla_\tau(g)|_{\tau^{(i)}}, v \in \mathbb{R}^N$. The gradient norm $\|v\|$ is the maximal 'slope' of the linearization of g at τ, so $d_g(\tau)$ is the distance from τ to the nearest point where this linearization vanishes ($d_g(\tau) =$ height/slope = distance). Hence, $d_g(\tau)$ is a proxy for the distance from τ to the vanishing locus of g. This measure of vanishing avoids scaling problems (see Appendix B.2 in [8]). We also maximize $\log \|v\|$ to further regularize g. Equation 16 summarizes our loss for g:

$$L(g) = d_g(\tau) - \log \|v\| \ ; \quad d_g(\tau) = |g(\tau)| / \|v\| \ ; \quad v = \nabla_\tau(g)|_\tau \tag{16}$$

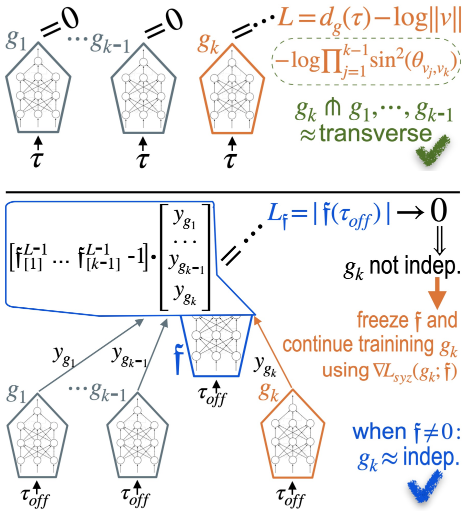
We proceed sequentially: first learn g_1, then g_2, and so on. This sequential approach has conceptual parallels with a functional Frank-Wolfe algorithm [89], but without convex optimization. Learning sequentially helps avoid instabilities, such as those that could arise from training flexible NN mixtures with EM [90].

Suppose that so far we learned (approximately) independent relations $g_1, ..., g_{k-1}$. We then keep their NN weights fixed and learn an initial version of the next relation g_k. To obtain g_k that is transverse to $g_1, .., g_{k-1}$ (Definition 5.2.4), we augment the loss as follows. We compute gradients of each $g_1, ...g_{k-1}$ w.r.t input τ. For example, for g_1 we denote this as $v_1 = \nabla_\tau(g_1)|_\tau$. Making g_k transverse to $g_1, ...g_{k-1}$ means ensuring that v_k is linearly independent of $v_1, ..., v_{k-1}$. We optimize a computationally efficient numerical measure of this: maximize the angles between v_k and all the previous $v_1, .., v_{k-1}$. Such measure encourages transversality of subsets of relations and strongly discourages small angles. Our overall measure of transversality is the product of sines of pairwise angles, with log for stability (Appendix B.3.1 in [8] gives further discussion):

$$L_{tr}(g_k) = d_{g_k}(\tau) - \log \|v_k\| - \log \prod_{j=1}^{k-1} \sin^2(\theta_{v_j, v_k}) \tag{17}$$

Algorithm AML

```
$\{\tau^{(i)}\}_{i=1}^{d} \leftarrow$ rollouts from RL actors
train $g_1$ with loss L from Eq.16
for $k = 2, 3, ...,$ do
    if aiming_for_transversality then
        train $g_k$ with loss $L_{tr}$ from Eq.17
    else // using syzygies
        train $g_k$ with loss $L$ from Eq.16
        for $j = 1, 2, ...,$ do
            generate $\tau_{off}, \tau_{off}^{test}$
            train $\mathfrak{f}_j$ with $L_{\mathfrak{f}} = |\mathfrak{f}_j(\tau_{off})|$
            if $\mathfrak{f}_j \neq 0$ on $\tau_{off}^{test}$ then
                break //$g_k \approx$indep.
            while $\mathfrak{f}_j(\tau_{off}^{test}) \approx 0$ do
                freeze $\mathfrak{f}_j$
                train $g_k$ with $L_{syz}$ (Eq.18)
```



$g_1 = 0$
$\cdots g_{k-1} = 0$
$g_k = \cdots L = d_g(\tau) - \log\|v\|$
$-\log\prod_{j=1}^{k-1}\sin^2(\theta_{v_j,v_k})$
$g_k \pitchfork g_1, \cdots, g_{k-1}$
≈transverse
τ
τ
τ
$[\mathfrak{f}_{[1]}^{L-1} \ldots \mathfrak{f}_{[k-1]}^{L-1} \text{-}1] \cdot [y_{g_1} \ldots y_{g_{k-1}} y_{g_k}] = \cdots L_{\mathfrak{f}} = |\mathfrak{f}(\tau_{off})| \to 0$
$\Downarrow$
g_k not indep.
freeze $\mathfrak{f}$ and
continue trainining g_k
using $\nabla L_{syz}(g_k; \mathfrak{f})$
$\mathfrak{f}$
y_{g_1}
$y_{g_{k-1}}$
y_{g_k}
g_1
$\cdots g_{k-1}$
g_k
τ_{off}
τ_{off}
τ_{off}
τ_{off}
when $\mathfrak{f} \neq 0$:
$g_k \approx$ indep.

Figure 5.1: Left: algorithm for learning latent relations. Top right: using transversality. Bottom right: training with syzygy $\mathfrak{f}$ to uncover if g_k is dependent, then using $\mathfrak{f}$ to modify g_k's loss. Orange & blue denotes NNs whose weights are being trained. Gray denotes learned relations whose NNs are frozen.

For independence based on Definition 5.2.2, we instead learn a restricted syzygy $\mathfrak{f}(\tau_{off}, g_1, ..., g_k) = 0$. Training data for $\mathfrak{f}$ is comprised of: 1) τ_{off} defined in Section 5.2 and 2) $y_{g_1}{=}g_1(\tau_{off}), ..., y_{g_k}{=}g_k(\tau_{off})$, i.e. outputs from $g_1, ...g_k$ with τ_{off} fed as inputs. $y_g s$ are passed directly to the next-to-last layer, which we denote as $\mathfrak{f}^{L\text{-}1} \in \mathbb{R}^{k-1}$. The last layer of $\mathfrak{f}$ computes a dot product of $[\mathfrak{f}_{[1]}^{L\text{-}1}, ..., \mathfrak{f}_{[k\text{-}1]}^{L\text{-}1}, \text{-}1]$ and $[y_{g_1}, ..., y_{g_k}]$. We use a simple L1 loss for training $\mathfrak{f}$. If $\mathfrak{f}$ outputs 0 at convergence: g_k is not independent. In this case, we freeze the weights of $\mathfrak{f}$ and continue to train g_k with augmented loss. We use gradients passed through $\mathfrak{f}$ to push g_k away from a solution that made it possible to learn $\mathfrak{f}$:

$$\nabla L_{syz}(g_k; \mathfrak{f}) = \nabla L(g_k) - \nabla_{g_k}\Big[\big|\mathfrak{f}(\tau_{off}, g_1, ..., g_k)\big|\Big] \qquad (18)$$

L_{syz} encourages adjusting g_k such that it makes the outputs of (frozen) $\mathfrak{f}$ non-zero. Once $L_{syz}(g_k; \mathfrak{f})$ is minimized, we can attempt to learn another syzygy $\mathfrak{f}_2$, and so on, until we cannot uncover any new dependencies. Then g_k can be declared (approximately) independent of $g_1, ...g_{k-1}$ and we can proceed to learn g_{k+1}. All g_ks, $\mathfrak{f}$s, Ls are in latent space, so networks are small & quick to train.

An additional benefit of our formulation is that prior knowledge can be incorporated without restricting the hypothesis space. g_ks can be pre-trained in a supervised way: to output values that a prior heuristic produces on- and off-manifold. Then, g_ks can be further trained using on-manifold data, and if prior knowledge is wrong, then g_k would move away from the wrong heuristic during further training.

5.4 Imposing AML Relations During Transfer

The previous section described how to encode the latent data manifold onto a set of analytic relations represented by neural networks. This section shows how to impose these relations into a latent space of a sequential VAE. The model for a basic sequential VAE could be defined as follows:

Generative model: $p(\tau, x_{1:T}) := p(s_{1:T})p(x_t|x_{<t}, s_{1:T})$

Approximate posterior: $q(\tau|x_{1:T}) := q(s_{1:T}|x_{1:T})$

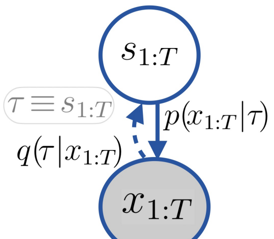


Figure 5.2: Basic sVAE

To model long-term dependencies it is customary to use recurrent neural networks (e.g. with LSTM, GRU, or other recurrent units). However, in case of non-stationary data these can slow down or even stagnate VAE's learning progress. Section 2.1 in [8] gives a detailed experimental analysis of this issue. Hence, a better choice is to use a convolutional encoder and decoder, with a one or more fully connected layers at the bottleneck.

Several recent works showed that a further improvement can be achieved by requiring to predict L next frames instead of only reconstructing the given frames [91, 92, 8]. Hence, we use a simple predictive version of a sequential VAE defined as follows:

PRED: a VAE that, given a sequence of frames $x_1, ..., x_T$, constructs a predictive sequence $x_1, ..., x_{T+L}$. First, the convolutional stack is applied to each x_t; then, the T output parts are aggregated and passed through fully connected layers. Their output constitutes the predictive latent state. To decode: this state is chunked into T+L parts, each fed into deconvolutional stack for reconstruction.

AML relations can be imposed on the latent state of *PRED* by augmenting the latent part of the loss as follows:

$$\mathcal{L}^{AML}_{PRED} = \mathbb{E}_{\substack{\tilde{\tau}_{1:T+L}\sim \\ q(\tau_{1:T+L}|x_{1:T})}} \Big[- \Big(\overbrace{\log p(x_{1:T+L}|\tilde{\tau}_{1:T+L}) - KL\big(q||\mathcal{N}(0,1)\big)}^{\text{standard } ELBO \text{ for } PRED \text{ version of } VAE} \Big) + \underbrace{\textstyle\sum_{k=1}^{K} \big|g_k(\tilde{\tau}_{1:T+L}, a_{1:T+L})\big|}_{\text{impose } AML \text{ relations}} \Big] \quad (19)$$

In the above, $\tilde{\tau}_{1:T+L}$ denotes a sample from the approximate posterior $q(\tau_{1:T+L}|x_{1:T})$. $p(x_{1:T+L}|\tilde{\tau}_{1:T+L})$ denotes the likelihood for *PRED*, with magenta color indicating that decoder outputs a predictive sequence $\hat{x}_{1:T+L}$ instead of a reconstruction $\hat{x}_{1:t}$. We would like to validate that imposing AML relations works well when the data is non-stationary, hence we train *PRED* on data stream generated during RL training. In this case, instead of learning relations on a subsequence of states, during AML training (described in the previous section) we learn relations on subsequences that include actions: $\tau = [s_1, a_1, s_2, a_2, ...]$.

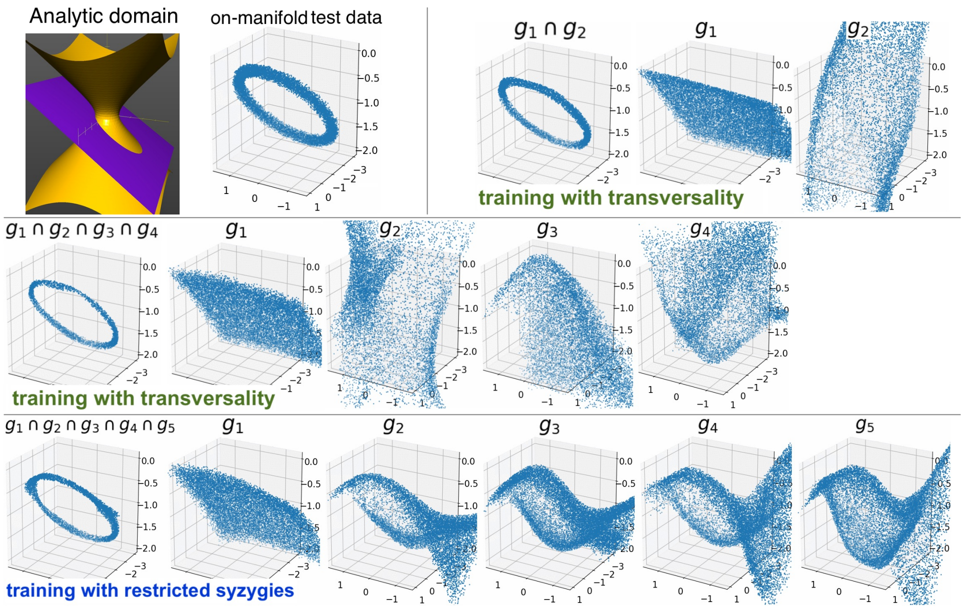


Figure 5.3: Top row: visualization of the analytic domain and noisy on-manifold data, followed by AML relations trained using transversality. First, $g_1 \cap g_2$ is shown: this is the intersection of the learned relations (i.e the intersection of the zero-level sets of g_1, g_2). The zero-level sets of individual relations are shown next. With transversality we get $g_1 \cap g_2$ as two simple relations: a plane and a hollow cylinder. Middle row: AML using transversality when we insist on getting more than 2 relations: $g_1 \cap g_2 \cap g_3 \cap g_4$ then includes smoothed cones. Bottom row: AML using restricted syzygies yields more advanced shapes.

5.5 Evaluating AML and Latent Space Transfer

We evaluate the proposed AML approach with 3 sets of experiments: 1) learning on an analytic domain and visualizing relations in 3D; 2) handling dynamics with friction and drag on a *block-on-incline* domain; 3) employing relations learned on a source domain to get better latent space properties on the *YCB-on-incline* as target.

Evaluating AML Training

Figure 5.3 visualizes the results on the analytic domain, for which on-manifold data comes from an intersection of a hyperboloid and a plane. It illustrates that using AML with transversality allows us to capture the latent data manifold using a small number of general relations, i.e. relations with simple shapes. The illustrations in 3D make it easy to see that these intersect transversely (top row), or attempt to maximize the angle of intersection (middle row). In contrast, relations found using syzygies have more complicated shapes and can be similar in some regions, as expected (bottom row). This could be useful when we need to avoid large changes, e.g. for fine-tuning or for flexible partial transfer using subsets of relations.

Next, we evaluate AML on a physics domain: a block sliding down an incline (Figure 5.4). The block is given a random initial velocity; gravity, friction and drag forces then determine its further motion. On-manifold data consists of noisy position & velocity of the block at the start and end of trajectories. Figure 5.5 shows results for AML with transversality. Appendix B.3 in [8] gives results with syzygies. Overall, these results show that AML can generalize beyond training data ranges and capture non-linear dynamics.

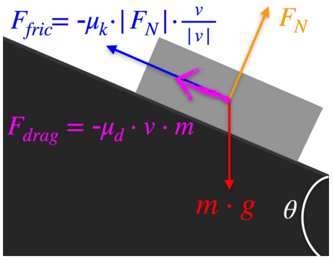

$F_{fric} = -\mu_k \cdot |F_N| \cdot \frac{v}{|v|}$
F_N
$F_{drag} = -\mu_d \cdot v \cdot m$
$m \cdot g$
θ

Figure 5.4: *block-on-incline*

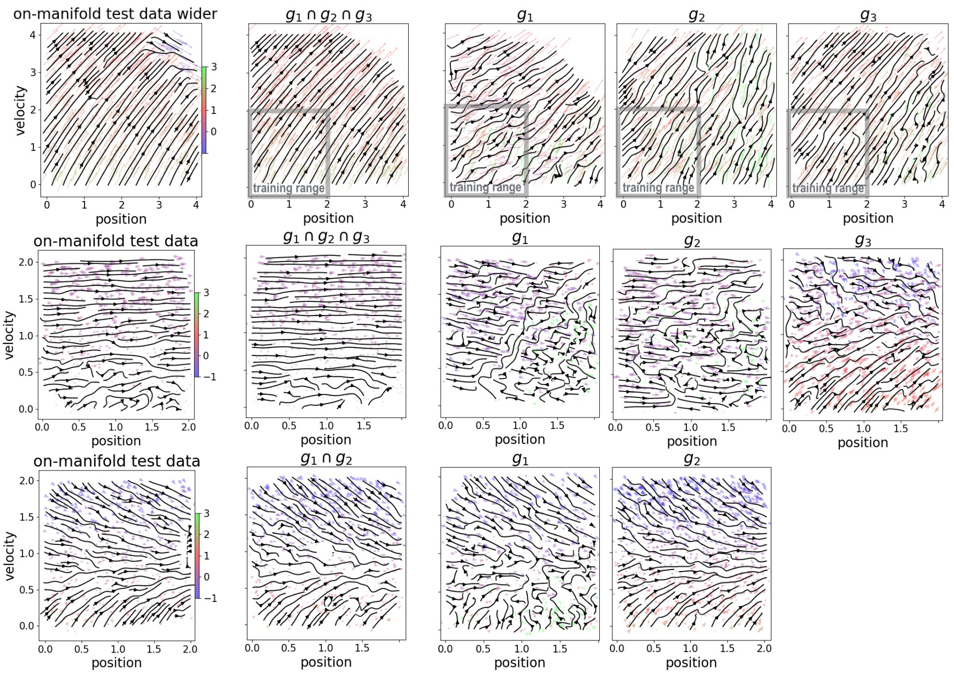


Figure 5.5: Phase space plots showing results for the *block-on-incline* domain. 1st column: on-manifold data; 2nd column: the learned manifold encoded by the intersection of AML relations trained with transversality; rest: zero-level sets of individual AML relations. Arrows show change in position & velocity after 1*sec* of sliding (scaled to fit). Top row: plots show the case of a 45° incline and demonstrate generalization. AML is only given training data with start position & velocity $\in [0, 0.2]$, but is able to generalize to $[0, 0.4]$. Middle row: high friction on a 35° incline. Bottom row: high drag on a 10° incline.

Latent Space Transfer

Emulating sim-to-real with sim-to-sim: In the final experiment we use a sim-to-sim setting that attempts to emulate challenges that arise in sim-to-real problems. We use a simulator with simple geometric shapes that move along an incline as a source domain. We refer to this source/simulation domain as *Geom-on-incline*. To emulate a 'real' domain, we use an advanced simulation involving objects with real mesh scans and RGB images as observations. We refer to this domain as *YCB-on-incline*, since it uses objects from the YCB dataset [80]. *YCB-on-incline* (visualized in Figure 5.6) yields realistic visual appearances and non-trivial object dynamics. The dynamics is dictated by meshes obtained from the 3D scans of real objects. Hence, there is a non-trivial sim-to-'real' gap between the dynamics of the simple shapes of *Geom-on-incline* domain vs realistic shapes of the *YCB-on-incline* domain. Furthermore, instead of building a dataset of RGB images, we train VAE-based learners directly on the stream of RGB frames that an RL agent generates during its own training. Hence, we ensure that the distribution of observations obtained on the target domain is non-stationary. As discussed in the introduction to this chapter: testing whether methods can handle non-stationary data is important for ensuring their applicability to robotics.

Measuring encoder distortion: One important quality measure of a latent space mapping is how much it distorts the true data manifold. For the experiment that runs on a sim-to-sim setting we have access to the low-dimensional states of the simulator of the target domain. We denote such sequences of low-dimensional simulation states as τ^{true}. We then quantify the distortion of the encoder map

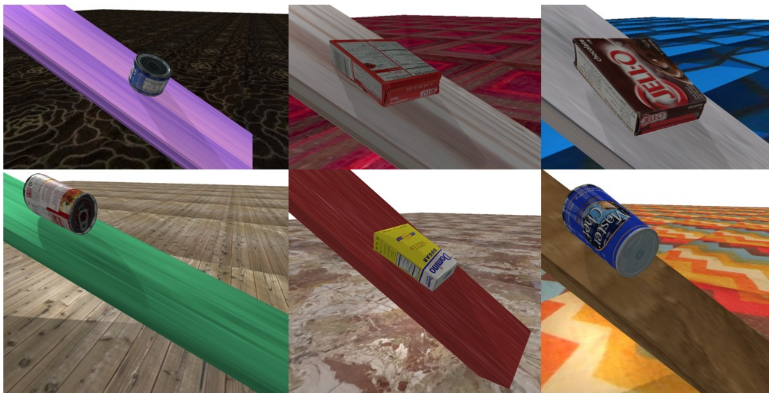


Figure 5.6: Visualizations of the *YCB-on-incline* domain. The textures of the ground and the incline plane are initialized randomly at the start of each episode. The sliding objects are the YCB boxes and cans. This domain is implemented in PyBullet simulator; it returns 64x64 RGB frames as observations; it defines a reward function for RL agents as: distance of the object to the center of the incline plane. This domain is a part of the evaluation suite available at `https://github.com/contactrika/bulb`. See [8] for further details.

(on 10K test points) as follows: we take pairs of low-dimensional representations τ_1^{true}, τ_2^{true} and the corresponding pixel-based representations x_1, x_2, then compute distortion coefficient $\rho_{distort}$ defined below.

$$\rho_{distort} = \log \frac{d_{L2}\big(\phi_{enc}(x_1), \phi_{enc}(x_2)\big)}{d_{L2}\big(\tau_1^{true}, \tau_2^{true}\big)} \tag{20}$$

Here, d_{L2} is the Euclidean distance. An encoder that yields low *variance* of these coefficients preserves the geometry of the low-dimensional manifold better (up to overall scale). This measure is related to approaches surveyed in [93, 94] (see Appendix B.3.2 [8]). The above evaluation could be done on an actual sim-to-real setting as well, if the real part is equipped with additional sensing. For example: if robot's joint angles and velocities are accurately reported; object positions and orientation are accurately estimated by a motion capture system. Such evaluation would be feasible in well-equipped labs. However, a sim-to-sim evaluation could be enough to compare the quality of the encoders produced by various algorithms, so doing this evaluation in an actual sim-to-real setting is not strictly required.

Latent Space Transfer with AML: In this experiment, we compare the results of imposing AML relations when training *PRED* on *YCB-on-incline* to two baselines: 1) *PRED* without AML relations imposed, and 2) a basic non-sequential *VAE*.

First, AML learns relations from *Geom-on-incline* domain. Incline angle, friction and object pose are initialized randomly. Actions are random forces that push objects along the incline. AML is given the incline angle, position & velocity at two subsequent steps, and the applied action. Note that, for the sim-to-real and sim-to-sim settings we have access to the simulator state of the source domains. Hence, we can use the low-dimensional state sequences to as training data for AML to learn the 'latent' data manifold of the source domain. For general cases that do not involve physics simulators: we could instead learn a VAE-based embedding on the source domain. This would yield the low-dimensional latents for learning AML relations on the source domain.

Then, we evaluate the latent space transfer to the target domain. For this, we train an unsupervised learner (*PRED*) on the target domain: *YCB-on-incline*. This domain is visualized in Figure 5.6. Recall that this sim-to-sim setup aims to emulate the challenges of sim-to-real transfer. In this case, *Geom-on-incline* plays a role of a simulator, while RGB frames from *YCB-on-incline* act as surrogates for 'real' observations. The distribution of the frames is non-stationary, since they are sampled using the current (changing) policy of the RL learner. We use PPO [81] to learn RL policies. The reward for RL is defined to be proportional to how close an object stays to the middle of the incline, i.e. RL has to learn to counteract gravity without pushing objects off the narrow incline plane. This task can be challenging, since several YCB objects we use have rounded shapes, causing them to easily roll off the sides of the incline plane.

We impose AML relations by extending the latent part of an ELBO-based loss as defined by Equation 19, explained in detail in the previous section. The left

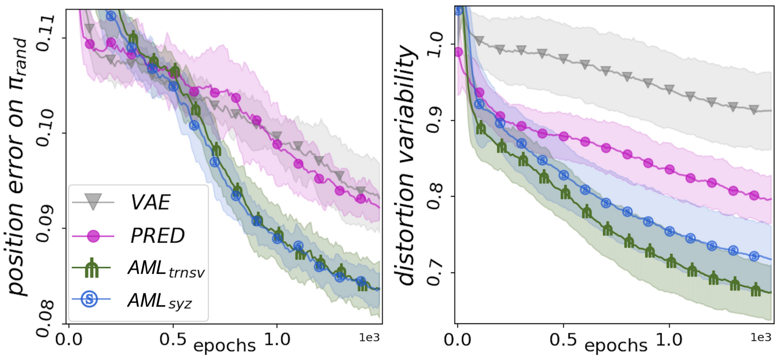


Figure 5.7: Results of imposing AML relations trained from *Geom-on-incline* domain when training *PRED* on the target *YCB-on-incline*. Left plot shows latent state alignment for object position computed using evaluation suite from [8]. The test observations for evaluation are obtained using a random policy π_{rand}. This is because we want to ensure that the alignment is good everywhere, not just in the state space regions that are visited often by the current RL policy. The results indicate that $\text{AML}_{\text{trnsv}}$ and AML_{syz} yield better alignment. The right plot in Figure 5.7 shows the distortion variability of the encoder map during training. It indicates that $\text{AML}_{\text{trnsv}}$ and AML_{syz} produce less distorting encoders that those obtained by than *VAE* and *PRED* without AML relations imposed.

plot in Figure 5.7 shows that the resulting $\text{AML}_{\text{trnsv}}$ (AML_{syz} when using syzygies) gets a better latent state alignment for object position, compared to *VAE* and *PRED* without AML relations imposed. Latent state alignment with the true simulation state is obtained using the evaluation suite described in Section 2 in [8]. The right plot in Figure 5.7 confirms that imposing AML relations trained with either transversality (AML_{trnsv}) or syzygies (AML_{syz}) yields an encoder with a lower distortion variability (i.e. lower variance of $\rho_{distort}$ coefficients defined by Equation 20).

Overall, results in Figure 5.7 show that imposing AML relations helps improve the latent space mapping of *PRED* when training on RGB frames that come from a non-stationary data stream of an RL learner.

Chapter 6

Conclusions and Future Directions

This thesis work presented several transfer-aware approaches. The proposed methods can incorporate large-scale data from imprecise simulations, while still retaining flexibility and agility for further active learning on hardware.

Methods from Chapter 3 demonstrated closing the sim-to-real gap using only 10 BO trials on a range of hardware platforms and tasks: bipedal locomotion with ATRIAS, hexapod locomotion with Daisy, task-oriented grasping and non-prehensile manipulation with ABB Yumi. BO-SVAE-DC method from Section 3.3 demonstrated applicability to different areas of robotics without requiring domain knowledge beyond setting up a general-purpose simulation of a target task. The code for this approach is available at `https://github.com/contactrika/bo-svae-dc`. It utilizes an open-source physics simulator PyBullet [73] and a recent open-source scalable BO library: BOTorch [95]. Hence, this code has the potential to be universally accessible and useful to the robotics community.

Chapter 4 provided an alternative way of utilizing simulation. The DET2STOC method interpreted simulators as regularizers for conditional VAEs. Sim-to-sim experiments in [7] indicated that DET2STOC could achieve better results than a CVAE that was given 10-100 times more data for training. This thesis focused on presenting the sim-to-real experiments (also described in [7]) to demonstrate the ability of DET2STOC to close the sim-to-real gap with hardware data. The use of simulation by DET2STOC can also be interpreted as placing an informed prior on the decoder. This 'simulation prior' is more flexible than parametric priors that are forced to encode information in a set of parameters of a chosen distribution. As a result, the decoder of DET2STOC yields a forward dynamics model $p_{\phi_{mix}}(\boldsymbol{s}_{t+1} \,|\, \boldsymbol{s}_t, \boldsymbol{a}_t)$ that utilizes simulation as a strong prior, but is also adapted to the real data. The adaptation to reality happens via finding simulation parameter posterior consistent with the data obtained from hardware. The future direction for this work would be to investigate the benefits this forward model can offer to methods like model predictive control (MPC).

Figure 6.1: A conceptual illustration that shows groceries from two different countries: objects that are similar in function but substantially different in appearance.

6.1 Lifelong Learning: Fast and Slow

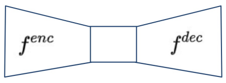
Chapter 5 presented a general approach to learn latent relations on source domains and transfer them to target domains. AML was applied to physics and robotics domains. However, in general AML does not assume that source or target domains are from any particular field. As as long some relations exist between the subsequences of latent states – AML would attempt to learn them, and would succeed if a chosen function approximator is capable of representing them. The modularity of the resulting latent relations is aimed to enable AML to tackle challenging future directions, such as lifelong learning.

Consider an autonomous agent that has acquired useful skills appropriate to a given environment e.g. a household robot doing a set of household chores. Suppose this agent is transported to an environment that has a different appearance, but similar latent rules/regularities/relations e.g a robot is moved to a different country to perform similar household tasks. In such cases we would like the agent to:

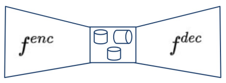
- quickly adapt to new visual appearances
- leverage experience embedded in the latent space structure, i.e. rules/regularities/relations inferred from previous experiences
- learn to infer new relations to better reshape the latent space, and retain ability to quickly re-adapt to the original environment

A sketch of applying AML to this situation could be as follows:

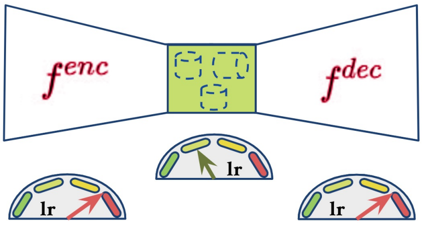
Step 0: *Learn a latent embedding on the source domain.* We assume that the source domain has plenty of data. This data can be from large datasets with real data or it could be synthetically generated data.



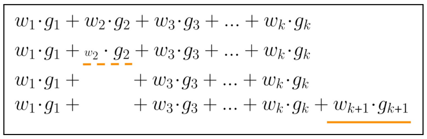
Step 1: *Learn relations to capture the latent data manifold.* We can learn a modular set of (non-linearly) independent relations using AML.



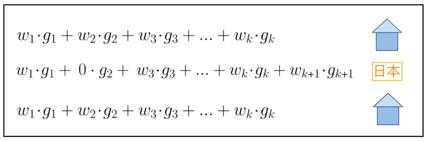
Step 2: *Impose the learned AML relations to help retain latent space structure when learning on the target domain.*



By imposing AML relations we can ensure that the latent space does not lose its previous structure rapidly. We can benefit from this by gaining an ability to increase encoder & decoder learning rates. This would help to adapt faster to changes in visual/high-dimensional aspects of the 'new world'. Hence, the agent would be able to quickly adjust to the new visual appearances.


$w_1 \cdot g_1 + w_2 \cdot g_2 + w_3 \cdot g_3 + ... + w_k \cdot g_k$
$w_1 \cdot g_1 + w_2 \cdot g_2 + w_3 \cdot g_3 + ... + w_k \cdot g_k$
$w_1 \cdot g_1 + \quad + w_3 \cdot g_3 + ... + w_k \cdot g_k$
$w_1 \cdot g_1 + \quad + w_3 \cdot g_3 + ... + w_k \cdot g_k + w_{k+1} \cdot g_{k+1}$

To let the latent space evolve/adapt as well: we could introduce weights for each imposed relation and slowly adapt them (e.g. by propagating gradients through the weights). We could suppress relations whose weights decay to zero and could also gradually expand the set by learning new relations.


$w_1 \cdot g_1 + w_2 \cdot g_2 + w_3 \cdot g_3 + ... + w_k \cdot g_k$
$w_1 \cdot g_1 + 0 \cdot g_2 + w_3 \cdot g_3 + ... + w_k \cdot g_k + w_{k+1} \cdot g_{k+1}$
日本
$w_1 \cdot g_1 + w_2 \cdot g_2 + w_3 \cdot g_3 + ... + w_k \cdot g_k$

We might anticipate that the agent would return to the 'old world' at some point. However, the agent might not want to store all f^{enc}, f^{dec} NNs from all the new environments it visits (these NNs would be large). Nonetheless, the agent could keep all the previously learned relations (small NNs), even those with weights ≈ 0. Hence, the old set of the latent relations and weights could be used upon return to the 'old world'.

Overall, this alternative way of adapting to new environments could be better than starting from scratch and better than fine-tuning. Starting from scratch is not data-efficient. Fine-tuning large NNs is prone to getting stuck in local optima, causing permanent degradation of performance, especially in case of a non-trivial gap between the visual appearances (or, in general, some high-dimensional aspects) of the source and target domains.

Another promising direction would be to learn policy representations (rather than state representations). If AML could be used to learn policies that are in some sense independent, then we could provide a way to learn a *portfolio* of policies that are complementary. Then, we could construct algorithms for learning diversified portfolios, such that a system capable of executing any policy in a portfolio could provide robustness to uncertainty and changes in the environment.

Bibliography

[1] An early “all models are wrong” perspective: Box, George EP. Robustness in the strategy of scientific model building. *Robustness in statistics*, pages 201–236, 1979.

[2] Rika Antonova, Akshara Rai, and Christopher G Atkeson. Deep kernels for optimizing locomotion controllers. In *Conference on Robot Learning (CoRL)*, 2017.

[3] Akshara Rai, Rika Antonova, Seungmoon Song, William Martin, Hartmut Geyer, and Christopher Atkeson. Bayesian optimization using domain knowledge on the atrias biped. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[4] Akshara Rai, Rika Antonova, Franziska Meier, and Christopher G Atkeson. Using simulation to improve sample-efficiency of bayesian optimization for bipedal robots. *Journal of Machine Learning Research (JMLR)*, 2019.

[5] Rika Antonova, Akshara Rai, Tianyu Li, and Danica Kragic. Bayesian optimization in variational latent spaces with dynamic compression. In *Conference on Robot Learning (CoRL)*, 2019.

[6] Rika Antonova, Mia Kokic, Johannes A Stork, and Danica Kragic. Global search with bernoulli alternation kernel for task-oriented grasping informed by simulation. In *Conference on Robot Learning (CoRL)*, 2018.

[7] Matin Hwasser, Danica Kragic, and Rika Antonova. Variational auto-regularized alignment for sim-to-real control. In *To appear in 2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[8] Rika Antonova, Maksim Maydanskiy, Danica Kragic, Sam Devlin, and Katja Hofmann. Analytic manifold learning: Unifying and evaluating representations for continuous control. *arXiv preprint arXiv:2006.08718*, 2020.

[9] Zirui Wang, Zihang Dai, Barnabás Póczos, and Jaime Carbonell. Characterizing and avoiding negative transfer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11293–11302, 2019.

[10] Yusen Zhan, Haitham Bou Ammar, et al. Theoretically-grounded policy advice from multiple teachers in reinforcement learning settings with applications to negative transfer. *arXiv preprint arXiv:1604.03986*, 2016.

[11] Igor Mordatch, Kendall Lowrey, and Emanuel Todorov. Ensemble-cio: Full-body dynamic motion planning that transfers to physical humanoids. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5307–5314. IEEE, 2015.

[12] Homanga Bharadhwaj, Zihan Wang, Yoshua Bengio, and Liam Paull. A data-efficient framework for training and sim-to-real transfer of navigation policies. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 782–788. IEEE, 2019.

[13] Antoine Cully, Jeff Clune, Danesh Tarapore, and Jean-Baptiste Mouret. Robots that can adapt like animals. *Nature*, 521(7553):503–507, 2015.

[14] Marvin Zhang, Xinyang Geng, Jonathan Bruce, Ken Caluwaerts, Massimo Vespignani, Vytas SunSpiral, Pieter Abbeel, and Sergey Levine. Deep reinforcement learning for tensegrity robot locomotion. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 634–641. IEEE, 2017.

[15] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. In *Robotics: Science and Systems (RSS)*, 2018.

[16] 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 *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017.

[17] Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In *International Conference on Robotics and Automation (ICRA)*, pages 8973–8979. IEEE, 2019.

[18] Rika Antonova, Silvia Cruciani, Christian Smith, and Danica Kragic. Reinforcement learning for pivoting task. *arXiv preprint arXiv:1703.00472*, 2017.

[19] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

[20] 2nd Workshop on Closing the Reality Gap in Sim2Real Transfer for Robotics. `https://sim2real.github.io`. *Robotics: Science and Systems*, 2020.

[21] Fabio Ramos, Rafael Possas, and Dieter Fox. BayesSim: Adaptive Domain Randomization Via Probabilistic Inference for robotics simulators. In *Robotics: Science and Systems*, 2019.

[22] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando De Freitas. Taking the Human Out of the Loop: A Review of Bayesian Optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

[23] Eric Brochu, Vlad M Cora, and Nando De Freitas. A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning. *arXiv preprint arXiv:1012.2599*, 2010.

[24] J Mockus, V Tiesis, and A Zilinskas. Chapter: Bayesian Methods for Seeking the Extremum. *Toward Global Optimization, Volume 2*, 1978.

[25] Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias Seeger. Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design. *Proceedings of the 27th International Conference on Machine Learning (ICML)*, 2010.

[26] Carl Edward Rasmussen and Hannes Nickisch. Gaussian processes for machine learning (gpml) toolbox. *J. Mach. Learn. Res.*, 11:3011–3015, December 2010.

[27] Jacob R Gardner, Geoff Pleiss, David Bindel, Kilian Q Weinberger, and Andrew Gordon Wilson. Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. In *Advances in Neural Information Processing Systems*, 2018.

[28] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[29] Edward Snelson and Zoubin Ghahramani. Sparse gaussian processes using pseudo-inputs. In *Advances in neural information processing systems*, pages 1257–1264, 2006.

[30] Joaquin Quiñonero-Candela and Carl Edward Rasmussen. A unifying view of sparse approximate gaussian process regression. *Journal of Machine Learning Research*, 6(Dec):1939–1959, 2005.

[31] Michalis Titsias. Variational learning of inducing variables in sparse gaussian processes. In *Artificial Intelligence and Statistics*, pages 567–574, 2009.

[32] Thang D Bui, Josiah Yan, and Richard E Turner. A unifying framework for gaussian process pseudo-point approximations using power expectation propagation. *The Journal of Machine Learning Research*, 18(1):3649–3720, 2017.

[33] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems (NIPS)*, pages 2951–2959, 2012.

[34] Marc Peter Deisenroth, Dieter Fox, and Carl Edward Rasmussen. Gaussian processes for data-efficient learning in robotics and control. *IEEE transactions on pattern analysis and machine intelligence*, 37(2):408–423, 2013.

[35] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[36] Athanasios S Polydoros and Lazaros Nalpantidis. Survey of model-based reinforcement learning: Applications on robotics. *Journal of Intelligent & Robotic Systems*, 86(2):153–173, 2017.

[37] Mustafa Mukadam, Xinyan Yan, and Byron Boots. Gaussian process motion planning. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 9–15. IEEE, 2016.

[38] Stanimir Dragiev, Marc Toussaint, and Michael Gienger. Gaussian process implicit surfaces for shape estimation and grasping. In *2011 IEEE International Conference on Robotics and Automation*, pages 2845–2850. IEEE, 2011.

[39] Zhe Hu, Peigen Sun, and Jia Pan. Three-dimensional deformable object manipulation using fast online gaussian process regression. *IEEE Robotics and Automation Letters*, 3(2):979–986, 2018.

[40] Nawid Jamali, Carlo Ciliberto, Lorenzo Rosasco, and Lorenzo Natale. Active perception: Building objects' models using tactile exploration. In *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, pages 179–185. IEEE, 2016.

[41] Sergio Caccamo, Yasemin Bekiroglu, Carl Henrik Ek, and Danica Kragic. Active exploration using gaussian random fields and gaussian process implicit surfaces. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 582–589. IEEE, 2016.

[42] Matthew Tesch, Jeff Schneider, and Howie Choset. Using response surfaces and expected improvement to optimize snake robot gait parameters. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1069–1074. IEEE, 2011.

[43] Daniel J Lizotte, Tao Wang, Michael H Bowling, and Dale Schuurmans. Automatic Gait Optimization with Gaussian Process Regression. In *International Joint Conference on Artificial Intelligence (IJCAI)*, volume 7, pages 944–949, 2007.

[44] OB Kroemer, Renaud Detry, Justus Piater, and Jan Peters. Combining active learning and reactive control for robot grasping. *Robotics and Autonomous systems*, 58(9):1105–1116, 2010.

[45] Luis Montesano and Manuel Lopes. Active learning of visual descriptors for grasping using non-parametric smoothed beta distributions. *Robotics and Autonomous Systems*, 60(3):452–462, 2012.

[46] John Oberlin and Stefanie Tellex. Autonomously acquiring instance-based object models from experience. In *Robotics Research*, pages 73–90. Springer, 2018.

[47] Kirthevasan Kandasamy, Gautam Dasarathy, Jeff Schneider, and Barnabás Póczos. Multi-fidelity Bayesian Optimisation with Continuous Approximations. In *International Conference on Machine Learning (ICML)*, pages 1799–1808, 2017.

[48] Alonso Marco, Felix Berkenkamp, Philipp Hennig, Angela P Schoellig, Andreas Krause, Stefan Schaal, and Sebastian Trimpe. Virtual vs. real: Trading off simulations and physical experiments in reinforcement learning with bayesian optimization. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1557–1563. IEEE, 2017.

[49] Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P Xing. Deep kernel learning. In *Artificial Intelligence and Statistics*, pages 370–378, 2016.

[50] Roberto Calandra, Jan Peters, Carl Edward Rasmussen, and Marc Peter Deisenroth. Manifold gaussian processes for regression. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 3338–3345. IEEE, 2016.

[51] Aaron Wilson, Alan Fern, and Prasad Tadepalli. Using Trajectory Data to Improve Bayesian Optimization for Reinforcement Learning. *The Journal of Machine Learning Research (JMLR)*, 15(1):253–282, 2014.

[52] Jasper Snoek, Kevin Swersky, Rich Zemel, and Ryan Adams. Input warping for Bayesian optimization of non-stationary functions. In *International Conference on Machine Learning (ICML)*, pages 1674–1682, 2014.

[53] Michael I Jordan, Zoubin Ghahramani, Tommi S Jaakkola, and Lawrence K Saul. An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233, 1999.

[54] Cheng Zhang, Judith Bütepage, Hedvig Kjellström, and Stephan Mandt. Advances in variational inference. *IEEE transactions on pattern analysis and machine intelligence*, 41(8):2008–2026, 2018.

[55] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[56] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and variational inference in deep latent gaussian models. In *International Conference on Machine Learning*, volume 2, 2014.

[57] Jakub Tomczak and Max Welling. Vae with a vampprior. In *International Conference on Artificial Intelligence and Statistics*, pages 1214–1223, 2018.

[58] Jan Stuehmer, Richard Turner, and Sebastian Nowozin. Independent subspace analysis for unsupervised learning of disentangled representations. In *Conference on Artificial Intelligence and Statistics (AISTATS)*, 2020.

[59] Li Yingzhen and Stephan Mandt. Disentangled sequential autoencoder. In *International Conference on Machine Learning*, pages 5670–5679, 2018.

[60] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning Structured Output Representation using Deep Conditional Generative Models. In *Advances in Neural Information Processing Systems*, 2015.

[61] Timothée Lesort, Natalia Díaz-Rodríguez, Jean-Franois Goudou, and David Filliat. State representation learning for control: An overview. *Neural Networks*, 108:379–392, 2018.

[62] Rika Antonova, Akshara Rai, and Christopher G Atkeson. Sample efficient optimization for learning controllers for bipedal locomotion. In *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, pages 22–28. IEEE, 2016.

[63] Christian Hubicki, Jesse Grimes, Mikhail Jones, Daniel Renjewski, Alexander Spröwitz, Andy Abate, and Jonathan Hurst. ATRIAS: Design and validation of a tether-free 3D-capable spring-mass bipedal robot. *The International Journal of Robotics Research (IJRR)*, 35(12):1497–1521, 2016.

[64] BJN Blight and L Ott. A bayesian approach to model inadequacy for polynomial regression. *Biometrika*, 62(1):79–88, 1975.

[65] Zachary Batts, Seungmoon Song, and Hartmut Geyer. Toward a virtual neuromuscular control for robust walking in bipedal robots. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6318–6323. IEEE, 2015.

[66] Christos Louizos, Kevin Swersky, Yujia Li, Max Welling, and Richard Zemel. The variational fair autoencoder. *International Conference on Learning Representations*, 2016.

[67] Durk P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. In *Advances in neural information processing systems*, pages 3581–3589, 2014.

[68] Pedro J Moreno, Purdy P Ho, and Nuno Vasconcelos. A kullback-leibler divergence based kernel for svm classification in multimedia applications. In *Advances in neural information processing systems*, pages 1385–1392, 2004.

[69] Tom Minka. Divergence measures and message passing. Technical report, Microsoft Research, 2005.

[70] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[71] Carlos Riquelme, Matthew Johnson, and Matt Hoffman. Failure modes of variational inference for decision making. *Prediction and Generative Modeling in RL Workshop (AAMAS, ICML, IJCAI)*, 2018.

[72] Sebastian Tschiatschek, Kai Arulkumaran, Jan Stühmer, and Katja Hofmann. Variational inference for data-efficient model learning in pomdps. *arXiv:1805.09281*, 2018.

[73] Pybullet simulator. `https://github.com/bulletphysics/bullet3`. Accessed: 2019-06.

[74] Hebi Robotics. `http://docs.hebi.us`. Accessed: 2019-06.

[75] Alessandro Crespi and Auke Jan Ijspeert. Online optimization of swimming and crawling in an amphibious snake robot. *IEEE Transactions on Robotics*, 24(1):75–87, 2008.

[76] S Surjanovic and D Bingham. Virtual library of simulation experiments: test functions and datasets. *Simon Fraser University, Burnaby, BC, Canada*, 13:2015, 2013.

[77] Rosen Diankov and James Kuffner. Openrave: A planning architecture for autonomous robotics. *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-08-34*, 79, 2008.

[78] Carlos Rubert, Daniel Kappler, Antonio Morales, Stefan Schaal, and Jeannette Bohg. On the relevance of grasp metrics for predicting grasp success. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 265–272. IEEE, 2017.

[79] Nikolaus Correll, Kostas E Bekris, Dmitry Berenson, Oliver Brock, Albert Causo, Kris Hauser, Kei Okada, Alberto Rodriguez, Joseph M Romano, and Peter R Wurman. Analysis and observations from the first amazon picking challenge. *IEEE Transactions on Automation Science and Engineering*, 15(1):172–188, 2016.

[80] Berk Calli, Arjun Singh, Aaron Walsman, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M Dollar. The ycb object and model set: Towards common benchmarks for manipulation research. In *International Conference on Advanced Robotics (ICAR)*, pages 510–517. IEEE, 2015.

[81] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[82] 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*, pages 5048–5058, 2017.

[83] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033. IEEE, 2012.

[84] Niko Sünderhauf, Oliver Brock, Walter Scheirer, Raia Hadsell, Dieter Fox, Jürgen Leitner, Ben Upcroft, Pieter Abbeel, Wolfram Burgard, Michael Milford, and Peter Corke. The limits and potentials of deep learning for robotics. *The International Journal of Robotics Research*, 37(4-5):405–420, 2018.

[85] Laurenz Wiskott and Terrence J Sejnowski. Slow feature analysis: Unsupervised learning of invariances. *Neural computation*, 14(4):715–770, 2002.

[86] Varun Raj Kompella, Matthew Luciw, and Juergen Schmidhuber. Incremental slow feature analysis: Adaptive and episodic learning from high-dimensional input streams. *arXiv preprint arXiv:1112.2113*, 2011.

[87] Ankesh Anand, Evan Racah, Sherjil Ozair, Yoshua Bengio, Marc-Alexandre Côté, and R Devon Hjelm. Unsupervised state representation learning in ATARI. In *Advances in Neural Information Processing Systems 32*, pages 8766–8779, 2019.

[88] Selman Akbulut and Henry King. On approximating submanifolds by algebraic sets and a solution to the nash conjecture. *Inventiones mathematicae*, 107(1):87–98, 1992.

[89] Martin Jaggi. Revisiting frank-wolfe: Projection-free sparse convex optimization. In *Proceedings of the 30th international conference on machine learning*, number CONF, pages 427–435, 2013.

[90] Klaus Greff, Raphaël Lopez Kaufman, Rishabh Kabra, Nick Watters, Christopher Burgess, Daniel Zoran, Loic Matthey, Matthew Botvinick, and Alexander Lerchner. Multi-object representation learning with iterative variational inference. In *International Conference on Machine Learning*, 2019.

[91] Haiyan Yin, Jianda Chen, and Sinno Jialin Pan. Hashing over predicted future frames for informed exploration of deep reinforcement learning. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI)*, 2018.

[92] Luisa Zintgraf, Kyriacos Shiarlis, Maximilian Igl, Sebastian Schulze, Yarin Gal, Katja Hofmann, and Shimon Whiteson. Varibad: A very good method for bayes-adaptive deep rl via meta-learning. 2020.

[93] Leena Chennuru Vankadara and Ulrike von Luxburg. Measures of distortion for machine learning. In *Advances in Neural Information Processing Systems 31*, pages 4886–4895. 2018.

[94] Yair Bartal, Nova Fandina, and Ofer Neiman. Dimensionality reduction: theoretical perspective on practical measures. In *Advances in Neural Information Processing Systems*, pages 10576–10588, 2019.

[95] M. Balandat, B. Karrer, D. Jiang, B. Letham, S. Daulton, A. Wilson, E. Bakshy. BoTorch. `https://botorch.org/`. Accessed: 2019-05.

Part II

Included Publications

Bayesian Optimization in Variational Latent Spaces with Dynamic Compression

Rika Antonova[1]
EECS, KTH, Stockholm, Sweden

Akshara Rai[1]
Facebook AI Research

Tianyu Li
Facebook AI Research

Danica Kragic
EECS, KTH, Stockholm, Sweden

Abstract

Data-efficiency is crucial for autonomous robots to adapt to new tasks and environments. In this work, we focus on robotics problems with a budget of only 10-20 trials. This is a very challenging setting even for data-efficient approaches like Bayesian optimization (BO), especially when optimizing higher-dimensional controllers. Previous work extracted expert-designed low-dimensional features from simulation trajectories to construct informed kernels and run ultra sample-efficient BO on hardware. We remove the need for expert-designed features by proposing a model and architecture for a sequential variational autoencoder that embeds the space of simulated trajectories into a lower-dimensional space of latent paths in an unsupervised way. We further compress the search space for BO by reducing exploration in parts of the state space that are undesirable, without requiring explicit constraints on controller parameters. We validate our approach with hardware experiments on a Daisy hexapod robot and an ABB Yumi manipulator. We also present simulation experiments with further comparisons to several baselines on Daisy and two manipulators. Our experiments indicate the proposed trajectory-based kernel with dynamic compression can offer ultra data-efficient optimization.

Keywords: Bayesian optimization, Variational inference, Data-efficient RL

1 Introduction

Reinforcement learning (RL) is becoming popular in robotics, since in some cases it can deal with real-world challenges, such as noise in control and measurements, non-convexity and discontinuities in objectives. However, most flexible RL methods

[1]Both of these authors contributed equally.

In proceedings of the 3rd Conference on Robot Learning (CoRL 2019), Osaka, Japan.
Published by PMLR http://proceedings.mlr.press/v100/antonova20a License: CC-BY 4.0

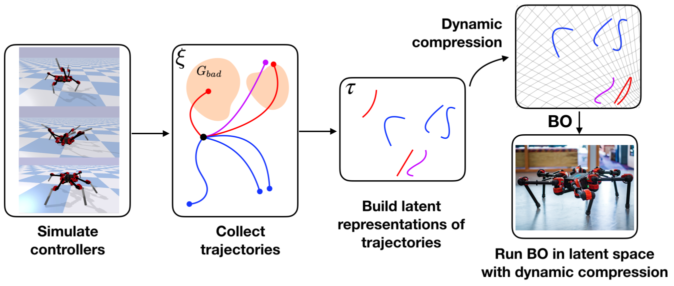

Simulate controllers
Collect trajectories
ξ
G_{bad}
τ
Build latent representations of trajectories
Dynamic compression
BO
Run BO in latent space with dynamic compression

Figure 1: An overview of our approach: We start by simulating controllers and collecting their trajectories $\boldsymbol{\xi}$, along with the fraction of time spent in undesirable regions given by G_{bad}. Next, we learn to embed trajectories into a lower-dimensional a space of latent paths $\boldsymbol{\tau}$. We use dynamic compression to scale distances between latent paths based on their desirability. This dynamically compressed latent space is used for BO on hardware. Trajectory data $\boldsymbol{\xi}$ consists of high-frequency readings of robot joint angles and object position/velocity estimates (the framework can accommodate vision-based data in the future, but we do not experiment with it in this work).

require thousands to millions of data samples, which can make direct application to real-world robotics infeasible. For example, 10,000 30s trials/episodes on a real robot would require ≈100 hours of operation. Most full-scale platforms, especially in locomotion, cannot operate this long without maintenance. Nowadays, commercially available arms can operate for longer, however sophisticated anthropomorphic hands and advanced grippers are still highly prone to breakage after even a handful of trials [1]. Hence the need for algorithms that can learn in very few trials, without causing significant wear-and tear to the hardware.

In this work we focus on cases with a budget of only 10-20 trials. In such settings, using approaches like Bayesian optimization (BO) to adjust parameters of structured controllers can help improve data efficiency. However, success of BO on hardware has been demonstrated either with low-dimensional controllers or with simulation-based kernels that required hand-designed features. We propose learning simulation-based kernels in an unsupervised way with a sequential variational autoencoder (SVAE). Our approach embeds simulated trajectories $\boldsymbol{\xi}$ to a space of latent paths $\boldsymbol{\tau}$, and jointly learns a probability distribution $p(\boldsymbol{\tau}|\boldsymbol{x})$ that controllers with parameters $\boldsymbol{x}$ induce over the space of latent paths. We were inspired by initial success of trajectory-based BO kernels [2], however that was demonstrated for BO in low dimensions (2-4D). Our results show that performance of a kernel based on raw trajectories deteriorates quickly for higher-dimensional problems. In contrast, our kernel based on latent paths can still offer gains even for 48-dimensional controllers.

Global optimization in latent space can still suffer from sampling unsuccessful controllers, especially in the absence of dense rewards. One solution can be adding domain-specific constraints to point optimization in the right direction. While these can be hard to define in controller parameter space, frequently they can be easily expressed in observation/state space. For example, high velocities might be undesirable if they result in hard impacts. However, formulating this as constrained optimization could result in overly conservative controllers. Instead, we incorporate controller desirability into BO by reducing exploration in the part of the trajectory space that leads to undesirable behavior. We compress the search space during BO dynamically by scaling the distance between controllers based on their desirability, initially inferred from simulation. BO can then quickly reject the undesirable parts of the search space, allowing for more exploration in the desirable parts. Figure 1 gives an overview of the proposed approach.

We test our approach (SVAE-DC: informed SVAE kernel with Dynamic Compression) on a Daisy hexapod and an ABB Yumi manipulator on hardware. We also conduct further simulation-based analysis on Daisy and two manipulators. On Daisy, our method consistently learns to walk in less than 10 hardware trials, outperforming uninformed BO. We also demonstrate significant gains on a nonprehensile manipulation task on Yumi. All latent components of our kernel can be adjusted online (by optimizing marginal likelihood as is done for BO hyperparameters). We anticipate that such adjustment could be useful for future works for settings with a medium budget of trials ($\approx$100+). Our code builds on the recently released BoTorch library [3] that supports highly scalable BO on GPUs. We open source our code for simulation environments, training and BO[2].

2 Background and Related Work

For learning with a small number of trials we turn to Bayesian Optimization (BO). It can be thought of as a data-efficient RL method that obtains a reward only at the end of each trial/episode. BO offers a principled way to trade-off exploration vs exploitation (see BO introduction and overview in [4]). For higher-dimensional robotics problems BO can benefit significantly from using simulation-based kernels. However, previous work required defining domain-specific features to be extracted from large-scale simulation data (see Section 2.1). Variational Autoencoders (VAEs) [5] provide an unsupervised alternative for embedding high-dimensional observations into a lower-dimensional space. For example, [6] recently used VAE in a Gaussian Process (GP) kernel to optimize chemical molecules. In robotics, VAEs have been used to process visual and tactile data (see [7] for a survey). We are interested in encoding trajectory data, so a sequential VAE (SVAE) could be applicable. [8, 9] show SVAEs learning latent dynamics. However, their physics simulations are low-dimensional (e.g. position of a 2D ball), sequences have length 20-30 steps, and the focus is on visual reconstruction. We aim to develop SVAE architecture that can easily handle simulations from full-scale robotics systems (state spaces 27D+) and much longer sequences (lengths 500-1000).

Our original motivation for embedding trajectory data into the kernel was Behavior Based Kernel (BBK) [2]. On low-dimensional problems it outperformed PILCO [10], which is one of the most popular model-based RL algorithms and has been widely used for small domains. For larger domains, such as those in our

[2]SVAE-DC and BO code: `https://github.com/contactrika/bo-svae-dc`

experiments, scaling PILCO can be difficult or intractable (see Section 5 in [2]). Instead of a direct comparison to PILCO, we compare our approach to a scalable version of BBK. BBK is directly applicable only to stochastic policies, but we adapted it to our setting as BBK-KL baseline. We randomize simulator parameters when collecting trajectories. Hence, even if the simulator and controllers are deterministic, each controller still induces a probability distribution over the trajectories. As proposed for BBK, for kernel distances we used symmetrized KL between trajectory distributions induced by the controllers. The generation and reconstruction parts of SVAE were used to estimate this KL. Since this baseline uses a neural network in the kernel, there is some relation to methods in [11, 12] (though these focused on GP regression, and did not use trajectories).

A part of our work can be viewed as learning a low-dimensional representation of trajectories, which is widely studied in robotics. For example, [13] use dynamic movement primitives (DMPs) to encode human demonstrations. Our locomotion controller is a variant of a cyclic DMP, which assumes synchronization between the different joints of the robot. For locomotion, we provide comparisons to BO with a standard kernel, which gives a sense of the performance of optimizing DMP parameters with standard BO. However, for manipulation DMPs require demonstrations for data-efficiency. Since we do not assume access to those, such approaches cannot be directly compared to our setup.

2.1 BO for Locomotion and Manipulation

Locomotion controllers most commonly used for real systems are structured and parametric [14, 15, 16]. BO has been used to optimize their parameters, e.g. [17, 18, 19]. Typically, these methods take ≈40 trials for low-dimensional controllers (3-5D). For high-dimensional controllers further domain information is needed. For example [20] use simulation and user-defined features to transform the space of a 36-dimensional controller into 6D, making the search for walking controllers of a hexapod much more data-efficient. [21] employ bipedal locomotion features to build informed kernels. While a number of other RL methods can succeed in simulation, obtaining results applicable for locomotion on hardware is challenging. Recently, [22, 23] showed that a deep RL method (PPO [24]) can be used for locomotion on hardware. However, they learn conservative controllers in simulation and help transfer via system identification of actuator dynamics [22] and a user-designed structured controller [23]. While these methods can help, they do not guarantee that a controller learned in simulation will perform well on hardware. [25] showed learning to walk on a Minitaur quadruped in only two hours. Minitaur has 8 motors that control its longitudinal motion, and no actuation for lateral movements. In comparison, our hexapod (Daisy) has 18 motors and omni-directional movements. Hence, learning control for Daisy would require significantly longer training. Since most present day locomotion robots (including Daisy) get damaged from wear and tear when operated for long, approaches that succeed for simpler quadruped controllers could be intractable in this setting.

In manipulation, active learning and BO have been used, for example, for grasping [26, 27]. These works did not incorporate simulation into the kernel, so their performance would be similar to BO with uninformed/standard kernel. [28] showed advantages of a simulation-based kernel, but needed grasping-specific features. Somewhat related are works in sim-to-real transfer, like [1], though many have visuomotor control as the focus (not considered here) and usually do not adapt online. [29] do adjust simulation parameters to match reality, so it would be interesting to combine this with BO in the future for global optimality (their work employs PPO, which is locally optimal). Due to uncertainty over friction and contact forces, sim-to-real is challenging for non-prehensile problems. However, such motions can be useful to make solutions feasible (e.g pushing when the object is too large/heavy to lift or the goal is out of reach). [30, 31] report success in transfer/adaptation on a push-to-goal task, showing the task is challenging but feasible. In our experiments we consider a 'stable push' task: push two tall objects across a table without tipping them over. The further challenges come from interaction between objects and inability to recover from them tipping over.

3 SVAE-DC: Learning Informed Trajectory Embeddings

We model our setting as a joint Variational Inference problem: learning to compress/reconstruct trajectories while at the same time learning to associate controllers with their corresponding probability distributions over the latent paths. For this we develop a version of sequential VAE (SVAE). The training is guided by ELBO (Evidence Lower Bound) derived for our setting directly from the modeling assumptions and doesn't require any auxiliary objectives.

First, we define notation:

$\pi_{\boldsymbol{x}}$: policy/controller with parameters $\boldsymbol{x}, \boldsymbol{x} \in \mathbb{R}^D$; policies can be either deterministic or stochastic; for brevity we will refer to $\pi_{\boldsymbol{x}}$ simply as 'controller $\boldsymbol{x}$'

$\boldsymbol{\xi} \equiv \boldsymbol{\xi}_{1:T}$: original trajectory for T time steps containing high-frequency sensor readings

$\boldsymbol{\tau} \equiv \boldsymbol{\tau}_{1:K}$: latent space 'path' (embedding of a trajectory)

$p(\boldsymbol{\xi}_{1:T}|\boldsymbol{x})$: a conditional probability distribution over the trajectories induced by controller $\boldsymbol{x}$; the relationship between the controller and trajectories could be probabilistic either because the controller is stochastic, or because the simulator environment is stochastic, or both

$p(\boldsymbol{\tau}_{1:K}|\boldsymbol{x})$: a conditional probability distribution over latent space paths induced controller by $\boldsymbol{x}$

$G_{bad} : S \rightarrow \{0, 1\}$ a map denoting whether an observation $\boldsymbol{\xi}_t \in S$ is within an undesirable region

y : fraction of time $\boldsymbol{\xi}$ spends in undesirable regions; $\boldsymbol{\psi}$ learns analogous notion for a latent path $\boldsymbol{\tau}$

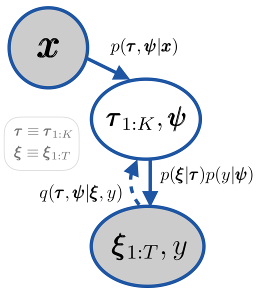

$\boldsymbol{x}$
$p(\boldsymbol{\tau},\boldsymbol{\psi}|\boldsymbol{x})$
$\boldsymbol{\tau}_{1:K},\boldsymbol{\psi}$
$\tau \equiv \tau_{1:K}$
$\xi \equiv \xi_{1:T}$
$q(\boldsymbol{\tau},\boldsymbol{\psi}|\boldsymbol{\xi},y)$
$p(\boldsymbol{\xi}|\boldsymbol{\tau})p(y|\boldsymbol{\psi})$
$\boldsymbol{\xi}_{1:T},y$

Figure 2: A sketch of generative and inference model.

Our goal is to learn $p(\boldsymbol{\tau},\boldsymbol{\psi}|\boldsymbol{x})$. $p(\boldsymbol{\tau}|\boldsymbol{x})$ is analogous to $p(\boldsymbol{\xi}|\boldsymbol{x})$, only the paths are encoded in a lower-dimensional latent space. This is useful for constructing kernels for efficient BO on hardware. As a measure of trajectory 'quality' we can keep track of how long each trajectory spends in undesirable regions (y). For the latent paths we learn the analogous notion ($\boldsymbol{\psi} = \psi_{1:K}$), which enables modularity and fast online updates (discussed in Section 4). We do not impose hard constraints during optimization, so G_{bad} used to compute y can be specified roughly, with approximate guesses. Our framework also supports $G_{bad} : S \to [0,1]$, but for users it is frequently easier to make a rough thresholded estimate rather than providing smooth estimates or probabilities. The graphical model we construct for this setting is shown in Figure 2. Not all independencies are captured by the illustration. So, explicitly, the generative model is: $p_{\boldsymbol{w}}(\boldsymbol{\tau},\boldsymbol{\psi},\boldsymbol{\xi},y \mid \boldsymbol{x}) = p(\boldsymbol{\tau}_{1:K},\boldsymbol{\psi}|\boldsymbol{x})p(y|\boldsymbol{\psi})\prod_{t=1}^{T} p(\boldsymbol{\xi}_t|\boldsymbol{\xi}_{t-1},\boldsymbol{\tau}_{1:K})$.
Approximate posterior is modeled by: $q_{\boldsymbol{\phi}}(\boldsymbol{\tau},\boldsymbol{\psi}|\boldsymbol{\xi},y) = q(\boldsymbol{\tau}_{1:K},\boldsymbol{\psi}|\boldsymbol{\xi}_{1:T},y)$.
We collect trajectories $\boldsymbol{\xi}_{1:T}^{(i)}$ by simulating N controllers with parameters $\boldsymbol{x}^{(i)}$ for T time steps. We derive ELBO for this setting to maximize $\log p(Data) = \log p(\{\boldsymbol{x}^{(i)},\boldsymbol{\xi}_{1:T}^{(i)}\}_{i=1...N})$. Using '$\tilde{\ }$' over the variables to indicate samples from the current variational approximation, we get:

$$\mathcal{L}^{DC}(\boldsymbol{w},\boldsymbol{\phi}|\boldsymbol{x},\boldsymbol{\xi},y) = \mathbb{E}_{\tilde{\boldsymbol{\tau}},\tilde{\boldsymbol{\psi}}\sim q(\boldsymbol{\tau},\boldsymbol{\psi}|\boldsymbol{\xi},y)}\big[\log p(\boldsymbol{\xi}|\tilde{\boldsymbol{\tau}}) + \log p(y|\tilde{\boldsymbol{\psi}}) + \\ \log p(\tilde{\boldsymbol{\tau}},\tilde{\boldsymbol{\psi}}|\boldsymbol{x}) - \log q(\tilde{\boldsymbol{\tau}},\tilde{\boldsymbol{\psi}}|\boldsymbol{\xi},y)\big] \tag{1}$$

$\boldsymbol{w}$, $\boldsymbol{\phi}$ are weights of deep neural networks optimized by gradient ascent on the ELBO.

4 Bayesian Optimization with Dynamic Compression

In Bayesian Optimization (BO), the problem of optimizing controllers is viewed as finding controller parameters $\boldsymbol{x}^*$ that optimize some objective function $f(\boldsymbol{x})$: $f(\boldsymbol{x}^*) = \max_{\boldsymbol{x}} f(\boldsymbol{x})$. At each optimization trial BO optimizes an auxiliary function to select the next promising $\boldsymbol{x}$ to evaluate. f is commonly modeled with a Gaussian process (GP): $f(\boldsymbol{x}) \sim \mathcal{GP}(m(\boldsymbol{x}), k(\boldsymbol{x}_i,\boldsymbol{x}_j))$.

The key object is the kernel function $k(\cdot,\cdot)$, which encodes similarity between inputs. If $k(\boldsymbol{x}_i,\boldsymbol{x}_j)$ is large for inputs $\boldsymbol{x}_i,\boldsymbol{x}_j$, then $f(\boldsymbol{x}_i)$ strongly influences $f(\boldsymbol{x}_j)$. One of the most widely used kernel functions is the Squared Exponential (SE) kernel: $k_{SE}(\boldsymbol{r} \equiv |\boldsymbol{x}_i - \boldsymbol{x}_j|) = \sigma_k^2 \exp\big(-\frac{1}{2}\boldsymbol{r}^T \operatorname{diag}(\boldsymbol{\ell})^{-2}\boldsymbol{r}\big)$, where σ_k^2, $\boldsymbol{\ell}$ are signal variance and a vector of length scales respectively. σ_k^2, $\boldsymbol{\ell}$ are called 'hyperparameters' and are optimized automatically by maximizing marginal likelihood ([4], Section V-A). SE belongs to a broader class of Matérn kernels. One common parameter choice yields Matérn$_{5/2}$: $k_{\text{Matérn}_{5/2}}(\boldsymbol{r}) = \big(1 + \frac{\sqrt{5}\boldsymbol{r}}{\boldsymbol{\ell}} + \frac{5\boldsymbol{r}^2}{3\boldsymbol{\ell}^2}\big)\exp\big(-\frac{\sqrt{5}\boldsymbol{r}}{\boldsymbol{\ell}}\big)$. SE and Matérn kernels are stationary, since they depend on $\boldsymbol{r} \equiv \boldsymbol{x}_i - \boldsymbol{x}_j \ \forall \boldsymbol{x}_{i,j}$, and not on individual $\boldsymbol{x}_i,\boldsymbol{x}_j$.

Section 2.1 discussed recent work that showed how to effectively remove stationarity by using informed feature transforms for kernel computations. But these required extracting domain-specific features manually, or learning to fit a pre-defined set of features using a deterministic NN in a supervised way.

We propose to use $p(\boldsymbol{\tau}, \boldsymbol{\psi}|\boldsymbol{x})$ learned by SVAE-DC. [2] showed that a 'symmetrization' of KL divergence can be used to define a KL-based kernel for trajectories in the original space:

$$\begin{aligned} k_{KL} &= \exp(\text{-}\alpha D(\boldsymbol{x}_i, \boldsymbol{x}_j)) \\ D(\boldsymbol{x}_i, \boldsymbol{x}_j) &= \sqrt{KL(p(\boldsymbol{\xi}|\boldsymbol{x}_i)||p(\boldsymbol{\xi}|\boldsymbol{x}_j))} + \sqrt{KL(p(\boldsymbol{\xi}|\boldsymbol{x}_j)||p(\boldsymbol{\xi}|\boldsymbol{x}_i))} \end{aligned} \tag{2}$$

In theory, we could use this to define an analogous kernel in the latent space:

$$\begin{aligned} k_{LKL} &= \exp(\text{-}\alpha D_\tau(\boldsymbol{x}_i, \boldsymbol{x}_j)) \\ D_\tau(\boldsymbol{x}_i, \boldsymbol{x}_j) &= \sqrt{KL(p(\boldsymbol{\tau}|\boldsymbol{x}_i)||p(\boldsymbol{\tau}|\boldsymbol{x}_j))} + \sqrt{KL(p(\boldsymbol{\tau}|\boldsymbol{x}_j)||p(\boldsymbol{\tau}|\boldsymbol{x}_i))} \end{aligned}$$

However, variational inference (VI) tends to under-estimate variances [32, 33, 34, 35]. Hence, our kernel works with latent means $\bar{\boldsymbol{\tau}}_{\boldsymbol{x}}, \bar{\boldsymbol{\psi}}_{\boldsymbol{x}} = E\big[p(\boldsymbol{\tau}, \boldsymbol{\psi}|\boldsymbol{x})\big]$ directly. We define our kernel function with:

$$\boldsymbol{r}_\tau = D_\tau(\boldsymbol{x}_i, \boldsymbol{x}_j) = (1-\bar{y}_{\boldsymbol{x}_i})\bar{\boldsymbol{\tau}}_{\boldsymbol{x}_i} - (1-\bar{y}_{\boldsymbol{x}_j})\bar{\boldsymbol{\tau}}_{\boldsymbol{x}_j} \quad ; \quad y_{\boldsymbol{x}} \sim p(y|\bar{\boldsymbol{\psi}}_{\boldsymbol{x}}) \tag{3}$$

$$k_{SVAE\text{-}DC}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sigma_k^2 \exp\left(-\tfrac{1}{2}\boldsymbol{r}_\tau^T \operatorname{diag}(\boldsymbol{\ell})^{-2}\boldsymbol{r}_\tau\right) \tag{4}$$

The form of Equation 4 allows us to apply existing machinery for optimizing kernel hyperparameters $\sigma_k^2, \boldsymbol{\ell}$ directly to the SVAE-DC kernel. Note that $\operatorname{diag}(\boldsymbol{\ell})^{-2}$ is related to covariance in the case diagonal Gaussians. So BO with $\bar{\boldsymbol{\tau}}_{\boldsymbol{x}_i} - \bar{\boldsymbol{\tau}}_{\boldsymbol{x}_j}$ in the kernel is related to using KL in the case of diagonal Gaussians (with a simplification to capture variance-only terms by learning σ_k^2). We can also conveniently obtain SVAE-DC-Matérn version of the kernel by simply changing the form of Equation 4 to the Matérn function.

Scaling latent representations by $1-\bar{y}_{\boldsymbol{x}}$ yields dynamic compression: latent representations that correspond to controllers frequently visiting undesirable parts of the space are scaled down. With this, we retain trajectory-based distance in the desirable parts of the space, but compress it in undesirable parts to reduce unwanted exploration. The 'dynamic compression' transformation is applied after SVAE training, in addition to the compression obtained by SVAE. The scaling can be made non-linear with $sigmoid(\alpha(\bar{y}_{\boldsymbol{x}} - c))$. This achieves aggressive compression in settings with an extremely small budget of trials. The additional parameters α, c, as well as $p(\boldsymbol{\tau}, \boldsymbol{\psi}|\boldsymbol{x}), p(y|\boldsymbol{\psi})$ can be optimized online (as BO hyperparameters). Note that because of the multiplicative formulation, two controllers with different $\boldsymbol{\tau}$ and y can appear similar during optimization. Theoretically, this can also bring the undesirable space close to a part of desirable space. We address this by updating the learned components online via GP's marginal likelihood, as in Deep kernel learning

[11, 12]. However, for large NNs such online updates would only be useful after a large number of hardware trials. Hence, we provide a modular architecture to ensure that the multiplicative factors can be updated faster. We structure SVAE to learn $p(y|\boldsymbol{\psi})$ and $p(\boldsymbol{\tau}, \boldsymbol{\psi}|\boldsymbol{x})$, instead of a joint $p(y, \boldsymbol{\tau}|\boldsymbol{x})$. This makes the NN for $p(y|\boldsymbol{\psi})$ small, facilitating more data-efficient NN updates during BO. Now, during hardware trials, shifts in $\bar{y}$ will be more pronounced, compared to updates in the full latent path representation.

In summary, SVAE-DC and the resulting kernel result in a fully automatic way of learning latent trajectory embeddings in unsupervised way. For domains where G_{bad} is given, we can also achieve dynamic compression of the latent space, making BO ultra data-efficient. All the components used during BO can be optimized online via the same methods as those for adjusting BO hyperparameters.

5 SVAE-DC: NN Architecture and Training

We propose to use time convolution architecture for $q(\boldsymbol{\tau}|\boldsymbol{\xi})$, de-convolutions for $p(\boldsymbol{\xi}|\boldsymbol{\tau})$. For this we use 1D convolutions for the sequential dimensions t, k and treat the dimensions of $\boldsymbol{\xi}_t, \boldsymbol{\tau}_k$ as different channels. With that, for all our experiments (all different robot and controller architectures) we were able to use the same network parameters: 3-layer 1D convolutions with [32, 64, 128] channels (reverse order for de-convolutions; kernel size 4, stride 2) followed by MLP layer for μ, σ outputs. We were also able to use same latent space sizes: 3-dimensional $\boldsymbol{\tau}$, latent sequence length K=3 for all our experiments. This yielded a small 9D optimization space for BO, which is highly desirable for optimization with few trials. Notably, this NN architecture also retained good reconstruction accuracy, not far from results with larger latent spaces ($\boldsymbol{\tau}$=$6D, 12D; K$=$5, 15$) and hidden sizes (256-1024). We also used de-convolutional architecture for $p(\boldsymbol{\tau}|\boldsymbol{x})$. Since $p(\boldsymbol{\tau}|\boldsymbol{x})$ was one of the key parts for BO we used 4 layers with [512, 256, 128, 128] channels (though a smaller CNN could have sufficed). For $p(y|\psi)$ we used a 2-layer MLP (hidden size 64). Training took ≈30-180 minutes on 1 GPU, using 1e-4 learning rate (decayed to 1e-5). We note that other advanced architectures like RNNs, LSTMs and Quasi-RNNs [36] did not result in reliably robust training in our experiments.

6 Locomotion Experiments on the Daisy Hexapod

For locomotion experiments, we use a Daisy robot (Figure 3) from Hebi robotics [37]. It has six legs, each with 3 motors – base, shoulder and elbow. A Vive tracking system measures the robot's position in a global frame for rewards. To obtain simulated trajectories for training SVAE we used PyBullet [38]. The simulator was fast, but did not have an accurate contact model with the ground. While free-space motion of individual joints transferred to hardware, the overall behavior of the robot when interacting with the ground was very different between simulation and hardware. As a result, rewards obtained by controllers in simulation could be significantly different on hardware.

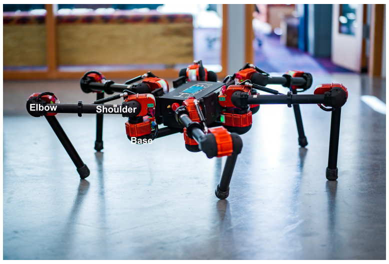


Figure 3: Daisy hexapod used in this work.

Daisy Controllers: We used Central Pattern Generators (CPGs) from [39]. CPGs are a variant of rhythmic DMPS [13], capable of generating a large number of locomotion gaits by changing the frequency, amplitude, and offset of each joint, as well as the relative phase differences between joints. Different CPG parameters can be restricted to obtain controllers with various dimensionalities. We experimented with 11D controller on hardware and 27D in simulation. For hardware, we assume that all joints have the same amplitude, frequency and offset (3 parameters), all base motors have independent phases (6 parameters), all shoulders and elbows have the same phase difference w.r.t. the base (2 parameters). This assumption implies that all joints are treated identically, which doesn't always hold, since each motor has slightly different tracking and bandwidth. In the future, we would like to use alternatives that allow each motor to learn independently. For simulation: base, shoulder and elbow joints were allowed to have independent amplitudes, frequencies and offsets, but fixed across the six legs (9 parameters); each of the 18 joints was allowed to have an independent phase (18 parameters).

Daisy Hardware Experiments: To construct SVAE-DC kernel for BO we trained SVAE using 500, 000 simulated trajectories (1000 time steps each, ≈16.5s). For dynamic compression the states were marked as undesirable if they had: high joint velocities (more than 10rad/sec); robot base tilting by more than 60°in roll and pitch, elbows hitting the ground; height of the base outside of [0.1, 0.7]cm from the ground. These aimed to reduce the chance of the robot breaking: controllers with high joint velocities can harm the motors on impact with the ground; tilting the torso can cause the robot to fall on its back; scraping the ground or lifting off and then falling can cause further damage. Since our BO trials were in a narrow walkway, we marked as undesirable states deviating more than 0.5m from the starting x-coordinate of the base.
The objective for BO was: $f(\boldsymbol{x}) = 10 \cdot y_{final} - N_{high\_vel}$, where y_{final} was the final y-coordinate of the robot (how much the robot walked forward), N_{high_vel} was the number of timesteps with velocities exceeding 10rad/sec. All BO experiments used UCB acquisition function (with $\beta = 1$). We completed 5 runs of BO on the Daisy robot hardware, initializing with 2 random samples, followed by 10 trials of BO (Figure 4). We also conducted baseline experiments with SE kernel by directly searching in the space of CPG parameters. This served as a comparison to more traditional trajectory compression methods that optimize DMPs

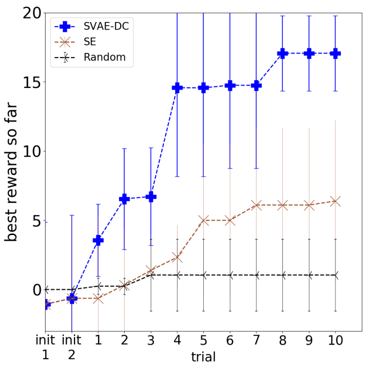


Figure 4: BO on Daisy hardware (means over 5 runs, 90% CIs).

(since CPG can be seen as a DMP variant). For Daisy robot, the controller would be considered acceptable if it walked forward for more than $1.5m$ during a trial of 25 seconds on hardware. For comparison to random search we sampled 60 controllers at random. Of these only 2 were able to walk forward a distance of over $1.5m$ in $25s$. So the problem was challenging, as the chance of randomly sampling a successful controller was <4%. BO with SVAE-DC kernel found walking controllers reliably in all 5/5 runs within fewer than 10 trials. In contrast, both BO with SE found forward walking controllers only in 2/5 runs.

Further experiments in simulation: We created an artificial 'sim-to-real' gap, allowing to gauge the potential for simulation-based kernels without running all the experiments on hardware. For each BO run we randomly sampled ground restitution parameters, and kept them fixed for all trials within a run. Hence, simulation-based kernels did not have full information about the exact properties of the environment used during BO. The range of parameters was the same for BO and for data collection, so informed kernels could identify controllers that perform well on average across settings. But such informed kernel could have caused negative transfer by lagging to identify controllers best for a particular BO setting, and instead favoring conservative (crawling) best-across-settings controllers. Figure 5 shows BO with 27D controller. BO with SVAE-DC outperformed all baselines. BBK-KL kernel obtained smaller improvements over SE and Random baselines. This indicated that a trajectory-based kernel was useful even when optimizing a high-dimensional controller, although BBK-KL benefits were greatly diminished compared to BBK results for 2-4 dimensional controllers reported in prior work. In these experiments, SVAE without dynamic compression was very similar to SE (omitted from the plot for clarity, since it was overlapping with SE). This showed that dimensionality reduction alone does not guarantee improvement (even when the latent space contains information needed to decode back into the space of original trajectories).

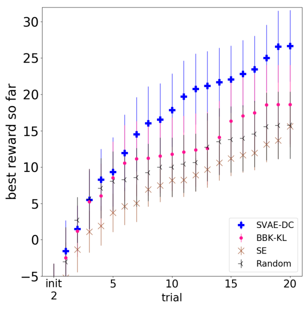


Figure 5: BO for Daisy in simulation (means over 50 runs, 90% CIs).

7 Manipulation Experiments

Our manipulation task was to push two objects from one side of the table to another without tipping them over. We used ABB Yumi robot for our hardware experiments (Figure 6), and conducted additional simulation experiments with Yumi and Franka Emika robot models. We used PyBullet for simulation. For Yumi environment the objects had mass and inertial properties similar to paper towel rolls (mass of 150g, 22cm height, 5cm radius); for Franka these had properties similar to wooden rolls (2kg, 22cm height, 8cm radius). Compared to 'push-to-target' task, our task had two different challenges. The objects were likely to come into contact with each other (not only the robot arm). Moreover, they could easily tip over, especially if forces were applied above an object's center of mass. Reward was given only at the end of the task: the distance each upright object moved in the desired direction minus a penalty for objects that tipped over (with y_{max} being table width): $f(\boldsymbol{x}) = \sum_i \left[(y^{obj_i}_{final} - y^{obj_i}_{start}) \mathbb{1}_{obj_i \in Up} - y_{max} \mathbb{1}_{obj_i \in Tipped} \right]$.

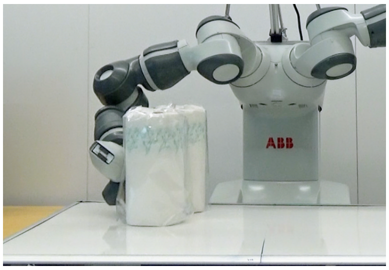


Figure 6: "Stable push" task with Yumi

Controllers: We tested our approach on two types of controllers: 1) joint velocity controller suitable for robots like ABB Yumi and 2) torque controller suitable for robots like Franka Emika. The first was parameterized by 6 joint velocity "waypoints", one target velocity for each joint of the robot arm (so $6 \cdot 7 = 42$ parameters for a 7DoF arm). Each "waypoint" also had a duration parameter that specified the fraction of time to be spent attaining the desired joint velocities. Overall this yielded a 48-dimensional parametric controller. The second controller type was aimed to be safe to use on robots with torque control that are more powerful than ABB Yumi. Instead of exploring randomly in torque space, we designed a parametric controller with desired waypoints in end-effector space. Each of the 6 waypoints had 6 parameters for the pose (3D position, 3D orientation) and 2 parameters for controller proportional and derivative gains. Overall this yielded a 48-dimensional parametric controller: $6 \cdot (6+2)$. This controller interpolated between the waypoints using a 5^{th} order minimum jerk trajectory for positions, and used linear interpolation for orientations. End effector Jacobian for the corresponding robot model was used to convert to joint torques.

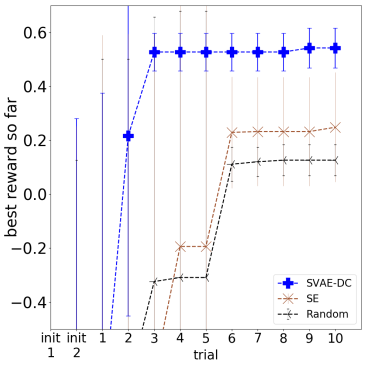


Figure 7: BO on ABB Yumi hardware (means over 5 runs, 90% CIs).

Yumi Hardware Experiments: For constructing SVAE-DC kernel used during BO on hardware we simulated 500,000 trajectories. These contained joint angles of the robot and object poses at each time step (1000 steps per trajectory). A step t on a trajectory $\boldsymbol{\xi}$ was marked as undesirable $\left(G_{bad}(\boldsymbol{\xi}_t) = 1\right)$ when: any object tipped over or was pushed beyond the table; robot collided with the table; the end effector was outside of main workspace (not over the table area). Mass, friction and restitution of the objects were randomized at the start of each episode/trajectory. Randomization ranges were set to roughly resemble variability of how real-world objects behaved. ABB Yumi robot available to us could operate effectively only at low velocities ($\frac{1}{5}$ of simulation maximum). High-velocity trajectories successful in simulation yielded different results on hardware. To prevent Yumi from shutting down due to high load

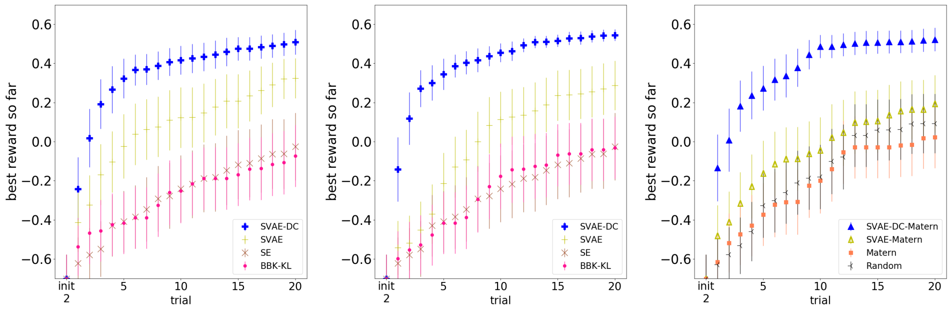


Figure 8: BO with various kernels on Franka Emika simulation. Left: SVAE trained with same parameters as in all the previous experiments. Middle: SVAE with larger latent space and NNs. Right: Matern used as outer function for all kernel. The plots show means over 50 runs, 90% CIs.

we stopped execution if the robot's arm extended too far outside the main workspace, also stopped if it was about to collide with the table (giving $-2y_{max}$ reward in such cases). These factors caused a large sim-real gap. Nonetheless, BO with SVAE-DC kernel was still able to significantly outperform BO with SE (Figure 7). Even when controllers successful in simulation yielded very different outcomes on hardware, SVAE-DC kernel was still able to find well-performing alternatives (more conservative, yet successful on hardware).

Further Yumi and Franka experiments in simulation: We emulated 'sim-to-real' gap as with Daisy simulation: sampled different object properties (mass, friction, restitution) at the start of each BO run. Results in Figure 9 show that BO on Yumi with SVAE-DC kernel yielded substantial improvement over all baselines. BO in the latent space of SVAE (without dynamic compression) was also able to substantially outperform all baselines, matching SVAE-DC gains after ≈15 trials.

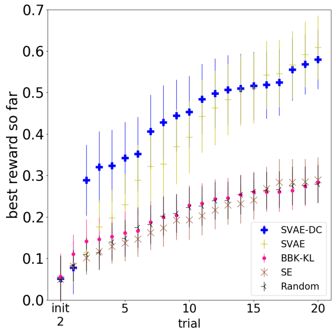


Figure 9: BO on ABB Yumi simulation (mean of 50 runs, 90% CIs).

Figure 8 shows BO results on Franka Emika simulation (left). Kernels were built in the same way as for Yumi, but from shorter trajectories (500 steps). Furthermore, we analyze how increasing the size of SVAE latent space and NNs impacts performance (middle). The larger latent space is 6·5=30D (vs 9D in other experiments), the hidden layer size of NNs is increased from 128 to 256. Larger latent space implies larger search space for BO, which could impair data efficiency. BO with SVAE kernel (no DC) still outperforms BBK-KL and SE kernels, but only after 10 trials. BO with SVAE-DC offers immediate gains with low variance between runs (well-performing points are found more reliably). This indicates that dynamic compression could counter-balance increase in kernel dimensionality. Finally, we experimented with Matérn kernel (right plot in Figure 8), but it did not show benefits over using SE kernel. We attempted changing hyperparameter prior and restricting hyperparameter ranges, but it did not consistently outperform random search (same held for SE in high dimensions). The performance of BO with SVAE kernel using Matérn as outer kernel function showed modest improvement over baselines. In contrast, BO with SVAE-DC kernel still offered substantial improvements.

8 Conclusion

In this work, we employed BO to optimize robot controllers with a small budget of trials. Previously, the success of BO has been either limited to low-dimensional controllers or required kernels with domain-specific features. We proposed an unsupervised alternative with sequential variational autoencoder. We used it to embed simulated trajectories into a latent space, and to jointly learn relating controllers with latent space paths they induce. Furthermore, we provided a mechanism for dynamic compression, helping BO reject undesirable regions quickly, and explore more in other regions. Our approach yielded ultra-data efficient BO in hardware experiments with hexapod locomotion and a manipulation task, using the same SVAE-DC architecture and training settings across experiments.

Acknowledgments

This research was supported in part by the Knut and Alice Wallenberg Foundation.

References

[1] OpenAI. Learning Dexterous In-Hand Manipulation. *arXiv:1808.00177*, 2018.

[2] A. Wilson, A. Fern, and P. Tadepalli. Using Trajectory Data to Improve Bayesian Optimization for Reinforcement Learning. *Journal of Machine Learning Research*, 15 (1):253–282, 2014.

[3] M. Balandat, B. Karrer, D. Jiang, B. Letham, S. Daulton, A. Wilson, E. Bakshy. BoTorch. `https://botorch.org/`. Accessed: 2019-05.

[4] B. Shahriari, K. Swersky, Z. Wang, R.P. Adams, N. de Freitas. Taking the Human Out of the Loop: A Review of Bayesian Optimization. *Proceedings of the IEEE*, 104 (1):148–175, 2016.

[5] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv:1312.6114*, 2013.

[6] R. Gómez-Bombarelli, J. N. Wei, D. Duvenaud, J. M. Hernández-Lobato, B. Sánchez-Lengeling, D. Sheberla, J. Aguilera-Iparraguirre, T. D. Hirzel, R. P. Adams, and A. Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, 4(2):268–276, 2018.

[7] T. Lesort, N. Díaz-Rodríguez, J.-F. Goudou, and D. Filliat. State representation learning for control: An overview. *Neural Networks*, 2018.

[8] L. Yingzhen and S. Mandt. Disentangled sequential autoencoder. In *International Conference on Machine Learning*, pages 5656–5665, 2018.

[9] M. Fraccaro, S. Kamronn, U. Paquet, and O. Winther. A disentangled recognition and nonlinear dynamics model for unsupervised learning. In *Advances in Neural Information Processing Systems*, pages 3601–3610, 2017.

[10] M. Deisenroth and C. E. Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472, 2011.

[11] R. Calandra, J. Peters, C. E. Rasmussen, and M. P. Deisenroth. Manifold gaussian processes for regression. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 3338–3345. IEEE, 2016.

[12] A. G. Wilson, Z. Hu, R. Salakhutdinov, and E. P. Xing. Deep kernel learning. In *Artificial Intelligence and Statistics*, pages 370–378, 2016.

[13] P. Pastor, L. Righetti, M. Kalakrishnan, and S. Schaal. Online movement adaptation based on previous sensor experiences. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 365–371. IEEE, 2011.

[14] N. Thatte, H. Duan, and H. Geyer. A method for online optimization of lower limb assistive devices with high dimensional parameter spaces. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–6. IEEE, 2018.

[15] S. Feng, E. Whitman, X. Xinjilefu, and C. G. Atkeson. Optimization-based Full Body Control for the DARPA Robotics Challenge. *Journal of Field Robotics*, 32(2):293–312, 2015.

[16] Y. Gong, R. Hartley, X. Da, A. Hereid, O. Harib, J.-K. Huang, and J. Grizzle. Feedback control of a cassie bipedal robot: Walking, standing, and riding a segway. *arXiv:1809.07279*, 2018.

[17] R. Calandra. *Bayesian Modeling for Optimization and Control in Robotics*. PhD thesis, Darmstadt University of Technology, Germany, 2017.

[18] D. J. Lizotte, T. Wang, M. H. Bowling, and D. Schuurmans. Automatic Gait Optimization with Gaussian Process Regression. In *International Joint Conference on Artificial Intelligence (IJCAI)*, volume 7, pages 944–949, 2007.

[19] M. Tesch, J. Schneider, and H. Choset. Using response surfaces and expected improvement to optimize snake robot gait parameters. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1069–1074. IEEE, 2011.

[20] A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret. Robots that can adapt like animals. *Nature*, 521(7553):503–507, 2015.

[21] A. Rai, R. Antonova, F. Meier, and C. G. Atkeson. Using simulation to improve sample-efficiency of bayesian optimization for bipedal robots. *Journal of Machine Learning Research*, 20(49):1–24, 2019.

[22] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. *arXiv:1804.10332*, 2018.

[23] T. Li, A. Rai, H. Geyer, and C. G. Atkeson. Using deep reinforcement learning to learn high-level policies on the atrias biped. *arXiv:1809.10811*, 2018.

[24] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[25] T. Haarnoja, S. Ha, A. Zhou, J. Tan, G. Tucker, and S. Levine. Learning to walk via deep reinforcement learning. In *Robotics: Science and Systems (RRS)*, 2019.

[26] O. Kroemer, R. Detry, J. Piater, and J. Peters. Combining active learning and reactive control for robot grasping. *Robotics and Autonomous Systems*, 58(9):1105–1116, 2010.

[27] L. Montesano and M. Lopes. Active learning of visual descriptors for grasping using non-parametric smoothed beta distributions. *Robotics and Autonomous Systems*, 60 (3):452–462, 2012.

[28] R. Antonova, M. Kokic, J. A. Stork, and D. Kragic. Global search with bernoulli alternation kernel for task-oriented grasping informed by simulation. In *Conference on Robot Learning*, pages 641–650, 2018.

[29] 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. *arXiv:1810.05687*, 2018.

[30] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018.

[31] I. Arnekvist, D. Kragic, and J. A. Stork. VPE: Variational Policy Embedding for Transfer Reinforcement Learning. In *2019 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2019.

[32] T. Minka. Divergence measures and message passing. *TR-2005-173 Microsoft Research*, 2005.

[33] C. M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[34] C. Riquelme, M. Johnson, and M. Hoffman. Failure modes of variational inference for decision making. *Prediction and Generative Modeling in RL Workshop (AAMAS, ICML, IJCAI)*, 2018.

[35] S. Tschiatschek, K. Arulkumaran, J. Stühmer, and K. Hofmann. Variational inference for data-efficient model learning in pomdps. *TR-2018-15 Microsoft Research*, 2018.

[36] J. Bradbury, S. Merity, C. Xiong, and R. Socher. Quasi-recurrent neural networks. *International Conference on Learning Representations*, 2017.

[37] Hebi Robotics. `http://docs.hebi.us`. Accessed: 2019-06.

[38] Pybullet simulator. `https://github.com/bulletphysics/bullet3`. Accessed: 2019-06.

[39] A. Crespi and A. J. Ijspeert. Online optimization of swimming and crawling in an amphibious snake robot. *IEEE Transactions on Robotics*, 24(1):75–87, 2008.

Using Simulation to Improve Sample-Efficiency of Bayesian Optimization for Bipedal Robots

Akshara Rai[1]
Robotics Institute, School of Computer Science
Carnegie Mellon University, PA, USA

Rika Antonova[1]
Robotics, Perception and Learning, EECS
KTH Royal Institute of Technology, Stockholm, Sweden

Franziska Meier
Paul G. Allen School of Computer Science & Engineering
University of Washington, Seattle, WA, USA

Christopher G. Atkeson
Robotics Institute, School of Computer Science
Carnegie Mellon University, PA, USA

[1]Both of these authors contributed equally.

Editor: Bayesian Optimization Special Issue
Journal of Machine Learning Research 20 (2019) 1-24
Submitted 4/18; Revised 12/18; Published 2/19; License: CC-BY 4.0
https://www.jmlr.org/papers/volume20/18-196/18-196.pdf

Abstract

Learning for control can acquire controllers for novel robotic tasks, paving the path for autonomous agents. Such controllers can be expert-designed policies, which typically require tuning of parameters for each task scenario. In this context, Bayesian optimization (BO) has emerged as a promising approach for automatically tuning controllers. However, sample-efficiency can still be an issue for high-dimensional policies on hardware. Here, we develop an approach that utilizes simulation to learn structured feature transforms that map the original parameter space into a domain-informed space. During BO, similarity between controllers is now calculated in this transformed space. Experiments on the ATRIAS robot hardware and simulation show that our approach succeeds at sample-efficiently learning controllers for multiple robots. Another question arises: What if the simulation significantly differs from hardware? To answer this, we create increasingly approximate simulators and study the effect of increasing simulation-hardware mismatch on the performance of Bayesian optimization. We also compare our approach to other approaches from literature, and find it to be more reliable, especially in cases of high mismatch. Our experiments show that our approach succeeds across different controller types, bipedal robot models and simulator fidelity levels, making it applicable to a wide range of bipedal locomotion problems.

Keywords: Bayesian Optimization, Bipedal Locomotion, Transfer Learning

1 Introduction

Machine learning can provide methods for learning controllers for robotic tasks. Yet, even with recent advances in this field, the problem of automatically designing and learning controllers for robots, especially bipedal robots, remains difficult. It is expensive to do learning experiments that require a large number of samples with physical robots. Specifically, legged robots are not robust to falls and failures, and are time-consuming to work with and repair. Furthermore, commonly used cost functions for optimizing controllers are noisy to evaluate, non-convex and non-differentiable. In order to find learning approaches that can be used on real robots, it is thus important to keep these considerations in mind.

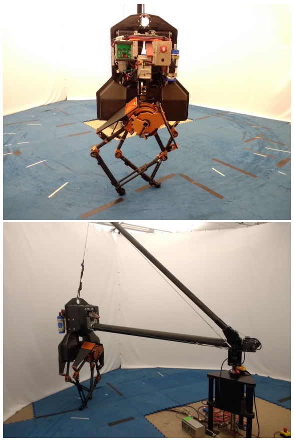


Figure 1: ATRIAS robot.

Deep reinforcement learning approaches can deal with noise, discontinuities and non-convexity of the objective, but they are not data-efficient. These approaches could take on the order of a million samples to learn locomotion controllers [23], which would be infeasible on a real robot. For example, on the ATRIAS robot, $10,000$ samples would take 7 days, in theory. But practically, the robot needs to be “reset" between trials and repaired in case of damage. Using structured expert-designed policies can help minimize damage to the robot and make the search for successful controllers feasible. However, the problem is black-box, non-convex and discontinuous. This eliminates approaches like PI^2 [33] which make assumptions about the dynamics of the system and PILCO [8] which assumes a continuous cost landscape. Evolutionary approaches like CMA-ES [13] can still be prohibitively expensive, needing thousands of samples [29].

In comparison, Bayesian optimization (BO) is a sample-efficient optimization technique that is robust to non-convexity and noise. It has been recently used in a range of robotics problems, such as Calandra et al. [6], Marco et al. [19] and Cully et al. [7]. However, sample-efficiency of conventional BO degrades in high dimensions, even for dimensionalities commonly encountered in locomotion controllers. Because of this, hardware-only optimization becomes intractable for flexible controllers and complex robots. One way of addressing this issue is to utilize simulation to optimize controller parameters. However, simulation-only optimization is vulnerable to learning policies that exploit the simulation and, because of that, perform well in simulation but poorly on the actual robot. This motivates the development of hybrid approaches that can incorporate simulation-based information into the learning method and then optimize with few samples on hardware.

Towards this goal, our previous work in Antonova, Rai, and Atkeson [1], Antonova et al. [2], Rai et al. [24] presents a framework that uses information from high-fidelity simulators to learn sample-efficiently on hardware. We use simulation to build informed feature transforms that are used to measure controller similarity during BO. Thus, during optimization on hardware, the similarity between controller parameters is informed by how they perform in simulation. With this, it becomes possible to quickly infer which regions of the input space are likely to perform well on hardware. This method has been tested on the ATRIAS biped robot (Figure 1) and shows considerable improvement in sample-efficiency over traditional BO.

In this article, we present in-depth explanations and empirical analysis of our previous work. Furthermore, for the first time, we present a procedure for systematically evaluating robustness of such approaches to simulation-hardware mismatch. We extend our previous work incorporating mismatch estimates [24] to this setting. We also conduct extensive comparisons with competitive baselines from prior work, e.g. [7].

The rest of this article is organized as follows: Section 2 provides background for BO, then gives an overview of related work on optimizing locomotion controllers. Section 3.1 describes the idea of incorporating simulation-based transforms into BO; Section 3.2 explains how we handle simulation-hardware mismatch. Sections 4.1-4.5 describe the robot and controllers we use for our experiments; Section 4.6 explains the motivation and construction of simulators with various kinds of simulation-hardware mismatch. Section 5 summarizes hardware experiments on the ATRIAS robot. Section 5.2 shows generalization to a different robot model in simulation. Section 5.3 presents empirical analysis of the impact of simulator quality on the performance of the proposed algorithms and alternative approaches.

2 Background and Related Work

This section gives a brief overview of Bayesian optimization (BO), the state-of-the-art research on optimizing locomotion controllers, and utilizing simulation information in BO.

2.1 Background on Bayesian Optimization

Bayesian optimization (BO) is a framework for online, black-box, gradient-free global search ([26] and [4] provide a comprehensive introduction). The problem of optimizing controllers can be interpreted as finding controller parameters $\boldsymbol{x}^*$ that optimize some cost function $f(\boldsymbol{x})$. Here $\boldsymbol{x}$ contains parameters of a pre-structured policy; the cost $f(\boldsymbol{x})$ is a function of the trajectory induced by controller parameters $\boldsymbol{x}$. For brevity, we will refer to 'controller parameters $\boldsymbol{x}$' as 'controller $\boldsymbol{x}$'. We use BO to find controller $\boldsymbol{x}^*$, such that: $f(\boldsymbol{x}^*) = \min_{\boldsymbol{x}} f(\boldsymbol{x})$.

BO is initialized with a prior that expresses the *a priori* uncertainty over the value of $f(\boldsymbol{x})$ for each $\boldsymbol{x}$ in the domain. Then, at each step of optimization, based on data seen so far, BO optimizes an auxiliary function (called *acquisition function*) to select the next $\boldsymbol{x}$ to evaluate. The acquisition function balances exploration vs exploitation. It selects points for which the posterior estimate of the objective f is promising, taking into account both mean and covariance of the posterior. A widely used representation for the cost function f is a Gaussian process (GP):

$$f(\boldsymbol{x}) \sim \mathcal{GP}(\mu(\boldsymbol{x}), k(\boldsymbol{x}_i, \boldsymbol{x}_j))$$

The prior mean function $\mu(\cdot)$ is set to 0 when no domain-specific knowledge is provided. The kernel function $k(\cdot,\cdot)$ encodes similarity between inputs. If $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is large for inputs $\boldsymbol{x}_i, \boldsymbol{x}_j$, then $f(\boldsymbol{x}_i)$ strongly influences $f(\boldsymbol{x}_j)$. One of the most widely used kernel functions is the Squared Exponential (SE):

$$k_{SE}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sigma_k^2 \exp\big(-\tfrac{1}{2}(\boldsymbol{x}_i - \boldsymbol{x}_j)^T \operatorname{diag}(\boldsymbol{\ell})^{-2}(\boldsymbol{x}_i - \boldsymbol{x}_j)\big),$$

where σ_k^2, $\boldsymbol{\ell}$ are signal variance and a vector of length scales respectively. σ_k^2, $\boldsymbol{\ell}$ are referred to as 'hyperparameters' in the literature.

The SE kernel belongs to a broader class of Matérn kernels, which in general have more free parameters. In some cases, carefully choosing kernel parameters improves performance of BO [27]. However, domain-informed kernels can easily out-perform even well-tuned Matérn kernels [7]. SE and Matérn kernels are stationary: $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ depend only on $r = \boldsymbol{x}_i - \boldsymbol{x}_j \ \forall \boldsymbol{x}_{i,j}$. We aim to remove this limitation in a manner informed by simulation. In future work, incorporating other approaches that relax stationarity (e.g. [21]) could further improve BO.

2.2 Optimizing Locomotion Controllers

Parametric locomotion controllers can be represented as $\boldsymbol{u} = \pi_{\boldsymbol{x}}(\boldsymbol{s})$, where π is a policy structure that depends on parameters $\boldsymbol{x}$. For example, π can be parameterized by feedback gains on the center of mass (CoM), reference joint trajectories, etc. Vector $\boldsymbol{s}$ is the state of the robot, such as joint angles and velocities, used in closed-loop controllers. Vector $\boldsymbol{u}$ represents the desired control action, for example: torques, angular velocities or positions for each joint on the robot. The sequence of control actions yields a sequence of state transitions, which form the overall 'trajectory' $[\boldsymbol{s}_0, \boldsymbol{u}_1, \boldsymbol{s}_1, \boldsymbol{u}_2, \boldsymbol{s}_2, ...]$. This trajectory is used in the cost function to judge the quality of the controller $\boldsymbol{x}$. In our work, we use structured controllers designed by experts. State of the art research on walking robots featuring such controllers includes Feng et al. [10] and Kuindersma et al. [17]. The overall optimization then includes manually tuning the parameters $\boldsymbol{x}$. An alternative to manual tuning is to use evolutionary approaches, like CMA-ES, as in Song and Geyer [29]. However, these require a large number of samples and can usually be conducted only in simulation. Optimization in simulation can produce controllers that perform well in simulation, but not on hardware. In comparison, BO is a sample-efficient technique which has become popular for direct optimization on hardware. Recent successes include manipulation [9] and locomotion [6].

BO for locomotion has been previously explored for several types of mobile robots. These include: snake robots [31], AIBO quadrupeds [18], and hexapods [7]. [31] optimize a 3-dimensional controller for a snake robot in 10-40 trials (for speeds up to $0.13m/s$). [18] use BO to optimize gait parameters for a AIBO robot in 100-150 trials. [7] learn 36 controller parameters for a hexapod. Even with hardware damage, they can obtain successful controllers for speeds up to $0.4m/s$ in 12-15 trials.

Hexapods, quadrupeds and snakes spend a large portion of their gaits with their center of mass within their support polygon (convex hull of the feet on the ground). Ignoring velocity, this is statically stable. In contrast, bipedal walking can be highly dynamic, especially for point-feet robots like ATRIAS. ATRIAS, like most bipeds, spends a significant time of its gait being “unstable", or dynamically stable. In our experiments on hardware, ATRIAS goes up to speeds of $1m/s$. All of this leads to a challenging optimization setting and discontinuous cost function landscape. [6] use BO for optimizing gaits of a dynamic biped on a boom, needing 30-40 samples for finding walking gaits for a 4-dimensional controller. While this is promising, optimizing a higher-dimensional controller needed for complex robots would be even more challenging. If significant number of samples lead to unstable gaits and falls, they could damage the robot. Hence, it is important to develop methods that can learn complex controllers fast, without damaging the robot.

2.3 Incorporating Simulation Information into Bayesian Optimization

The idea of using simulation to speed up BO on hardware has been explored before. Information from simulation can be added as a prior to the GP used in BO, such as in [7]. While such methods can be successful, one needs to carefully tune the influence of simulation points over hardware points, especially when simulation is significantly different from hardware.

More generally, approaches such as [16] and [19] address trading off computation vs simulation accuracy when selecting the source for the next trial/evaluation (with real world viewed as the most expensive source). Instead, we consider a setting with ample compute resources for simulation, but an extremely small number of experiments on a real robot. This is appropriate for bipedal locomotion with full-scale robots, since these can be expensive to run. Hence, our work does not fall into the ‘multi-fidelity’ BO paradigm, and we instead take a two step approach: pre-computing information needed for kernel transforms in the first stage, then running an ultra-sample efficient version of BO in the second stage on a real robot.

Recently, several approaches proposed incorporating Neural Networks (NNs) into the Gaussian process (GP) kernels (Wilson et al. [36], Calandra et al. [5]). The strength of these approaches is that they can jointly update the GP and the NN. Calandra et al. [5] demonstrated how this added flexibility can handle discontinuities in the cost function landscape. However, these approaches do not directly address the problem of incorporating a large amount of data from simulation in hardware BO experiments.

Wilson et al. [35] explored enhancing GP kernel with trajectories. Their Behavior Based Kernel (BBK) computes an estimate of a symmetric variant of the KL divergence between trajectories induced by two controllers, and uses this as a distance metric in the kernel. However, getting an estimate would require samples for each controller $\boldsymbol{x}_i, \boldsymbol{x}_j$ whenever $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is needed. This can be impractical, as it involves an evaluation of every controller considered. To overcome this, the authors suggest combining BBK with a model-based approach. But building an accurate model from hardware data might be an expensive process in itself.

Cully et al. [7] utilize simulation by defining a behavior metric and collecting best performing points in simulation. This behavior metric then guides BO to quickly find controllers on hardware, and can even compensate for damage to the robot. The search on hardware is conducted in behavior space, and limited to pre-selected "successful" points from simulation. This helps make their search faster and safer on hardware. However, if an optimal point was not pre-selected, BO cannot sample it during optimization.

In our work, we utilize trajectories from simulation to build feature transforms that can be incorporated in the GP kernel used for BO. Our approaches incorporate trajectory/behavior information, but ensure that $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is computed efficiently during BO. Our work is related, in part, to input space warping, as described in [28], but we construct our transforms from simulated data. Our simulation-informed kernels bias the search towards regions that look promising, but are able to 'recover' and search in other parts of the space if simulation-hardware mismatch becomes apparent.

3 Proposed Approach: Bayesian Optimization with Informed Kernels

In this section, we offer in-depth explanation of approaches from our work in Antonova, Rai, and Atkeson [1], Antonova et al. [2], and Rai et al. [24]. This work proposes incorporating domain knowledge into BO with the help of simulation. We evaluate locomotion controllers in simulation, and collect their induced trajectories, which are then used to build an informed transform. This can be achieved by using a domain-specific feature transform (Section 3.1.1) or by learning to reconstruct short trajectory summaries (Section 3.1.2). This feature transform is used to construct an informed distance metric for BO, and helps BO discover promising regions faster. Figure 2 gives an overview. In Section 3.2 we discuss how to incorporate

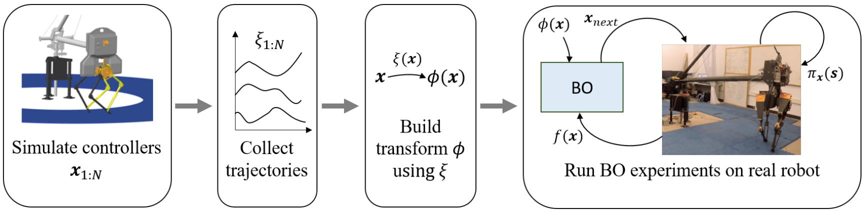


Figure 2: Overview of our proposed approach. Here, $\pi_{\boldsymbol{x}}(\boldsymbol{s})$ is the policy (Section 2.2); $\boldsymbol{x}$ is a vector of controller parameters; $\boldsymbol{s}$ is the state of the robot; $\xi(\boldsymbol{x})$ is a trajectory observed in simulation for $\boldsymbol{x}$; $\phi(\cdot)$ is the transform built using $\xi(\cdot)$; $f(\boldsymbol{x})$ is the cost of $\boldsymbol{x}$ evaluated on hardware. BO uses $\phi(\boldsymbol{x})$ and evaluated costs $f(\boldsymbol{x})$ to propose next promising controller x_{next}.

simulation-hardware mismatch in to the transform, ensuring that BO can benefit from inaccurate simulations as well.

3.1 Constructing Flexible Kernels using Simulation-based Transforms

High dimensional problems with discontinuous cost functions are very common with legged robots, where slight changes to some parameters can make the robot unstable. Both of these factors can adversely affect BO's performance, but informed feature transforms can help BO sample high-performing controllers even in such scenarios.

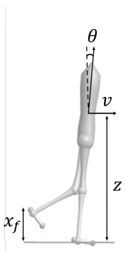
In this section, we demonstrate how to construct such transforms $\phi(\boldsymbol{x})$ utilizing simulations for a given controller $\boldsymbol{x}$. We then use ϕ to create an informed kernel $k_\phi(\boldsymbol{x}_i, \boldsymbol{x}_j)$ for BO on hardware:

$$\begin{aligned} \boldsymbol{t}_{ij} &= \phi(\boldsymbol{x}_i) - \phi(\boldsymbol{x}_j) \\ k_\phi(\boldsymbol{x}_i, \boldsymbol{x}_j) &= \sigma_k^2 \exp\Big(-\tfrac{1}{2}\boldsymbol{t}_{ij}^T \operatorname{diag}(\boldsymbol{\ell})^{-2} \boldsymbol{t}_{ij} \Big) \end{aligned} \tag{1}$$

Note that the functional form above is the same as that of Squared Exponential kernel, if considered from the point of view of the transformed space, with $\phi(\boldsymbol{x})$ as input. Using a low-dimensional ϕ could improve sample efficiency by reducing dimensionality. More notably, crucial gains arise when ϕ brings controllers that perform similar in simulation closer together, as compared to the original parameter space. For locomotion, this could bring failing controllers close together to occupy only a small portion of the transformed space, as illustrated in [24]. In essence, this means that the resultant kernel, though stationary in ϕ, is non-stationary in $\boldsymbol{x}$.

3.1.1 The Determinants of Gait Transform

We propose a feature transform for bipedal locomotion derived from physiological features of human walking called Determinants of Gaits (DoG) [15]. ϕ_{DoG} was originally developed for human-like robots and controllers [1], and then generalized to be applicable to a wider range of bipedal locomotion controllers and robot morphologies [24]. It is based on the features in Table B.1.



M_1 (Swing leg retraction) – If the maximum ground clearance of the swing foot x_f is more than a threshold, $M_1 = 1$ (0 otherwise); ensures swing leg retraction.

M_2 (Center of mass height) – If CoM height z stays about the same at the start and end of a step, $M_2 = 1$ (0 otherwise); checks that the robot is not falling.

M_3 (Trunk lean) – If the average trunk lean θ is the same at the start and end of a step, $M_3 = 1$ (0 otherwise); ensures that the trunk is not changing orientation.

M_4 (Average walking speed) – Average forward speed v of a controller per step, $M_4 = v_{avg}$; M_4 helps distinguish controllers that perform similar on M_{1-3}.

Table B.1: Illustration of the features used to construct DoG transform.

ϕ_{DoG} combines features M_{1-4} per step and scales them by the normalized simulation time to obtain the DoG score of controller $\boldsymbol{x}$:

$$score_{DoG} = \frac{t_{sim}}{t_{max}} \cdot \sum_{s=1}^{N} \sum_{k=1}^{4} M_{ks} \tag{2}$$

Here, subscript k identifies the feature metric, s is the step number, N is the total number of steps taken in simulation, t_{sim} is time at which simulation terminated (possibly due to a fall), t_{max} is total time allotted for simulation. Since larger number of steps lead to higher DoG, some controllers that chatter (step very fast before falling) could get misleadingly high scores; we scale the scores by $\frac{t_{sim}}{t_{max}}$ to prevent that. $\phi_{DoG}(\boldsymbol{x})$ for controller parameters $\boldsymbol{x}$ now becomes the computed $score_{DoG}$ of the resulting trajectories when $\boldsymbol{x}$ is simulated. ϕ_{DoG} essentially aids in (soft) clustering of controllers based on their behaviour in simulation. High scoring controllers are more likely to walk than low scoring ones. Since M_{1-4} are based on intuitive gait features, they are more likely to transfer between simulation and hardware, as compared to direct cost. The thresholds in M_{1-3} are chosen according to values observed in nominal human walking from [37].

3.1.2 Learning a Feature Transform with a Neural Network

While domain-specific feature transforms can be extremely useful and robust, they might be difficult to generate when a domain expert is not present. This motivates directly learning such feature transforms from trajectory data. In this section we describe our approach to train neural networks to reconstruct trajectory summaries [2] that achieves this goal of minimizing expert involvement.

Trajectory summaries are a convenient choice for reparametrizing controllers into an easy to optimize space. For example, controllers that fall would automatically be far away from controllers that walk. If these trajectories can be extracted from a high-fidelity simulator, we would not have to evaluate each controller on hardware. However, conventional implementations of BO evaluate the kernel function for a large number of points per iteration, requiring thousands of simulations each iteration. To avoid this, a Neural Network (NN) can be trained to reconstruct trajectory summaries from a large set of pre-sampled data points. NN provides flexible interpolation, as well as fast evaluation (controller $\rightarrow$ trajectory summary). Furthermore, trajectories are agnostic to the specific cost used during BO. Thus the data collection can be done offline, and there is no need to re-run simulations in case the definition of the cost is modified.

We use the term 'trajectory' in a general sense, referring to several sensory states recorded during a simulation. To create trajectory summaries for the case of locomotion, we include measurements of: walking time (time before falling), energy used during walking, position of the center of mass and angle of the torso. With this, we construct a dataset for NN to fit: a Sobol grid of controller parameters ($\boldsymbol{x}_{1:N}$, $N\approx0.5$ million) along with trajectory summaries $\xi_{\boldsymbol{x}_i}$ from simulation. NN is trained using mean squared loss:

NN input: $\boldsymbol{x}$ – a set of controller parameters
NN output: $\phi_{trajNN}(\boldsymbol{x}) = \hat{\xi}_{\boldsymbol{x}}$ – reconstructed trajectory summary
NN loss: $\frac{1}{2}\sum_{i=1}^{N}||\hat{\xi}_{\boldsymbol{x}_i} - \xi_{\boldsymbol{x}_i}||^2$

The outputs $\phi_{trajNN}(\boldsymbol{x})$ are then used in the kernel for BO:

$$k_{trajNN}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sigma_k^2 \exp\left(-\tfrac{1}{2}\boldsymbol{t}_{ij}^T \operatorname{diag}(\boldsymbol{\ell})^{-2}\boldsymbol{t}_{ij}\right) \; ; \; \boldsymbol{t}_{ij} = \phi_{trajNN}(\boldsymbol{x}_i) - \phi_{trajNN}(\boldsymbol{x}_j) \quad (3)$$

Appendix A describes our data collection and training. We did not carefully select the sensory traces used in the trajectory summaries. Instead, we used the most obvious states, aiming for an approach that could be easily adapted to other domains. To apply this approach to a new setting, one could simply include information that is customarily tracked, or used in costs. For example, for a manipulator, the coordinates of the end effector(s) could be recorded at relevant points. Force-torque measurements could be included, if available.

3.2 Kernel Adjustment for Handling Simulation-Hardware Mismatch

Approaches described in previous sections could provide improvement for BO when a high-fidelity simulator is used in kernel construction. In Rai et al. [24] we presented promising results of experimental evaluation on hardware. However, it is unclear how the performance changes when simulation-hardware mismatch becomes apparent.

In Rai et al. [24], we also proposed a way to incorporate information about simulation-hardware mismatch into the kernel from the samples evaluated so far. We augment the simulation-based kernel to include this additional information about mismatch, by expanding the original kernel by an extra dimension that contains the predicted mismatch for each controller $\boldsymbol{x}$.

A separate Gaussian process is used to model the mismatch experienced on hardware, starting from an initial prior mismatch of 0: $g(\boldsymbol{x}) \sim \mathcal{GP}(0, k_{SE}(\boldsymbol{x}_i, \boldsymbol{x}_j))$. For any evaluated controller $\boldsymbol{x}_i$, we can compute the difference between $\phi(\boldsymbol{x}_i)$ in simulation and on hardware: $d_{\boldsymbol{x}_i} = \phi^{sim}(\boldsymbol{x}_i) - \phi^{hw}(\boldsymbol{x}_i)$. We can now use mismatch data $\{d_{\boldsymbol{x}_i} | i = 1...n\}$ to construct a model for the expected mismatch: $\bar{g}(\boldsymbol{x})$. In the case of using a GP-based model, $\bar{g}(\cdot)$ would denote the posterior mean. With this, we can predict simulation-hardware mismatch in the original space of controller parameters for unevaluated controllers. Combining this with kernel k_ϕ we obtain an adjusted kernel:

$$\boldsymbol{\phi}_{\boldsymbol{x}}^{adj} = \begin{bmatrix} \phi^{sim}(\boldsymbol{x}) \\ \bar{g}(\boldsymbol{x}) \end{bmatrix}, \qquad \boldsymbol{t}_{ij}^{adj} = \boldsymbol{\phi}_{\boldsymbol{x}_i}^{adj} - \boldsymbol{\phi}_{\boldsymbol{x}_j}^{adj}$$
$$k_{\phi_{adj}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sigma_k^2 \exp\Big(-\tfrac{1}{2} (\boldsymbol{t}_{ij}^{adj})^T \operatorname{diag}\Big(\begin{bmatrix} \boldsymbol{\ell_1} \\ \boldsymbol{\ell_2} \end{bmatrix} \Big)^{-2} \boldsymbol{t}_{ij}^{adj} \Big) \tag{4}$$

The similarity between points $\boldsymbol{x}_i, \boldsymbol{x}_j$ is now dictated by two components: representation in ϕ space and expected mismatch. This construction has an intuitive explanation: Suppose controller $\boldsymbol{x_i}$ results in walking when simulated, but falls during hardware evaluation. $k_{\phi_{adj}}$ would register a high mismatch for $\boldsymbol{x}_i$. Controllers would be deemed similar to $\boldsymbol{x}_i$ only if they have both similar simulation-based $\phi(\cdot)$ and similar estimated mismatch. Points with similar simulation-based $\phi(\cdot)$ and low predicted mismatch would still be 'far away' from the failed $\boldsymbol{x}_i$. This would help BO sample points that still have high chances of walking in simulation, but are in a different region of the original parameter space. In the next section, we present a more mathematically rigorous interpretation for $k_{\phi_{adj}}$.

3.2.1 Interpretation of Kernel with Mismatch Modeling

Let us consider a controller $\boldsymbol{x}_i$ evaluated on hardware. The difference between simulation-based and hardware-based feature transform for $\boldsymbol{x}_i$ is $d_{\boldsymbol{x}_i} = \phi^{sim}(\boldsymbol{x}_i) - \phi^{hw}(\boldsymbol{x}_i)$. The 'true' hardware feature transform for $\boldsymbol{x_i}$ is $\phi^{hw}(\boldsymbol{x_i}) = \phi^{sim}(\boldsymbol{x_i}) - d_{\boldsymbol{x_i}}$. After n evaluations on hardware, $\{d_{\boldsymbol{x}_i} | i = 1...n\}$ can serve as data for modeling simulation-hardware mismatch. In principle, any data-efficient model $g(\cdot)$ can be used, such as GP (a multi-output GP in case $\phi(\cdot) > 1D$). With this, we can obtain an adjusted transform: $\hat{\phi}^{hw}(\boldsymbol{x}) = \phi^{sim}(\boldsymbol{x}) - \bar{g}(\boldsymbol{x})$, where $\bar{g}(\cdot)$ is the output of the model fitted using $\{d_{\boldsymbol{x}_i} | i = 1, ...n\}$.

Suppose $\boldsymbol{x}_{new}$ has not been evaluated on hardware. We can use $\hat{\phi}^{hw}(\boldsymbol{x}_{new}) = \phi^{sim}(\boldsymbol{x}_{new}) - \bar{g}(\boldsymbol{x}_{new})$ as the adjusted estimate of what the output of ϕ should be, taking into account what we have learned so far about simulation-hardware mismatch.

Let's construct kernel $k_{\phi_{adj}}^{v2}(\boldsymbol{x}_i, \boldsymbol{x}_j)$ that uses these hardware-adjusted estimates directly:

$$\begin{aligned}
\boldsymbol{q}_{ij}^{adj} &= \hat{\phi}^{hw}(\boldsymbol{x}_i) - \hat{\phi}^{hw}(\boldsymbol{x}_j) \\
&= (\phi^{sim}(\boldsymbol{x}_i) - \bar{g}(\boldsymbol{x}_i)) - (\phi^{sim}(\boldsymbol{x}_j) - \bar{g}(\boldsymbol{x}_j)) \\
&= \underbrace{(\phi^{sim}(\boldsymbol{x}_i) - \phi^{sim}(\boldsymbol{x}_j))}_{\boldsymbol{v}_\phi} - \underbrace{(\bar{g}(\boldsymbol{x}_i) - \bar{g}(\boldsymbol{x}_j))}_{\boldsymbol{v}_g}
\end{aligned}$$

$$\begin{aligned}
k_{\phi_{adj}}^{v2}(\boldsymbol{x}_i, \boldsymbol{x}_j) &= \sigma_{k_{v_0}}^2 \exp\Big(- \tfrac{1}{2} (\boldsymbol{q}_{ij}^{adj})^T \operatorname{diag}(\boldsymbol{\ell})^{-2} \boldsymbol{q}_{ij}^{adj} \Big) \\
&= \sigma^2 \exp\Big(- \tfrac{1}{2} \big[(\boldsymbol{v}_\phi - \boldsymbol{v}_g)^T \operatorname{diag}(\boldsymbol{\ell})^{-2} (\boldsymbol{v}_\phi - \boldsymbol{v}_g) \big] \Big) \\
&= \sigma^2 \exp\Big(- \tfrac{1}{2} \big[\boldsymbol{v}_\phi^T \operatorname{diag}(\boldsymbol{\ell})^{-2} \boldsymbol{v}_\phi + \boldsymbol{v}_g^T \operatorname{diag}(\boldsymbol{\ell})^{-2} \boldsymbol{v}_g - prod_{ij} \big] \Big) \\
&\qquad \text{where } prod_{ij} = 2\boldsymbol{v}_\phi^T \operatorname{diag}(\boldsymbol{\ell})^{-2} \boldsymbol{v}_g
\end{aligned}$$

Using $\exp(a+b+c) = \exp(c) \cdot \exp(a+b)$, we have:

$$k_{\phi_{adj}}^{v2}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sigma^2 \exp(prod_{ij}) \exp\Big(- \tfrac{1}{2} \big[\boldsymbol{v}_\phi^T \operatorname{diag}(\boldsymbol{\ell})^{-2} \boldsymbol{v}_\phi + \boldsymbol{v}_g^T \operatorname{diag}(\boldsymbol{\ell})^{-2} \boldsymbol{v}_g \big] \Big)$$

Which can be written as:

$$\begin{aligned}
k_{\phi_{adj}}^{v2}(\boldsymbol{x}_i, \boldsymbol{x}_j) &= \sigma^2 \exp(prod_{ij}) \exp\Big(- \tfrac{1}{2} (\boldsymbol{t}_{ij}^{adj})^T \operatorname{diag}\Big(\begin{bmatrix} \boldsymbol{\ell} \\ \boldsymbol{\ell} \end{bmatrix} \Big)^{-2} \boldsymbol{t}_{ij}^{adj} \Big) \\
\boldsymbol{t}_{ij}^{adj} &= \begin{bmatrix} \boldsymbol{v}_\phi \\ \boldsymbol{v}_g \end{bmatrix} = \begin{bmatrix} \phi^{sim}(\boldsymbol{x}_i) - \phi^{sim}(\boldsymbol{x}_j) \\ \bar{g}(\boldsymbol{x}_i) - \bar{g}(\boldsymbol{x}_j) \end{bmatrix}
\end{aligned}$$

Compare this to $k_{\phi_{adj}}$ from Equation 4:

$$k_{\phi_{adj}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sigma_k^2 \exp\Big(- \tfrac{1}{2} (\boldsymbol{t}_{ij}^{adj})^T \operatorname{diag}\Big(\begin{bmatrix} \boldsymbol{\ell_1} \\ \boldsymbol{\ell_2} \end{bmatrix} \Big)^{-2} \boldsymbol{t}_{ij}^{adj} \Big) \tag{5}$$

Now we see that $k_{\phi_{adj}}^{v2}$ and $k_{\phi_{adj}}$ have a similar form. Hyperparameters $\boldsymbol{\ell_1}, \boldsymbol{\ell_2}$ provide flexibility in $k_{\phi_{adj}}$ as compared to having only vector $\boldsymbol{\ell}$ in $k_{\phi_{adj}}^{v2}$. They can be adjusted manually or with Automatic Relevance Determination. For $k_{\phi_{adj}}^{v2}$, the role of signal variance is captured by $\sigma^2 \exp(-prod_{ij})$. This makes the kernel nonstationary in the transformed ϕ space. Since $k_{\phi_{adj}}$ is already non-stationary in $\boldsymbol{x}$, it is unclear whether non-stationarity of $k_{\phi_{adj}}^{v2}$ in the transformed ϕ space has any advantages.

The above discussion shows that $k_{\phi_{adj}}$ proposed in [24] is motivated both intuitively and mathematically. It aims to use a transform that accounts for the hardware mismatch, without adding extra non-stationarity in the transformed space. While the above analysis shows the connection between $k_{\phi_{adj}}$ and $k_{\phi_{adj}}^{v2}$, a more systematic empirical analysis would be beneficial as part of future work.

4 Robots, Simulators and Controllers Used

In this section we give a concise description of the robots, controllers and simulators used in experiments with BO for bipedal locomotion. Our approach is applicable to a wide range of bipedal robots and controllers, including state-of-the-art controllers [10]. We work with two different types of controllers – a reactively stepping controller and a human-inspired neuromuscular controller (NMC). The reactively stepping controller is model-based: it uses inverse-dynamics models of the robot to compute desired motor torques. In contrast, NMC is model-free: it computes desired torques using hand-designed policies, created with biped locomotion dynamics in mind. These controllers exemplify two different and widely used ways of controlling bipedal robots. In addition to this, we show results on two different robot morphologies – a parallel bipedal robot ATRIAS, and a serial 7-link biped model. Our hardware experiments are conducted on ATRIAS; the 7-link biped is only used in simulation. Our success on both robots shows that the approaches developed in this paper are widely applicable to a range of bipedal robots and controllers.

4.1 ATRIAS Robot

Our hardware platform is an ATRIAS robot (Figure 1). ATRIAS is a parallel bipedal robot, weighing $\approx 64kg$. The legs are 4-segment linkages actuated by 2 Series Elastic Actuators (SEAs) in the sagittal plane and a DC motor in the lateral plane. Details can be found in [14]. In this work we focus on planar motion around a boom. ATRIAS is a highly dynamic system due to its point feet, with static stability only in double stance on the boom.

4.2 Planar 7-link Biped

The second robot used in our experiments is a 7-link biped [32]. It has a trunk and segmented legs with ankles, with actuators in the hip, knees and ankles. The inertial properties of its links are similar to an average human [37]. This 7-link model is a canonical simulator for testing bipedal walking algorithms, for example in Song and Geyer [29]. It is a simplified two-dimensional simulator for a large range of humanoid robots, like Atlas [10]. We use this simulator to study the generalizability of our proposed approaches to systems different from ATRIAS.

4.3 Feedback Based Reactive Stepping Policy

We design a parametric controller for controlling the CoM height, torso angle and the swing leg by commanding desired ground reaction forces and swing foot landing location.

$$\begin{aligned}
F_x &= K_{pt}(\theta_{des} - \theta) - K_{dt}\dot{\theta} \\
F_z &= K_{pz}(z_{des} - z) - K_{dz}\dot{z} \\
x_p &= k(v - v_{tgt}) + Cd + 0.5vT
\end{aligned}$$

Here, F_x is the desired horizontal ground reaction force (GRF), K_{pt} and K_{dt} are the proportional and derivative feedback gains on the torso angle θ and velocity $\dot{\theta}$. F_z is the desired vertical GRF, K_{pz} and K_{dz} are the proportional and derivative gains on the CoM height z and vertical velocity $\dot{z}$. z_{des} and θ_{des} are the desired CoM height and torso lean. x_p is the desired foot landing location for the end of swing; v is the horizontal CoM velocity, k is the feedback gain that regulates v towards the target velocity v_{tgt}. C is a constant and d is the distance between the stance leg and the CoM; T is the swing time.

The desired GRFs are sent to ATRIAS inverse dynamics model that generates desired motor torques τ_f, τ_b. Details can be found in [24]. This controller assumes no double-support, and the swing leg takes off as soon as stance is detected. This leads to a highly dynamic gait, posing a challenging optimization problem.

To investigate the effects of increasing dimensionality on our optimization, we construct two controllers with different number of free parameters:

- *5-dimensional controller* : optimizing 5 parameters $[K_{pt}, K_{dt}, k, C, T]$ (z_{des}, θ_{des} and the feedback on z are hand tuned)
- *9-dimensional controller* : optimizing all 9 parameters of the high-level policy $[K_{pt}, K_{dt}, \theta_{des}, K_{pz}, K_{dz}, z_{des}, k, C, T]$

4.4 16-dimensional Neuromuscular Controller

We use neuromuscular model policies, as introduced in [12], as our controller for the 7-link planar human-like biped model. These policies use approximate models of muscle dynamics and human-inspired reflex pathways to generate joint torques, producing gaits that are similar to human walking.

Each leg is actuated by 7 muscles, which together produce torques about the hip, knee and ankle. Most of the muscle reflexes are length or force feedbacks on the muscle state aimed at generating a compliant leg, keeping knee from hyperextending and maintaining torso orientation in stance. The swing control has three main components – target leg angle, leg clearance and hip control due to reaction torques. Together with the stance control, this leads to a total of 16 controller parameters, described in details in [1].

Though originally developed for explaining human neural control pathways, this controller has recently been applied to prosthetics and bipeds, for example Thatte and Geyer [32] and Van der Noot et al. [34]. This controller is capable of generating a variety of locomotion behaviours for a humanoid model – walking on rough ground, turning, running, and walking upstairs, making it a very versatile controller [29]. This is a model-free controller as compared to the reactive-stepping controller, which was model-based.

4.5 50-dimensional Virtual Neuromuscular Controller

Another model-free controller we use on ATRIAS is a modified version of [3]. VNMC maps a neuromuscular model, similar to the one described in Section 4.4 to the ATRIAS robot's topology and emulates it to generate desired motor torques. We adapt VNMC by removing some biological components while preserving its basic functionalities. The final version of the controller consists of 50 parameters including low-level control parameters, such as feedback gains, as well as high level parameters, such as desired step length and desired torso lean. Details can be found in [24]. When optimized using CMA-ES, it can control ATRIAS to walk on rough terrains with height changes of ± 20 cm in planar simulation [3].

4.6 Increasingly Inaccurate Simulators

To compare the performance of different methods that can be used to transfer information from simulation to hardware, we create a series of increasingly approximate simulators. These simulators emulate increasing mismatch between simulation and hardware and its effect on the information transfer. In this setting, the high-fidelity ATRIAS simulator [20], which was used in all the previous simulation experiments becomes the *"hardware"*. Next we make dynamics approximations to the original simulator, which are used commonly in simulators to decrease fidelity and increase simulation speed. For example, the complex dynamics of harmonic drives are approximated as a torque multiplication, and the boom is removed from the simulation, leading to a two-dimensional simulator. These approximate simulators now become the *"simulators"*. As the approximations in these simulators are increased, we expect the performance of methods that utilize simulation for optimization on hardware to deteriorate.

The details of the approximate simulators are described in the two paragraphs below:

1. Simulation with simplified gear dynamics : We replace the original high-fidelity gear model with a commonly used approximation for geared systems – multiplying the rotor torque by the gear ratio. This reduces the simulation time to about a third of the original simulator, but leads to an approximate gear dynamics model.

2. Simulation with no boom and simplified gear dynamics : The ATRIAS robot walks on a boom in our hardware experiments. In our second approximation, we remove the boom from the original simulator and constraint the motion of the robot to a 2-dimensional plane. This is a common approximation for two-dimensional robots. However, the boom leads to lateral forces on the robot, which have vertical force components that are not modelled anymore.

The advantage of such an arrangement is that we can test the effect of un-modelled and wrongly modelled dynamics on information transfer between simulation and hardware. Even in our high-fidelity original simulator, there are several un-modelled components of the actual hardware. For example, the non-rigidness of the robot parts, misaligned motors and relative play between joints. In our experiments, we find that the 50-dimensional VNMC is a sensitive controller, with little hope of directly transferring from simulation to hardware. Anticipating this, we can now test several methods of compensating for this mismatch using our increasingly approximate simulators.

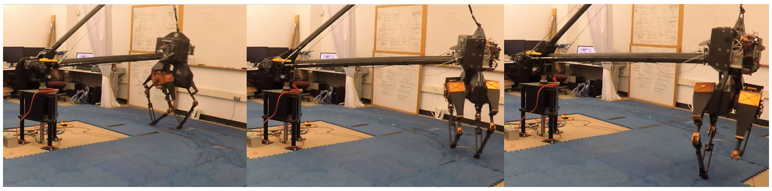


Figure 3: ATRIAS during BO with DoG-based kernel (video: `https://youtu.be/hpXNFREgaRA`)

5 Experiments

We will now present our experiments on optimizing controllers that are 5, 9, 16 and 50 dimensional. We split our experiments into three categories: hardware experiments on the ATRIAS robot, simulation experiments on the 7-link biped and experiments using simulators with different levels of mismatch. We demonstrate that our proposed approach is able to generalize to different controllers and robot structures and is also robust to simulation inaccuracies. The sections below present experimental results, while Appendix A gives further details on data collection, BO implementation and kernel generation.

5.1 Hardware Experiments on the ATRIAS Robot

In this section we describe experiments conducted on the ATRIAS robot (hardware from Section 4.1). These experiments were conducted around a boom. The cost function used in our experiments is a slight modification of the cost used in [29]:

$$cost = \begin{cases} 100 - x_{fall}, \text{if fall} \\ ||v_{avg} - v_{tgt}||, \text{if walk} \end{cases} \tag{6}$$

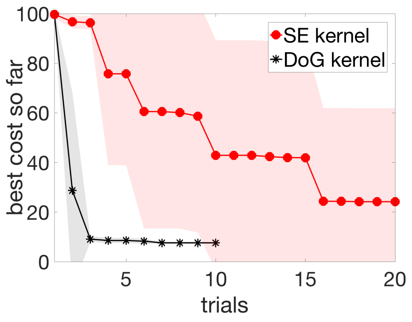
where x_{fall} is distance covered before falling, v_{avg} is average speed per step and v_{tgt} contains target velocity profile, which can be variable. This cost function heavily penalizes falls, and encourages walking controllers to track target velocity.

We do multiple runs of each algorithm on the robot. Each run typically consists of 10 experiments on the robot. All BO runs start from scratch, with an uninformed GP prior. At the end of the run, the GP posterior has 10 data points, depending on the experiment. Each robot trial is designed to be between $30s$ to $60s$ long and the robot is reset to its "home" position between trials.

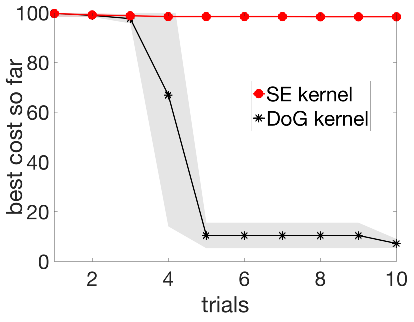
We will present two sets of hardware experiments in the following subsections. First we describe experiments with the DoG-based kernel on the 5 and 9 dimensional controllers (first reported in Rai et al. [24]). The second set describes a new set of experiments for optimizing a 9-dimensional controller using a Neural Network based kernel on hardware.

5.1.1 Experiments with a 5-dimensional controller and DoG kernel

In our first set of experiments on the robot, we investigated optimizing the 5-dimensional controller from Section 4.3. For these experiments we picked a



(a) BO for 5D controller. BO with SE finds walking points in 4/5 runs within 20 trials. BO with DoG-based kernel finds walking points in 5/5 runs within 3 trials.



(b) BO for 9D controller. BO with SE doesn't find walking points in 3 runs. BO with DoG-based kernel finds walking points in 3/3 runs within 5 trials.

Figure 4: BO for 5D and 9D controller on ATRIAS robot hardware. Plots show mean best cost so far. Shaded region shows $\pm$ one standard deviation. Plots from Rai et al. [24]

challenging variable target speed profile:

$0.4m/s$ (15 steps) - $1.0m/s$ (15 steps) - $0.2m/s$ (15 steps) - $0m/s$ (5 steps).

The controller was stopped after the robot took 50 steps. To evaluate the difficulty of this setting, we sampled 100 random points on hardware. 10% of these randomly sampled controllers could walk. In contrast, in simulation the success rate of random sampling was ≈27.5%. This indicates that the simulation was easier, which could be potentially detrimental to algorithms that rely heavily on simulation, because a large portion of controllers that walk in simulation fall on hardware. Nevertheless, using a DoG-based kernel offered significant improvements over a standard SE kernel (Figure 4a).

We conducted 5 runs of BO with DoG-based kernel and 5 runs of BO with SE, 10 trials for DoG-based kernel per run, and 20 for SE kernel. In total, this led to 150 experiments on the robot (excluding the 100 random samples, which were not used during BO). BO with DoG-based kernel found walking points in 100% of runs within 3 trials. In comparison, BO with SE found walking points in 10 trials in 60% runs, and in 80% runs in 20 trials (Figure 4a). Although BO could find walking controllers as early as the second trial (with no prior hardware information), it is worth noting that the optimization did not converge after only a few trials. Sampling more could possibly lead to better walking controllers, but our goal was to find the best controller with a budget of only 10 to 20 trials.

5.1.2 Experiments with a 9-dimensional controller and DoG kernel

Our next set of experiments optimized the 9-dimensional controller from Section 4.3. First, we sampled 100 random points for the variable speed profile described above, but this led to no walking points. To ensure that we have a reasonable baseline we decided to simplify the speed profile for this setting: $0.4m/s$ for 30 steps. We evaluated 100 random points on hardware, and 3 walked for the easier speed profile. In comparison, the success rate in simulation was 8% for the tougher variable-speed profile, implying an even greater mismatch between hardware and simulation than the 5-dimensional controller. For experiments in Sections 5.1.1 and 5.1.2 the inertial measurement unit (IMU) of the robot was broken, and we replaced its functionality with external boom sensors. These were lower resolution than the IMU, leading to noisier readings and larger time delays. As a result, the system did not have a good estimation of vertical height of the CoM, leading to poor control authority. However, the IMU on ATRIAS is a very expensive fiber-optic IMU that is not commonly used on humanoid robots, and most robots use simple state estimation methods. So, this is a common setting for humanoid robots, even if it presents a challenge for the optimization methods.

We conducted 3 runs of BO with DoG-based kernel and BO with SE, 10 trials for DoG-based kernel per run, and 10 for SE. In total, this led to 60 experiments on the hardware (excluding the random samples, which were not used for BO). BO with DoG-based kernel found walking points in 5 trials in 3/3 runs. BO with SE did not find any walking points in 10 trials in all 3 runs. These results are shown in Figure 4b.

Based on these results, we concluded that BO with DoG-based kernel was indeed able to extract useful information from simulation and speed up learning on hardware, even when there was mismatch between simulation and hardware.

5.1.3 Experiments with a 9-dimensional controller and NN kernel

In the next set of experiments, we evaluated performance of the NN-based kernel described in Section 3.1.2. We optimize the 9-dimensional controller from Section 4.3.

The target of hardware experiments was to walk for 30 steps at $0.4m/s$, similar to Section 5.1.2. For these experiments the IMU was repaired, leading to better state estimation on the robot. For a fair comparison, we re-ran experiments with the baseline for this setting and the baseline performed slightly better than the baseline of earlier experiments (because of improved sensing).

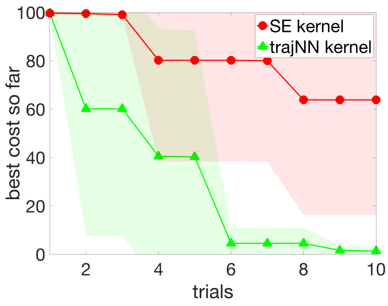
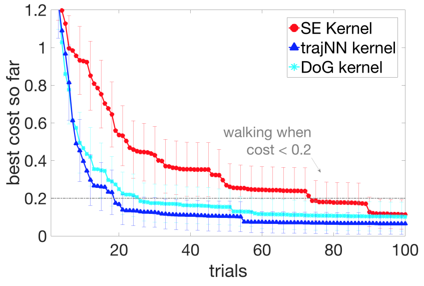
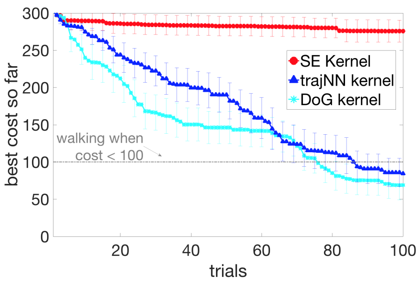


Figure 5: BO for 9D controller on ATRIAS robot hardware. Shaded region shows ± one standard dev.

Figure 5 shows comparison of BO with NN-based kernel and SE kernels. We conducted 5 runs of both algorithms with 10 trials in each run, leading to a total of 100 robot trials. BO with the NN-based kernel found walking points in all 5 runs within 6 trials, while BO with SE kernel only found walking points in 2 of 5 runs in 10 trials. Hence, even without explicit hand-designed domain knowledge, like the DoG-based kernel, the NN-based kernel is able to extract useful information from simulation and successfully guide hardware experiments.



(a) Using smooth cost from Equation 7



(b) Using non-smooth cost from Equation 8

Figure 6: BO for the Neuromuscular controller. *trajNN* and *DoG* kernels were constructed with undisturbed model on flat ground. BO is run with mass/inertia disturbances on different rough ground profiles. Means of 50 runs, 95% CIs. Plots from Antonova et al. [2]

5.2 Simulation Experiments on a 7-link Biped

This section describes simulation experiments with a 16-dimensional Neuromuscular controller (Section 4.4) on a 7-link biped model. These experiments, from Antonova et al. [2], highlight the cost-agnostic nature of our approach by optimizing two very different costs.

Figure 6 shows BO with DoG-based kernel, NN-based kernel and SE kernel for two different costs from prior literature. The first cost promotes walking further and longer before falling, while penalizing deviations from the target speed [1]:

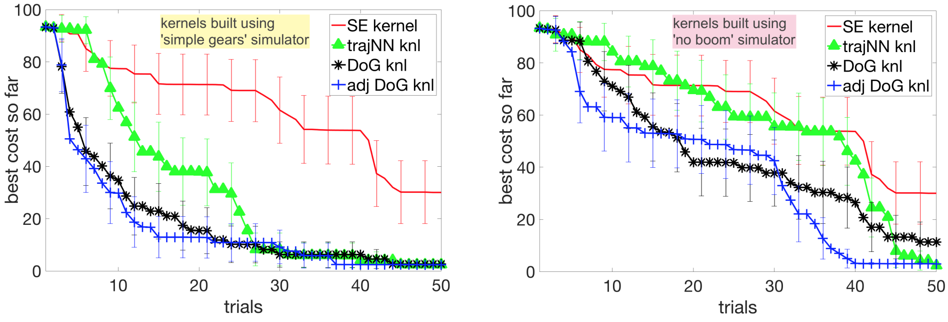
$$cost_{smooth} = 1/(1+t) + 0.3/(1+d) + 0.01(s - s_{tgt}), \tag{7}$$

where t is seconds walked, d is the final CoM position, s is speed and s_{tgt} is the desired walking speed ($1.3m/s$ in our case). The second cost function is similar to the cost used in Section 5. It penalizes falls explicitly, and encourages walking at desired speed and with lower cost of transport:

$$cost_{non\text{-}smooth} = \begin{cases} 300 - x_{fall}, \text{if fall} \\ 100||v_{avg} - v_{tgt}|| + c_{tr}, \text{if walk} \end{cases} \tag{8}$$

where x_{fall} is the distance covered before falling, v_{avg} is the average speed of walking, v_{tgt} is the target velocity, and c_{tr} captures the cost of transport. The changed constants is to account for a longer simulation time. Figure 6a shows that the NN-based kernel and the DoG-based kernel offer a significant improvement over BO with the SE kernel in sample efficiency when using the $cost_{smooth}$, with more than 90% of runs achieving walking after 25 trials. BO with the SE kernel takes 90 trials to get 90% success rate. Figure 6b shows that similar performance by the two proposed approaches is observed on the non-smooth cost. With the NN-based kernel, 70% of the runs find walking solutions after 100 trials, similar to the DoG-based kernel. However, optimizing non-smooth cost is very challenging for BO with the SE kernel: a walking solution is found only in 1 out of 50 runs after 100 trials.

We attribute the difference in performance of the SE kernel on the two costs to the nature of the costs. If a point walks some distance d, Equation 7 reduces in



(a) Informed kernels generated using simulator with simplified gear dynamics

(b) Informed kernels generated using simplified gear dynamics without a boom

Figure 7: BO is run on the original simulator. Informed kernels perform well despite significant mismatch, when kernels are generated using simulator with simplified gear dynamics (left). In the case of severe mismatch, when the boom model is also removed, informed kernels still improve over baseline SE (right). Plots show best cost for mean over 50 runs for each algorithm, 95% CIs.

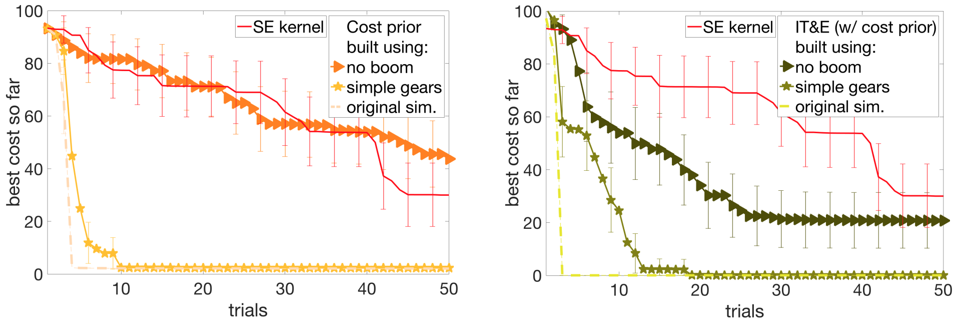
terms of $\frac{1}{d}$ and Equation 8 reduces by $-d$. A sharper fall in the first cost encourages BO to exploit around points that walk some distance, quickly finding points that walk forever. BO with the second cost continues to explore, as the signal is too weak. However the success of both NN-based and DoG-based kernels on both costs shows that the same kernel can indeed be used for optimizing multiple costs robustly, without any further tuning needed.

5.3 Experiments with Increasing Simulation-Hardware Mismatch

In this section, we describe our experiments with increasing simulation-hardware mismatch and its effect on approaches that use information from simulation during hardware optimization. The quality of information transfer between simulation and hardware depends not only on the mismatch between the two, but also on the controller used. For a robust controller, small dynamics errors would not cause a significant deterioration in performance, while for a sensitive controller this might be much more detrimental.

In the rest of this section, we provide experimental analyses of settings with increasing simulated mismatch and their effect on optimization of the 50-dimensional VNMC from Section 4.5. We compare several approaches that improve sample-efficiency of BO and investigate if the improvement they offer is robust to mismatch between the simulated setting used for constructing kernel/prior and the setting on which BO is run.

First, we examine the performance of our proposed approaches with informed kernels: k_{DoG}, k_{trajNN} and $k_{DoG_{adj}}$. Figure 7a shows the case when informed kernels are generated using the simulator with simplified gear dynamics while BO is run on



(a) BO with cost prior: straightforward approach useful for low-to-medium mismatch; but no improvement if mismatch is severe

(b) Performance of IT&E algorithm (our implementation of Cully et al. [7], adapted to bipedal locomotion)

Figure 8: BO using prior-based approaches. Mean of 50 runs for each algorithm, 95% CIs.

the original simulator. After 50 trials, all runs with informed kernels find walking solutions, while for SE only 70% have walking solutions.

Next, Figure 7b shows performance of k_{DoG}, k_{trajNN} and $k_{DoG_{adj}}$ when the kernels are constructed using a simulator with simplified dynamics and without a the boom. In this case the mismatch with the original simulator is larger than before and we see the advantage of using adjustment for DoG-based kernel: $k_{DoG_{adj}}$ finds walking points in all runs after 35 trials. k_{trajNN} also achieves this, but after 50 trials. k_{DoG} finds walking points in $\approx$90% of the runs after 50 trials. The performance of SE stays the same, as it uses no prior information from any simulator.

This illustrates that while the original DoG-based kernel can recover from slight simulation-hardware mismatch, the adjusted DoG-kernel is required for higher mismatch. k_{trajNN} seems to recover from the mismatch, but might benefit from an adjusted version. We leave this to future work.

5.3.1 Comparisons of Prior-based and Kernel-based Approaches

In this section, we classify approaches that use simulation information in hardware optimization as prior-based or kernel-based. Prior-based approaches use costs from the simulation in the prior of the GP used in the BO. This can help BO a lot if the costs are similar between simulation and hardware, and the cost function is fixed. However, in the presence of large mismatch, controllers that perform well in simulation might fail on hardware. A prior-based method can be biased towards sampling promising points from simulation, resulting in worse performance than uninformed BO. Kernel-based approaches consist of methods that incorporate information from simulation into the kernel of the GP. These can be less sample-efficient as compared to prior-based method, but not as likely to be biased towards unpromising regions in the presence of mismatch. They also easily generalize to multiple costs, so that there is no need to re-run simulations for data collection if the cost is changed. This is important because a lot of these approaches can take several days of computation to generate a cost prior or informed kernel. For example, [7] report taking 2 weeks on a 16-core computer to generate their map.

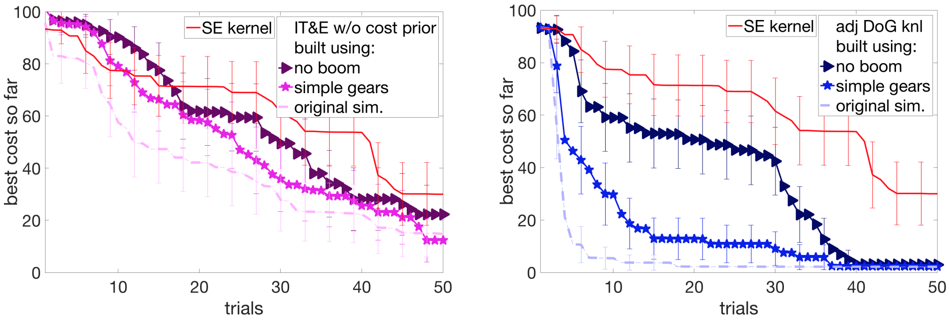
It is possible to also combine both prior-based and kernel-based methods, as in [7]. We classify these as 'prior-based' methods, since in our experiments prior outweighs the kernel effects for such cases. In our comparison with [7], we implement a version with and without the prior points. We do not add a cost prior to BO using DoG-based kernel, as this would limit us to a particular cost and high-fidelity simulators, and both of these can be major obstacles in real robot experiments.

Figure 8a shows the performance when using simulation cost in the prior during BO. BO with a cost prior created using the original version of the simulator illustrates what would happen in the best case scenario. When the simulator with simplified gear dynamics is used for constructing the prior, we observe significant improvements over uninformed BO prior. However, when the prior is constructed from simplified gear dynamics and no boom setting, the approach performs slightly worse than uninformed BO. This shows that while an informed prior can be very helpful when created from a simulator close to hardware, it can hurt performance if simulator is significantly different from hardware.

Next, we discuss experiments with our implementation of Intelligent Trial and Error (IT&E) algorithm from Cully et al. [7]. This algorithm combines adding a cost prior from simulated evaluations with adding simulation information into the kernel. IT&E defines a behavior metric and tabulates best performing points from simulation on their corresponding behavior score. The behavior metric used in our experiments is duty-factor of each leg, which can go from 0 to 1.0. We discretize the duty factor into 21 cells of 0.05 increments, leading to a 21×21 grid. We collect the 5 highest performing controllers for each square in the behavior grid, creating a $21 \times 21 \times 5$ grid. Next, we generate 50 random combinations of a 21×21 grid, selecting 1 out of the 5 best controllers per grid cell. Care was taken to ensure that all 5 controllers had comparable costs in the simulator used for creating the map. Cost of each selected controller is added to the prior and BO is performed in the behavior space, like in [7].

Figure 8b shows BO with IT&E constructed using different versions of the simulator. IT&E constructed using simplified gear dynamics simulator is slightly less sample-efficient than the straightforward 'cost prior' approach. When constructed with the simulator with no boom, IT&E is able to improve over uninformed BO. However, it only finds walking points in 77% of the runs in 50 trials in this case, as some of the generated maps contained no controllers that could walk on the 'hardware'. This is a shortcoming of the IT&E algorithm, as it eliminates a very large part of the search space and if the pre-selected space does not contain a walking point, no walking controllers can be sampled with BO. This problem could possibly be avoided by using a finer grid, or a different behavior metric. However tuning such hyper-parameters can turn out to be expensive, in computation and hardware experiment time.

To separate the effects of using simulation information in prior mean vs kernel,



(a) BO using our implementation of IT&E without cost prior (from Cully et al. [7])

(b) BO using $k_{DoG_{adj}}$ constructed from simulators with various levels of mismatch

Figure 9: BO with kernel-based approaches. Mean of 50 runs for each algorithm, 95% CIs

we evaluated a kernel-only version of IT&E algorithm (Figure 9a). It shows that the cost prior is crucial for the success of IT&E and performance deteriorates without it. Hence, it is not practical to use IT&E on a cost different than what it was generated for. Nonetheless, Figure 7 showed that BO with adjusted DoG kernel is able to handle both moderate and severe mismatch with kernel-only information, collected in Figure 9b.

Summarizing this section, we created two simulators with increasing modelling approximations, and studied the effect of using these to aid optimization on the original simulator. We found that while methods that use cost in the prior of BO can be very sample-efficient in low mismatch, their performance worsens as mismatch increases. IT&E, introduced in [7], uses simulation information in both prior mean and kernel, and is very sample-efficient in cases of low mismatch. Even with high mismatch, it performed better than just prior-based BO but doesn't find walking controllers reliably. In comparison, adjusted DoG-based kernel performed well in all the tested scenarios, and can reliably improve sample-efficiency of BO even when the mismatch between simulation and hardware is high.

6 Conclusion

In this paper, we presented and analyzed in detail our work from Antonova et al. [1], Antonova et al. [2] and Rai et al. [24]. These works introduce domain-specific feature transforms that can be used to optimize locomotion controllers on hardware efficiently. The feature transforms project the original controller space into a space where BO can discover promising regions quickly. We described a transform for bipedal locomotion designed with the knowledge of human walking and a neural network based transform that uses more general information from simulated trajectories. Our experiments demonstrate success at optimizing controllers on the ATRIAS robot. Further simulation-based experiments also indicate potential for other bipedal robots. For optimizing sensitive high-dimensional controllers, we proposed an approach to adjust simulation-based kernel using data seen on hardware. To study the performance of this, as well as compare our approach to other methods, we created a series of increasingly approximate simulators. Our experiments show that while several methods from prior literature can perform well with low simulation-hardware mismatch (sometimes even better than our proposed approach), they suffer when this mismatch increases. In such cases, our proposed kernels with hardware adjustment can yield reliable performance across different costs, simulators and robots.

Acknowledgments

This research was supported in part by National Science Foundation grant IIS-1563807, the Max-Planck-Society, & the Knut and Alice Wallenberg Foundation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding organizations.

Appendix A: Implementation Details

In this appendix we provide a summary of data collection and implementation details. Our implementation of BO was based on the framework in Gardner et al. [11]. We used Expected Improvement (EI) acquisition function [22]. We also experimented with Upper Confidence Bound (UCB) [30], but found that performance was not sensitive to the choice of acquisition function. Hyper-parameters for BO were initialized to default values: 0 for mean offset, 1.0 for kernel length scales and signal variance, 0.1 for σ_n (noise parameter). Hyperparameters were optimized using the marginal likelihood ([26], Section V-A). For all algorithms, we started optimizing hyperparameters after a low-cost controller was found (to save compute resources and avoid premature hyperparameter optimization). The search space boundaries for controller parameters was designed with physical constraints of the ATRIAS robot in mind.

k_{trajNN} discussed in Section 3.1.2 was constructed by training a fully connected neural network (NN) with 3 hidden layers, using L1 loss to reconstruct features encoded by trajectory summaries. For experiments with 16D controller in Section 5.2, for example, the hidden layers contained 512, 128, 32 units; NN was trained on 100K simulated examples to reconstruct 8D trajectory summaries (see next-to-last row of Table B.2). We experimented with various hidden layer sizes for 9D and 50D controllers, but did not find the overall BO performance to be sensitive to size or other training parameters, like NN learning rate. This was likely because NN used to construct an informed kernel only needs to approximately learn to separate well-performing parts from failing parts of the control parameter space. This is the benefit of our choice not to put the output of NN directly into a GP posterior.

To create cost prior for experiments in Section 5.3 we collected 50,000 evaluations of 30s trials for a range of controller parameters. Then we conducted 50 runs, using

Kernel type	Controller dim	# Sim points	Sim duration	Kernel dim	Features in kernel
k_{DoG}	5	20K	3.5s	1	$score_{DoG}$
	9	100K	5s	1	$score_{DoG}$
	50	200K	5s	1	$score_{DoG}$
k_{trajNN}	9	100K	5s	4	t_{walk}, x_{end}, θ_{avg}, $v_{x,avg}$
	16	100K	5s	8	t_{walk}, x_{end}, θ_{end}, $v_{x,end}$ c_{τ}, y_{end}, $v_{y,end}$, $\dot{\theta}_{end}$
	50	200K	5s	13	t_{walk}, x_{end}, c_{τ}, $\boldsymbol{traj}_x$, $\boldsymbol{traj}_\theta$

Table B.2: Simulation data collection details. $score_{DoG}$ was described in Section 3.1.1. For k_{trajNN}: t_{walk} is time walked in simulation before falling, x_{end} and y_{end} are the x and y positions of Center of Mass (CoM) at the end of the short simulation, θ is the torso angle, $\dot{\theta}$ is the torso velocity, v is the CoM speed (v_x is the horizontal and v_y is the vertical component), c_{τ} is the squared sum of torques applied; $\boldsymbol{traj}_x$, $\boldsymbol{traj}_\theta$ denote vectors with mean CoM and θ measurements every second.

random subsets of 35,000 evaluations to construct the prior. The numbers were chosen such that this approach used similar amount of computation as our kernel-based approaches. To accommodate GP prior with a large number of points we used a sparse GP construction provided by [25].

References

[1] Rika Antonova, Akshara Rai, and Christopher G Atkeson. Sample efficient optimization for learning controllers for bipedal locomotion. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 22–28. IEEE, 2016.

[2] Rika Antonova, Akshara Rai, and Christopher G Atkeson. Deep Kernels for Optimizing Locomotion Controllers. In *Conference on Robot Learning (CoRL), PMLR 78*, pages 47–56, 2017.

[3] Zachary Batts, Seungmoon Song, and Hartmut Geyer. Toward a virtual neuromuscular control for robust walking in bipedal robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6318–6323. IEEE, 2015.

[4] Eric Brochu, Vlad M Cora, and Nando De Freitas. A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning. *arXiv preprint arXiv:1012.2599*, 2010.

[5] Roberto Calandra, Jan Peters, Carl Edward Rasmussen, and Marc Peter Deisenroth. Manifold Gaussian processes for regression. In *International Joint Conference on Neural Networks (IJCNN)*, pages 3338–3345. IEEE, 2016.

[6] Roberto Calandra, André Seyfarth, Jan Peters, and Marc Peter Deisenroth. Bayesian Optimization for Learning Gaits Under Uncertainty. *Annals of Mathematics and Artificial Intelligence*, 76(1-2):5–23, 2016.

[7] Antoine Cully, Jeff Clune, Danesh Tarapore, and Jean-Baptiste Mouret. Robots that can adapt like animals. *Nature*, 521(7553):503–507, 2015.

[8] Marc Deisenroth and Carl E Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *International Conference on Machine Learning (ICML)*, pages 465–472, 2011.

[9] Peter Englert and Marc Toussaint. Combined Optimization and Reinforcement Learning for Manipulation Skills. In *Robotics: Science and Systems*, 2016.

[10] Siyuan Feng, Eric Whitman, X Xinjilefu, and Christopher G Atkeson. Optimization-based Full Body Control for the DARPA Robotics Challenge. *Journal of Field Robotics*, 32(2):293–312, 2015.

[11] Jacob R Gardner, Matt J Kusner, Zhixiang Eddie Xu, Kilian Q Weinberger, and John Cunningham. Bayesian Optimization with Inequality Constraints. In *International Conference on Machine Learning (ICML)*, pages 937–945, 2014.

[12] Hartmut Geyer and Hugh Herr. A Muscle-reflex Model that Encodes Principles of Legged Mechanics Produces Human Walking Dynamics and Muscle Activities. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 18(3):263–273, 2010.

[13] Nikolaus Hansen. The CMA evolution strategy: a comparing review. In *Towards a new evolutionary computation*, pages 75–102. Springer, 2006.

[14] Christian Hubicki, Jesse Grimes, Mikhail Jones, Daniel Renjewski, Alexander Spröwitz, Andy Abate, and Jonathan Hurst. ATRIAS: Design and validation of a tether-free 3D-capable spring-mass bipedal robot. *The International Journal of Robotics Research (IJRR)*, 35(12):1497–1521, 2016.

[15] Verne T Inman, Howard D Eberhart, et al. The major determinants in normal and pathological gait. *Journal of Bone and Joint Surgery (JBJS)*, 35(3):543–558, 1953.

[16] Kirthevasan Kandasamy, Gautam Dasarathy, Jeff Schneider, and Barnabás Póczos. Multi-fidelity Bayesian Optimisation with Continuous Approximations. In *International Conference on Machine Learning (ICML)*, pages 1799–1808, 2017.

[17] Scott Kuindersma, Robin Deits, Maurice Fallon, Andrés Valenzuela, Hongkai Dai, Frank Permenter, Twan Koolen, Pat Marion, and Russ Tedrake. Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot. *Autonomous Robots*, 40(3):429–455, 2016.

[18] Daniel J Lizotte, Tao Wang, Michael H Bowling, and Dale Schuurmans. Automatic Gait Optimization with Gaussian Process Regression. In *International Joint Conference on Artificial Intelligence (IJCAI)*, volume 7, pages 944–949, 2007.

[19] Alonso Marco, Felix Berkenkamp, Philipp Hennig, Angela P Schoellig, Andreas Krause, Stefan Schaal, and Sebastian Trimpe. Virtual vs. real: Trading off simulations and physical experiments in reinforcement learning with Bayesian optimization. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1557–1563. IEEE, 2017.

[20] William C Martin, Albert Wu, and Hartmut Geyer. Robust spring mass model running for a physical bipedal robot. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 6307–6312. IEEE, 2015.

[21] Ruben Martinez-Cantin. Funneled Bayesian optimization for design, tuning and control of autonomous systems. *IEEE transactions on cybernetics*, (99):1–12, 2018.

[22] J Mockus, V Tiesis, and A Zilinskas. Toward Global Optimization, Volume 2, Chapter: Bayesian Methods for Seeking the Extremum. 1978.

[23] Xue Bin Peng, Glen Berseth, and Michiel van de Panne. Terrain-adaptive locomotion skills using deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 35 (4):81, 2016.

[24] Akshara Rai, Rika Antonova, Seungmoon Song, William Martin, Hartmut Geyer, and Christopher Atkeson. Bayesian optimization using domain knowledge on the ATRIAS biped. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1771–1778. IEEE, 2018.

[25] Carl Edward Rasmussen and Hannes Nickisch. Gaussian processes for machine learning (gpml) toolbox. *J. Mach. Learn. Res.*, 11:3011–3015, December 2010. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=1756006.1953029`.

[26] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando de Freitas. Taking the Human Out of the Loop: A Review of Bayesian Optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

[27] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems (NIPS)*, pages 2951–2959, 2012.

[28] Jasper Snoek, Kevin Swersky, Rich Zemel, and Ryan Adams. Input warping for Bayesian optimization of non-stationary functions. In *International Conference on Machine Learning (ICML)*, pages 1674–1682, 2014.

[29] Seungmoon Song and Hartmut Geyer. A Neural Circuitry that Emphasizes Spinal Feedback Generates Diverse Behaviours of Human Locomotion. *The Journal of Physiology*, 593(16):3493–3511, 2015.

[30] Niranjan Srinivas, Andreas Krause, Sham Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: no regret and experimental design. In *International Conference on Machine Learning (ICML)*, pages 1015–1022. Omnipress, 2010.

[31] Matthew Tesch, Jeff Schneider, and Howie Choset. Using response surfaces and expected improvement to optimize snake robot gait parameters. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1069–1074. IEEE, 2011.

[32] Nitish Thatte and Hartmut Geyer. Toward Balance Recovery with Leg Prostheses Using Neuromuscular Model Control. *IEEE Transactions on Biomedical Engineering*, 63(5):904–913, 2016.

[33] Evangelos Theodorou, Jonas Buchli, and Stefan Schaal. A generalized path integral control approach to reinforcement learning. *Journal of Machine Learning Research (JMLR)*, 11(Nov):3137–3181, 2010.

[34] Nicolas Van der Noot, Luca Colasanto, Allan Barrea, Jesse van den Kieboom, Renaud Ronsse, and Auke J Ijspeert. Experimental validation of a bio-inspired controller for dynamic walking with a humanoid robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 393–400. IEEE, 2015.

[35] Aaron Wilson, Alan Fern, and Prasad Tadepalli. Using Trajectory Data to Improve Bayesian Optimization for Reinforcement Learning. *The Journal of Machine Learning Research (JMLR)*, 15(1):253–282, 2014.

[36] Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P Xing. Deep kernel learning. In *Artificial Intelligence and Statistics*, pages 370–378, 2016.

[37] DA Winter and HJ Yack. EMG profiles during normal human walking: stride-to-stride and inter-subject variability. *Electroencephalography and clinical neurophysiology*, 67 (5):402–411, 1987.

Deep Kernels for Optimizing Locomotion Controllers

Rika Antonova[1]
EECS, KTH, Stockholm, Sweden

Akshara Rai[1]   **Christopher G. Atkeson**
Robotics Institute, Carnegie Mellon University, USA

Abstract

Sample efficiency is important when optimizing parameters of locomotion controllers, since hardware experiments are time consuming and expensive. Bayesian Optimization, a sample-efficient optimization framework, has recently been widely applied to address this problem, but further improvements in sample efficiency are needed for practical applicability to real-world robots and high-dimensional controllers. To address this, prior work has proposed using domain expertise for constructing custom distance metrics for locomotion. In this work we show how to learn such a distance metric automatically. We use a neural network to learn an informed distance metric from data obtained in high-fidelity simulations. We conduct experiments on two different controllers and robot architectures. First, we demonstrate improvement in sample efficiency when optimizing a 5-dimensional controller on the ATRIAS robot hardware. We then conduct simulation experiments to optimize a 16-dimensional controller for a 7-link robot model and obtain significant improvements even when optimizing in perturbed environments. This demonstrates that our approach is able to enhance sample efficiency for two different controllers, hence is a fitting candidate for further experiments on hardware in the future.

Keywords: Bayesian optimization, Sim-to-Real transfer, Biped Locomotion

1 Introduction

[1]Both of these authors contributed equally.

In proceedings of the 1st Conference on Robot Learning (CoRL 2017), Mountain View, CA, US. Published by PMLR http://proceedings.mlr.press/v78/antonova17a License: CC-BY 4.0

NOTE: The JMLR2019 article included *trajNN* kernel from this work and reported new hardware experiments. This CoRL2017 paper shows cost landscape and cost-based *asymNN* kernel results.

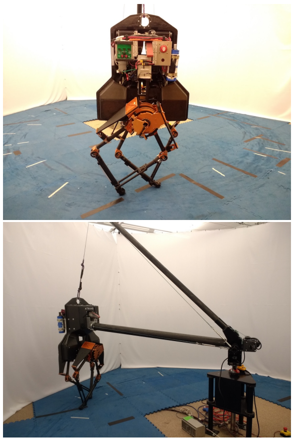


Figure 1: ATRIAS robot.

Bayesian Optimization (BO) is rapidly becoming a popular approach for optimizing controllers in robotics. It offers sample-efficient, black-box and gradient-free optimization, well suited for many problems in the field. Recently, some success has also been achieved when optimizing controllers directly on hardware [1, 2, 3]. Hence, this sample-efficient optimization framework has the potential to alleviate the need for manual tuning by experts, to a large extent. However, for high-dimensional controllers and challenging cost functions the performance of conventional BO often degrades. Without an informative prior, the number of data points required could be prohibitively expensive for hardware-only optimization. Hence, it seems ideal to exploit simulation to speed up learning, as proposed in [4] and [3]. These prior approaches, however, need extensive expert domain knowledge to define the problem-specific informed distance metric.

In this work we demonstrate how to construct an informed metric automatically, without relying heavily on domain experts. We propose to learn a distance metric with a neural network, utilizing data obtained from a high-fidelity simulator. This involves first running short simulations of a locomotion controller on a large grid of control parameters and recording the behavior of each set of parameters. The neural network then learns a mapping between input controller parameters and simulation output/behavior. We propose two ways of defining the target to be learned by the network. The first approach is based on the cost function that is to be optimized with BO on hardware, or a perturbed simulator. The second is cost-agnostic: learning to reconstruct a summary of robot trajectories obtained from simulation. This provides a useful re-parameterization: controller parameters that produce similar walking trajectory summaries are closer in this re-parameterized space.

In our first set of experiments we optimize a 5-dimensional controller on the ATRIAS robot hardware (Figure 1). We demonstrate that using cost-based kernel obtained with our approach outperforms using an uninformed kernel for BO. The setting we consider for ATRIAS experiments yields a proof-of-concept rather than a large-scale optimization problem. Nonetheless, we believe that its an important step towards optimizing locomotion policies for complex humanoid robots on hardware.

Prior Bayesian Optimization studies often used simpler robots. For example, [5] use snake robots, [3] use a hexapod, which often have statically stable gaits, or spend a significant amount of gait duration in a statically stable state. [1] use a smaller biped with a finite-state-machine controller, which is not widely used. In contrast, ATRIAS is a complex bipedal robot, which cannot be statically stable in single support because of point feet. Hence, it is likely to fall with unstable controllers. Moreover, our control framework is in line with most state-of-the-art robot controllers [6], [7]. Hence, results on our testbed can be transferred to other systems.

Our second set of experiments is on the Neuromuscular model [8]. We optimize a 16-dimensional controller for a 7-link robot model in simulation. Our approach of reconstructing trajectory summaries again yields a significant improvement over using uninformed kernels for BO. This is the case for both a smooth and a challenging non-smooth cost suggested in prior literature [9]. Hence the proposed approach offers a promising way to construct cost-agnostic kernels for BO automatically.

2 Background

2.1 Optimizing Bipedal Locomotion Controllers

Approaches to optimizing locomotion controllers range from manual tuning to fully automatic optimization. For complex controllers fully manual tuning is sometimes infeasible or excessively time consuming. In such cases, approaches like CMA-ES have been used to find points yielding good performance in simulation first [9]. A domain expert could then use such points as starting points to later manually adjust the parameters such that they are effective on hardware.

Recently there has been significant interest in developing methods for automatic parameter optimization. Bayesian Optimization has been suggested as one of the promising approaches due its sample efficiency. However, it still can take 30-40 samples to optimize a 4 dimensional controller [1]. To enhance kernel flexibility [10] suggests supervised learning of a feature transform during regression. However, this approach does not directly support incorporating a very large amount of data from simulation. Even if it is extended to pre-train on simulated data, it is not clear whether further joint optimization would be desirable: 10-100 hardware samples might not be enough to meaningfully affect the transform built from hundreds of thousands of points from simulation.

Recent works proposed using simulation to aid learning on hardware, for example [2], [3], [4]. Authors of [2] propose adding noisy evaluations from simulation to BO posterior directly. The limitation is the need to carefully balance the influence of the samples obtained from simulation versus hardware. Authors of [3] tabulate best performing points versus their average score on a behavioural metric – average contact time of their hexapod system in simulation. This metric guides trial-and-error learning to quickly find behaviours that compensate for damage of the robot. The search is done in behaviour space, and limited to pre-selected “successful” points from simulation. This helps make their search faster and potentially safer. However, if an optimal point was not pre-selected, BO cannot sample it during optimization. “Best points” are cost-specific (the map needs to be re-generated for each cost) and problem specific, so expert-knowledge is needed to apply the method to other systems. Authors of [4] propose a new distance metric using domain knowledge about bipedal locomotion. Short simulations are used to compute this metric for a large number of points (sets of control parameters), thus distinguishing points based on their behaviour in simulation, rather than the Euclidean distance between them. The method generalizes to different costs and locomotion controllers. However, the distance metric is specifically designed for bipedal locomotion. Further domain-specific expertise would be needed to adapt this approach to other settings.

Another recent direction for learning locomotion controllers utilized deep neural networks. [11] formulates the problem of learning locomotion gaits as actor-critic Reinforcement Learning with neural networks as function approximators for policy and value functions. However, it is not straightforward to make such approaches data-efficient enough for real hardware ([11] uses 10 million state-action transitions for training). So in our work we are interested in combining sample efficiency of an approach like Bayesian Optimization with the flexibility and scalability of deep neural networks.

2.2 Background on Bayesian Optimization

Bayesian Optimization is a framework for sample-efficient global search ([12] gives a recent overview). The goal is to find $\boldsymbol{x}^*$ that optimizes a given objective function $f(\boldsymbol{x})$, while executing as few evaluations of f as possible. In order to select the most promising points to evaluate next, an "acquisition" function is defined. One example is Expected Improvement (EI) function that selects $\boldsymbol{x}$ to maximize expected improvement over the value of the best result obtained so far [13]. EI requires defining the prior/posterior mean and variance of f, and Gaussian Process is frequently used for this:

$$f(\boldsymbol{x}) \sim \mathcal{GP}(\mu(\boldsymbol{x}), k(\boldsymbol{x}_i, \boldsymbol{x}_j))$$

Here μ is a mean function and k defines a kernel. $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ encodes the similarity of two inputs $\boldsymbol{x}_i, \boldsymbol{x}_j$. The value of $f(\boldsymbol{x}_i)$ has a significant influence on the posterior value of $f(\boldsymbol{x}_j)$ if $\boldsymbol{x}_i, \boldsymbol{x}_j$ have high similarity according to the kernel. Squared Exponential kernel is widely used:

$$k_{SE}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sigma_k^2 \exp\big(-\tfrac{1}{2}(\boldsymbol{x}_i - \boldsymbol{x}_j)^T \operatorname{diag}(\boldsymbol{\ell})^{-2}(\boldsymbol{x}_i - \boldsymbol{x}_j)\big),$$

where hyperparameters: σ_k^2, $\boldsymbol{\ell}$ are signal variance and a vector of length scales respectively. It is customary to adjust these automatically during optimization to learn the overall variance and how quickly f varies in each input dimension.

Gaussian Process conditioned on evidence represents a posterior distribution for f. After evaluating f at points $\boldsymbol{x}_1, ..., \boldsymbol{x}_t$ the predictive posterior $P(f_{t+1}|\boldsymbol{x}_{1:t}, \boldsymbol{y}, \boldsymbol{x}_{t+1}) \sim \mathcal{N}\big(\mu_t(\boldsymbol{x}_{t+1}), cov_t(\boldsymbol{x}_{t+1})\big)$ can be computed in closed form with mean and covariance:

$$\mu_t(\boldsymbol{x}_{t+1}) = \boldsymbol{k}^T[\boldsymbol{K} + \sigma_{noise}^2 \boldsymbol{I}]^{-1}\boldsymbol{y} \qquad cov_t(\boldsymbol{x}_{t+1}) = k(\boldsymbol{x}_{t+1}, \boldsymbol{x}_{t+1}) - \boldsymbol{k}^T[\boldsymbol{K} + \sigma_{noise}^2 \boldsymbol{I}]^{-1}\boldsymbol{k},$$

where $\boldsymbol{k} \in \mathbb{R}^t$, with $\boldsymbol{k}_i = k(\boldsymbol{x}_{t+1}, \boldsymbol{x}_i)$; $\boldsymbol{K} \in \mathbb{R}^{t \times t}$ with $\boldsymbol{K}_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$; $\boldsymbol{I}$ is an identity $\in \mathbb{R}^{t \times t}$, and $\boldsymbol{y}$ is a vector of values obtained after evaluating $f(\boldsymbol{x}_1), ..., f(\boldsymbol{x}_t)$, assuming Gaussian noise with variance σ_{noise}^2: $y_i = f(\boldsymbol{x}_i) + \epsilon_{\mathcal{N}(0, \sigma_{noise}^2)}$. More details can be found in [14].

3 Problem Formulation

In this work we aim to automatically optimize parameters of controllers for bipedal locomotion with respect to some commonly used cost functions. We assume that for a d-dimensional controller there is a bounded region of interest (a hypercube) defined by low/high limits on the values of controller parameters: $\boldsymbol{x} \in [\boldsymbol{x}_{low}, \boldsymbol{x}_{high}] \subset \mathbb{R}^n$. Some parts of this region contain points corresponding to parameter sets of the controller that yield the desired walking behavior. Such regions might comprise a large part of the space with numerous local optima, or might comprise only a small part of the space (e.g. less than 1%). In other words: we do not impose any overly restrictive assumptions on the space of controller parameters, local/global optima, or structure and properties of the cost functions of interest.

The first setting we consider is the case of optimizing 5-dimensional parameters for Raibert locomotion controller of the ATRIAS robot similar to [15], [16] and [17]. This controller has a Raibert-like foot placement policy [18]. It uses a linear feedback law operating on horizontal speed and displacement of the center of mass (CoM) to determine a desired foot touch down point:

$$x_p = k(v - v_{tgt}) + C \cdot d + 0.5 \cdot v \cdot T$$

Here, x_p is the desired location for the end of swing; v is the current speed of the CoM; k is a feedback term that regulates v towards the target speed v_{tgt}; C is a constant and d is the measured distance between the stance leg and the CoM; T is the step time. The term $0.5 \cdot v \cdot T$ is a feedforward term, similar to [18]. The swing foot trajectory is defined as a 5th order spline ending at x_p.

In stance, we regulate both the torso pitch and CoM height to maintain constant desired values:

$$F_x = K_{pt}(\theta_{des} - \theta) + K_{dt}(\dot{\theta}_{des} - \dot{\theta}) \qquad F_z = K_{pz}(z_{des} - z) + K_{dz}(\dot{z}_{des} - \dot{z})$$

These desired forces are sent to an inverse dynamics solver to return the corresponding joint torques that produce these desired ground reaction forces. Our 5-dimensional controller consists of $[k, C, T, K_{pt}, K_{dt}]$. Other parameters can be included, but the performance is not sensitive to them. This controller does not specify a target CoM trajectory. Instead it tries to maintain a constant height and torso angle in stance. The foot-placement strategy determines the resulting motion and speed.

To demonstrate applicability to a challenging setting with a higher-dimensional controller we also experiment with a Neuromuscular model for control [9]. Since it has not yet been fully adapted to work on ATRIAS in hardware, for this setting we evaluate our work on a 7-link planar model [19]. To facilitate comparison of our results with prior work in [4], we optimize over a 16-dimensional subspace of controller parameters. Description of the Neuromuscular controller and detailed information about the 16 parameters that are optimized can be found in Section III of [4]. We collect training data from simulations on flat ground. We conduct the evaluation of our approaches on perturbed models to create a simulated mismatch between simulation and hardware. We generate a set of model disturbances for each link of the robot, perturbing the mass, inertia and center of mass location up to $\pm 15\%$ of the original value. In addition, instead of walking on flat ground, we use a set of randomly generated rough ground profiles with step height of up to ± 6 *cm*.

4 Proposed Approach

The aim of our approach is to automatically learn an informed kernel for optimizing bipedal locomotion controllers with Bayesian Optimization. An uninformed kernel, like Squared Exponential, operates with vectors that represent controller parameters directly. In contrast, we learn a re-parameterization that incorporates information from simulation. We run short simulations for a range of parameter sets and record the resulting costs from the same cost function as that used in Bayesian Optimization. Costs obtained during short simulations serve as approximate indicators of the quality of the controller parameters. Our idea is to train a neural network to reconstruct the cost landscape of short simulations while focusing on the more promising parts of the space. Section 4.1 describes how this approach yields an informed kernel that helps focus the search on the well-performing regions. We also develop a cost-agnostic approach of reconstructing trajectory summaries instead of cost landscape from short simulations. This is described in Section 4.2.

4.1 Regression with Implicitly Asymmetric Loss

We consider a cost function focused on matching the desired walking speed and heavily penalizing falls:

$$cost_{atrias} = \begin{cases} 100 - x_{fall}, \text{if fall} \\ 10 \cdot ||\boldsymbol{v}_{tgt} - \boldsymbol{v}_{actual}||^2, \text{if walk} \end{cases} \tag{1}$$

where x_{fall} is the distance travelled before falling, $\boldsymbol{v}_{tgt}$ is the target velocity and $\boldsymbol{v}_{actual}$ is the vector containing actual velocities of the robot. This kind of cost function is of interest because it helps easily distinguish points that walk from points that fall. Similar costs have been considered in prior work when optimizing locomotion controllers [9, 4].

Figure 2 shows a scatter plot of applying cost from Equation 1 to simulations of Raibert controller for the ATRIAS robot as introduced in Section 3. For visualization we restrict attention to a 2-dimensional subspace of the parameter space. We pick a well-performing set of parameters (in 5D), then vary the first two dimensions to obtain a 2D subspace. The challenge comes from the fact that the boundary between the well-performing (blue) and poorly performing (yellow) parameters is discontinuous. This is a typical landscape for bipedal systems, where a controller that makes the robot fall is much worse than

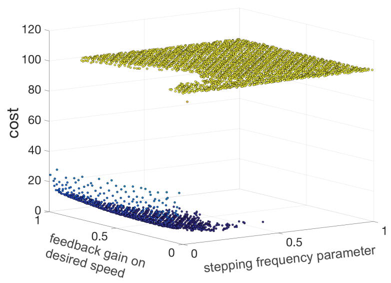


Figure 2: 2D slice of cost landscape.

one than walks, and the boundary is extremely sharp. Fitting such cost function with regression could be difficult. Learning to reconstruct the boundary exactly using the training set might result in overfitting and poor performance on the test set. Applying regularization is likely to yield high loss and uncertainty about points close to the boundary. This is particularly problematic if poorly performing points lie close to the most promising regions of the parameter space, which is the case in our setting.

We propose to use a transformation of the cost as the target for the supervised learning. Our approach is to train a deep neural network to reconstruct a reflected shifted softplus function of the cost:

$$score_{NN} = \zeta\big(c_{walk} - f_{sim}(\boldsymbol{x})\big) \tag{2}$$

Here ζ is a softplus function: $\zeta(a) = ln\big(1 + e^a\big)$, c_{walk} is the average cost for the parameter sets that walk during short simulations, $f_{sim}(\boldsymbol{x})$ is the cost computed by the simulator for vector $\boldsymbol{x}$ of controller parameter values. Using this transformation yields a “score” function such that parameter sets which produce poor results in simulation are mapped to values close to zero. With this, the differences in scores of the poorly performing parameter sets become small or zero. In contrast, the differences in scores of the parameter sets yielding potentially promising results remain proportional to the difference in the corresponding costs. Figure 3 gives a visualization of this transformation.

Cost transformation in Equation 2 serves to essentially create an asymmetric loss for neural network training. This loss is minimized when the promising (low-cost, high-score) points are reconstructed correctly. For the poorly performing (high-cost, low-score) points, it only matters that the output of the neural network is close to zero. Such asymmetric loss can be interpreted as implementing a hybrid of regression and “soft” classification. The regression aspect aims to fit the promising points which correspond to walking behaviors. The “soft” classification aspect gives an increase in the loss only if a poorly performing point is “mis-classified” as well-performing.

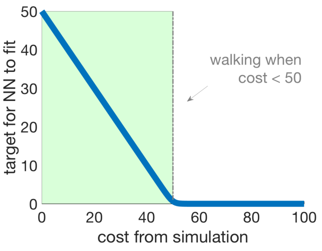


Figure 3: Cost transform.

When training the neural network we apply L1 loss instead of the usual L2 loss. With this, errors in reconstructing points on the boundary contribute only linearly to the overall loss. This helps achieve a better fit of the stable parts of the parameter space, instead of focusing on the boundary. We utilize the reconstructed transformed costs to define *asymNN* kernel for Bayesian Optimization:

$$k_{asymNN}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sigma_k^2 \exp\Big(-\tfrac{1}{2\ell^2} |score_{NN}(\boldsymbol{x}_i) - score_{NN}(\boldsymbol{x}_j)|^2\Big) \tag{3}$$

with hyperparameters σ_k^2, ℓ as described in Section 2.2. The proposed approach is able to clearly separate the unpromising part of the parameter space. Under the resulting distance metric poorly performing sets of parameters are close together and far from well-performing ones.

4.2 Reconstructing Cost-agnostic Trajectory Summaries

Utilizing costs obtained from short simulations provides a way to build an informed kernel without specifying any additional domain knowledge. If simulations are computationally expensive it is desirable to minimize the need to repeat data collection. Often, the cost function needs to be modified to accommodate different objectives, hence, there is a need for a cost-agnostic approach. For such cases we propose to train a neural network to reconstruct summaries of trajectories that are cost-agnostic, then utilize these trajectory summaries for constructing kernel distance metric.

We summarize trajectories by recording fairly generic aspects of locomotion: walking time (time before falling), energy used during walking, position of the torso, angle of the torso, coordinates of the center of mass at the end of the short simulation runs. These summaries of simulated trajectories are collected for a range of controller parameters and comprise the training set for the neural network to fit (input: $\boldsymbol{x}$ – a set of control parameters; output: $\boldsymbol{traj_x}$ – the corresponding trajectory summary obtained from simulation). The outputs of the (trained) neural network offer the reconstructed/approximate trajectory summaries: $f_{NN}(\boldsymbol{x}) = \widehat{\boldsymbol{traj}}_{\boldsymbol{x}}$, where $\boldsymbol{x}$ is the input controller parameters, and $\widehat{\boldsymbol{traj}}_{\boldsymbol{x}}$ is the corresponding reconstructed trajectory summary. These are then used to construct an informed cost-agnostic kernel for Bayesian Optimization:

$$k_{trajNN}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sigma_k^2 \exp\left(-\tfrac{1}{2}\boldsymbol{t}_{ij}^T \operatorname{diag}(\boldsymbol{\ell})^{-2}\boldsymbol{t}_{ij}\right), \qquad \boldsymbol{t}_{ij} = f_{NN}(\boldsymbol{x}_i) - f_{NN}(\boldsymbol{x}_j) \tag{4}$$

The general concept of utilizing trajectory data to improve sample efficiency of BO has been proposed before, for example in [20]. However, prior work assumed obtaining trajectory data is possible every time kernel values $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ need to be evaluated. This is not the case in our setting. Trajectory summaries are initially obtained via costly high-fidelity simulations, and it would be infeasible to compute trajectory information via simulation during BO. Hence, our approach is to train a neural network to learn reconstructing trajectory summaries first. Running a forward pass of the neural network is a relatively inexpensive operation, hence reconstructed/approximate trajectory summaries can be quickly obtained during BO whenever $k_{trajNN}(\boldsymbol{x}_i, \boldsymbol{x}_j)$ needs to be computed.

When defining trajectory summaries we did not focus on carefully selecting what aspects to include/exclude. Our goal was an approach that could be quickly adapted to other domains. When applying this approach to a new domain, the strategy could be simply to include trajectory information used to compute cost functions that are of interest/relevance in the domain. For example, for a manipulator, the coordinates of end-effector(s) could be recorded at relevant points. In principle, our approach also could utilize domain- and task-specific 'descriptors', like those proposed in [4, 3].

5 Experimental Results

In this section we describe our experiments with cost-based and trajectory-based kernels. We first consider the setting of optimizing a 5-dimensional controller for the ATRIAS robot. We show that the cost-based kernel is able to improve sample efficiency over standard Bayesian Optimization. We present hardware experiments to demonstrate that our kernel allows obtaining a set of parameters close to optimal on the second trial. We then discuss simulation experiments with a 16 dimensional controller that utilizes a Neuromuscular model [9]. These experiments show that our trajectory-based kernel is able to significantly outperform standard Bayesian Optimization for a higher-dimensional controller even when a sharply discontinuous cost is used during optimization.

5.1 Experiments with Raibert controller on the ATRIAS robot

For our experiments on the ATRIAS robot we used a high-fidelity ATRIAS simulator [17] to generate the kernel. We did an initial analysis of the performance of our approach in simulation, followed by hardware experiments. As described in Section 4.1, we trained a neural network to reconstruct cost information obtained from short simulations. We created a sobol grid on the input parameter space with 20K points and ran short 3.5 second simulations on each of the corresponding 20K parameter sets to compute the costs. We then trained a fully connected network with 4 hidden layers (128, 64, 16, 4 units) to reconstruct $score_{NN}$ (the transformation of the cost described in Section 4.1).

In Figure 4 we first compare the performance of BO that used our neural network-based kernel (*asymNN*) versus using a standard Squared Exponential kernel (*SE*) in simulation. For these experiments we used the cost from Equation 1, Section 4.1 with target velocity of $1m/s$. Simulations with cost less than 50 yielded walking behavior, those with cost less than 20 yielded stable walking close to the target speed. BO with *asymNN* kernel reliably found stable walking points after only 8 trials. In contrast, BO with *SE* kernel did not find stable walking solutions in the first 20 trials reliably.

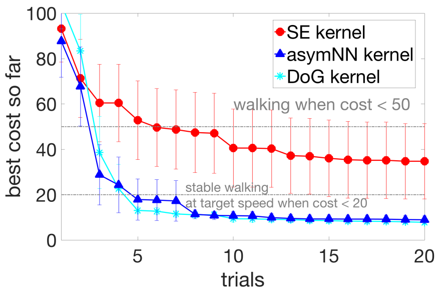
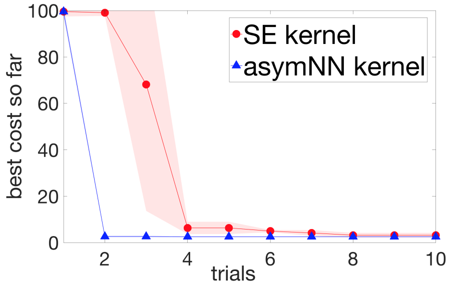


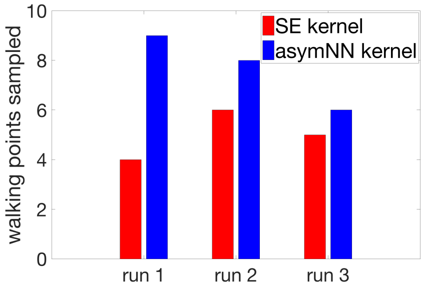
Figure 4: Initial tests of Bayesian Optimization for 5-dimensional controller in simulation. The plot shows mean of best cost so far over 30 runs for each kernel, error bars are 95% confidence intervals.

We also compare with a recently proposed Determinants of Gait (*DoG*) kernel [4]. *DoG* utilized more specific domain knowledge to construct an informed kernel for BO of locomotion controllers for a fixed set of points. *asymNN* was able to closely match the performance of *DoG* in this setting after 8 trials.

After experiments in simulation suggested that *asymNN* kernel can yield a significant improvement in sample efficiency of BO, we conducted a set of ex-



(a) Best cost so far during BO (mean over 3 runs, shaded region shows ± 1 st. dev.)



(b) Number of "walking" points sampled (out of 10 trials in each run)

Figure 6: Hardware experiments on the ATRIAS robot.

periments on the ATRIAS robot. We completed 6 sets of runs of BO: 3 using *asymNN* kernel and 3 using a standard *SE* kernel with 10 trials each, leading to a total of 60 hardware experiments.

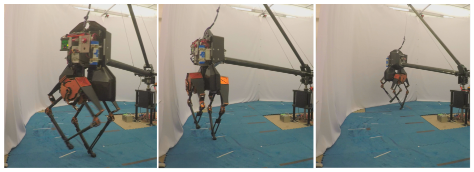
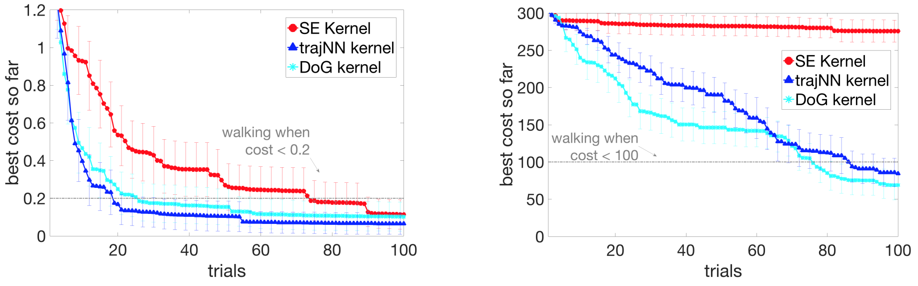


Figure 5: ATRIAS during BO with *asymNN* kernel.

Since ATRIAS walks around a rather short boom in 2D, walking at high speeds needs a lot of torque from the robot motors. This means higher lateral forces between the robot and the boom, which do not affect our direction of motion but can lead to a lot of internal forces, eventually breaking the robot. So, in our first attempt, we tried to start with lower speeds of $0.4m/s$ so that we could do hardware experiments and analyze the validity of our approach on hardware without breaking the robot too often. In this setting with low target speed, stable walking points comprised $\approx \frac{1}{6}$ of the parameter space. We anticipated it would be challenging to improve over BO with *SE* kernel, since it was able to find stable walking solutions after only 3-4 trials.

Figure 6a shows the performance of BO with SE versus *asymNN* kernel. SE obtains a stable walking solution on the 3rd trial in one run, and on the 4th trial in the two other runs. *asymNN* kernel is able to find the best-performing set of parameters on the second trial in each of the 3 runs. This confirms that using *asymNN* kernel offers an improvement over using *SE* kernel in this setting. We suggest that *asymNN* reliably selects an excellent point on the 2nd trial because such points lie far from poorly performing subspace of parameters (under the distance metric constructed with *asymNN*). *asymNN* kernel also helped sampling more walking points overall (Figure 6b). This is desirable as stable points are less likely to break the robot.

While in the above hardware setup most methods are likely to sample walking points within 10 trials, we believe our experimentation is an important step towards optimizing locomotion policies for complex humanoid robots. BO studies in the past also used real robot hardware (e.g. [1, 3, 5]). However, [5, 21, 3] used robots which



(a) Using smooth cost from Equation 5 (b) Using non-smooth cost from Equation 6

Figure 7: Bayesian Optimization for the Neuromuscular model controller in simulation. *trajNN* and *DoG* kernels were constructed with undisturbed model on flat ground. BO is run with mass/inertia disturbances on different rough ground profiles to simulate mismatch. Plots show means over 50 runs, 95% confidence intervals.

are statically stable for significant parts of their gait, making discontinuities in the cost function landscape less likely and in turn making the optimization easier. In contrast, ATRIAS is a complex bipedal system which is likely to fall with unstable controllers due to point feet. [1] use a walking robot similar to ours. However, their controller parametrization is very different, and not widely used, unlike our inverse dynamics and force-based controller which is more modern and state-of-the-art [7], [22], [6]. While with our hardware setting it might be hard to show improvement over simpler approaches at low constant target speeds, we believe the setting is adequate, because our testbed is fairly complex and our problem formulation is widely applicable. In Appendix A we describe initial results for variable target speed experiments, with SE kernel not finding walking solutions reliably even after 20 trials and asymNN succeeding after the first 10 trials.

5.2 Experiments with the Neuromuscular Model

16-dimensional controller of the Neuromuscular model (described in Section 3) yielded a challenging optimization setting: walking points comprised less than 2% of the parameter space in simulation. Here we describe our experiments with cost-agnostic approach for constructing an informed kernel introduced in Section 4.2. We created a grid of 100K points in the input parameter space and ran short 5 second simulations on each of the corresponding 100K parameter sets to collect the trajectory summaries. We then trained a fully connected network with 3 hidden layers (512, 128, 32 units) with L1 loss to reconstruct 8-dimensional trajectory summaries (as described in Section 4.2). All experiments were done on perturbed models, as described in Section 3.

Figure 7 compares using *trajNN* versus *SE* kernel for BO with two different costs from prior literature. The first cost promotes walking further and longer before falling, while penalizing deviations from the target speed [4]:

$$cost_{smooth} = 1/(1+t) + 0.3/(1+d) + 0.01(s - s_{tgt}), \tag{5}$$

where t is seconds walked, d is the final hip position, s is mean speed and s_{tgt} is the desired walking speed ($1.3m/s$ in our case). The second cost function is a simplified version of the cost used in [9]. It penalizes falls explicitly, and encourages walking at desired speed and with lower cost of transport:

$$cost_{non\text{-}smooth} = \begin{cases} 300 - x_{fall}, \text{if fall} \\ 100||v_{avg} - v_{tgt}|| + c_{tr}, \text{if walk} \end{cases} \tag{6}$$

where x_{fall} is the distance covered before falling, v_{avg} is the average speed of walking, v_{tgt} is the target velocity, and c_{tr} captures the cost of transport.

Figure 6a shows that *trajNN* offers a significant improvement in sample efficiency when using $cost_{smooth}$. Points with cost less than 0.2 correspond to robust walking behavior. With *trajNN*, more than 90% of runs obtain walking solutions after only 25 trials. In contrast, using *SE* requires more than 90 trials for such success rate. The performance of *trajNN* matches that of a *DoG* kernel from prior work [4]. This is notable, since *trajNN* is learned automatically, whereas *DoG* kernel is constructed using domain expertise. Figure 7b shows that *trajNN* also provides a significant improvement when using the second cost. Points with cost less than 100 correspond to walking. With *trajNN*, 70% of the runs find walking solutions after 100 trials. In contrast, optimizing non-smooth cost is very challenging for BO with *SE* kernel: a walking solution is found only in 1 out of 50 runs after 100 trials. The difference in performance on the two costs is due to the nature of the costs. For example, if a point walks some distance d, Equation 5 includes a term $\frac{1}{d}$ and Equation 6 includes $-d$. A sharper fall in the first cost causes BO to exploit around points that walk some distance. It then quickly finds points that walk forever, while BO with the second cost continues to explore.

6 Conclusion and Future Work

In this work we proposed learning informed kernels for Bayesian Optimization of locomotion controllers without relying heavily on domain experts. We optimized a 5-dimensional controller on the ATRIAS robot and showed that our cost-based kernel offered an improvement over using an uninformed kernel. We also proposed a cost-agnostic alternative. Experiments with a 16-dimensional Neuromuscular controller in simulation showed a significant improvement with different costs.

In future work it would be interesting to further analyze various approaches that enhance sample efficiency of BO. Approaches that embed simulation-based information into the kernel (like those we proposed) can enhance sample efficiency dramatically by focusing BO on regions that look promising in simulation. Approaches that use simulation-based samples in BO posterior mean directly (e.g. [2]) could be more robust to simulation-based inaccuracies after collecting a larger amount of data from hardware experiments. However, they can only incorporate cost-based information (e.g. no way to add trajectory information directly to the posterior mean). Perhaps there is an effective way to combine the two directions.

Another promising line for future work is learning flexible models of simulation-vs-hardware mismatch. Such models could help decrease the influence of distortion from incorrect simulations and could help enhance both 'kernel-based' and 'mean-based' methods.

Acknowledgments

This research was supported in part by National Science Foundation grant IIS-1563807, the Max-Planck-Society, & the Knut and Alice Wallenberg Foundation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding organizations.

References

[1] R. Calandra, A. Seyfarth, J. Peters, and M. P. Deisenroth. Bayesian Optimization for Learning Gaits Under Uncertainty. *Annals of Mathematics and Artificial Intelligence*, 76(1-2):5–23, 2016.

[2] A. Marco, F. Berkenkamp, P. Hennig, A. P. Schoellig, A. Krause, S. Schaal, and S. Trimpe. Virtual vs. real: Trading off simulations and physical experiments in reinforcement learning with bayesian optimization. *arXiv preprint arXiv:1703.01250*, 2017.

[3] A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret. Robots that can adapt like animals. *Nature*, 521(7553):503–507, 2015.

[4] R. Antonova, A. Rai, and C. G. Atkeson. Sample efficient optimization for learning controllers for bipedal locomotion. In *Humanoid Robots (Humanoids), 2016 IEEE-RAS 16th International Conference on*, pages 22–28. IEEE, 2016.

[5] M. Tesch, J. Schneider, and H. Choset. Using response surfaces and expected improvement to optimize snake robot gait parameters. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 1069–1074. IEEE, 2011.

[6] S. Feng, E. Whitman, X. Xinjilefu, and C. G. Atkeson. Optimization-based full body control for the darpa robotics challenge. *Journal of Field Robotics*, 32(2):293–312, 2015.

[7] S. Kuindersma, R. Deits, M. Fallon, A. Valenzuela, H. Dai, F. Permenter, T. Koolen, P. Marion, and R. Tedrake. Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot. *Autonomous Robots*, 40(3):429–455, 2016.

[8] H. Geyer and H. Herr. A Muscle-reflex Model that Encodes Principles of Legged Mechanics Produces Human Walking Dynamics and Muscle Activities. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 18(3):263–273, 2010.

[9] S. Song and H. Geyer. A Neural Circuitry that Emphasizes Spinal Feedback Generates Diverse Behaviours of Human Locomotion. *The Journal of Physiology*, 593(16): 3493–3511, 2015.

[10] R. Calandra, J. Peters, C. E. Rasmussen, and M. P. Deisenroth. Manifold gaussian processes for regression. In *Neural Networks (IJCNN), 2016 International Joint Conference on*, pages 3338–3345. IEEE, 2016.

[11] X. B. Peng, G. Berseth, and M. van de Panne. Terrain-adaptive locomotion skills using deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 35(4):81, 2016.

[12] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas. Taking the Human Out of the Loop: A Review of Bayesian Optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

[13] J. Mockus, V. Tiesis, and A. Zilinskas. Toward Global Optimization, volume 2, chapter Bayesian Methods for Seeking the Extremum. 1978.

[14] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005. ISBN 026218253X.

[15] W. C. Martin, A. Wu, and H. Geyer. Experimental evaluation of deadbeat running on the atrias biped. *IEEE Robotics and Automation Letters*, 2(2):1085–1092, 2017.

[16] C. Hubicki, A. Abate, P. Clary, S. Rezazadeh, M. Jones, A. Peekema, J. Van Why, R. Domres, A. Wu, W. Martin, et al. Walking and running with passive compliance: Lessons from engineering a live demonstration of the atrias biped. *IEEE Robotics and Automation Magazine*, 2(4.1):4–1, 2016.

[17] W. C. Martin, A. Wu, and H. Geyer. Robust spring mass model running for a physical bipedal robot. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 6307–6312. IEEE, 2015.

[18] M. H. Raibert. *Legged robots that balance.* MIT press, 1986.

[19] 16D Simulator for Neuromuscular Models for Biped Locomotion. *Code available from https://github.com/nthatte/Neuromuscular-Transfemoral-Prosthesis-Model.*

[20] A. Wilson, A. Fern, and P. Tadepalli. Using Trajectory Data to Improve Bayesian Optimization for Reinforcement Learning. *The Journal of Machine Learning Research*, 15(1):253–282, 2014.

[21] D. J. Lizotte, T. Wang, M. H. Bowling, and D. Schuurmans. Automatic gait optimization with gaussian process regression. In *IJCAI*, volume 7, pages 944–949, 2007.

[22] A. Herzog, N. Rotella, S. Mason, F. Grimminger, S. Schaal, and L. Righetti. Momentum control with hierarchical inverse dynamics on a torque-controlled humanoid. *Autonomous Robots*, 40(3):473–491, 2016.

Bayesian Optimization Using Domain Knowledge on the ATRIAS Biped

Akshara Rai[1 §], Rika Antonova[1 *], Seungmoon Song[§], William Martin[§], Hartmut Geyer[§], Christopher G. Atkeson[§]

[§]Robotics Institute, Carnegie Mellon University, USA
[*]EECS, KTH, Stockholm, Sweden

Abstract

Robotics controllers often consist of expert-designed heuristics, which can be hard to tune in higher dimensions. Simulation can aid in optimizing these controllers if parameters learned in simulation transfer to hardware. Unfortunately, this is often not the case in legged locomotion, necessitating learning directly on hardware. This motivates using data-efficient learning techniques like Bayesian Optimization (BO) to minimize collecting expensive data samples. BO is a black-box data-efficient optimization scheme, though its performance typically degrades in higher dimensions. We aim to overcome this problem by incorporating domain knowledge, with a focus on bipedal locomotion. In our previous work, we proposed a feature transformation that projected a 16-dimensional locomotion controller to a 1-dimensional space using knowledge of human walking. When optimizing a human-inspired neuromuscular controller in simulation, this feature transformation enhanced sample efficiency of BO over traditional BO with a Squared Exponential kernel. In this paper, we present a generalized feature transform applicable to non-humanoid robot morphologies and evaluate it on the ATRIAS bipedal robot, in both simulation and hardware. We present three different walking controllers and two are evaluated on the real robot. Our results show that this feature transform captures important aspects of walking and accelerates learning on hardware and simulation, as compared to traditional BO.

[1]Both of these authors contributed equally.

Published in the proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA). Available at: https://ieeexplore.ieee.org/abstract/document/8461237

Global Search with Bernoulli Alternation Kernel for Task-oriented Grasping Informed by Simulation

Rika Antonova[1], **Mia Kokic**[1], **Johannes A. Stork**, **Danica Kragic**

Robotics, Perception and Learning, EECS
KTH Royal Institute of Technology Sweden, Stockholm, Sweden

Abstract

We develop an approach that benefits from large simulated datasets and takes full advantage of the limited online data that is most relevant. We propose a variant of Bayesian optimization that alternates between using informed and uninformed kernels. With this Bernoulli Alternation Kernel we ensure that discrepancies between simulation and reality do not hinder adapting robot control policies online. The proposed approach is applied to a challenging real-world problem of task-oriented grasping with novel objects. Our further contribution is a neural network architecture and training pipeline that use experience from grasping objects in simulation to learn grasp stability scores. We learn task scores from a labeled dataset with a convolutional network, which is used to construct an informed kernel for our variant of Bayesian optimization. Experiments on an ABB Yumi robot with real sensor data demonstrate success of our approach, despite the challenge of fulfilling task requirements and high uncertainty over physical properties of objects.

Keywords: Bayesian optimization, Deep learning, Task-oriented grasping

1 Introduction

Recent advances in deep learning motivated using data-driven methods in robotics. However, collecting large amounts of training data is challenging, since it requires actual execution on a real system. One way to trim hours of execution time is to use simulation. Simulators make simplifying assumptions, so often there is mismatch between simulation and real world. We propose to bridge this gap by using a variant

[1]Both of these authors contributed equally.

In proceedings of the 2nd Conference on Robot Learning (CoRL 2018), Zürich, Switzerland.
Published by PMLR http://proceedings.mlr.press/v87/antonova18a License: CC-BY 4.0

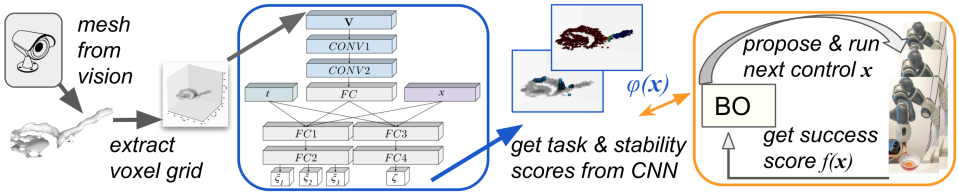


Figure 1: Overview of the overall approach. Blue highlights components trained offline; orange highlights online learning components that guide real-time decisions made by the robot. A vision system is used to first construct a mesh from raw point cloud. Then, a set of grasping points, voxelized mesh and task identity are passed through the network to obtain task suitability and grasp stability scores. These are used to construct informed kernel for BO. Then, the robot executes online search with BO to find the best task-appropriate grasp.

of Bayesian optimization (BO) that incorporates simulation-based information in a way that is robust to mismatch between simulation and reality. Furthermore, we apply this to the problem of task-oriented grasping. The challenge is that successful grasps are limited to specific object parts that afford the execution of a task. This presents a highly constrained problem.

Our first contribution is a variant of BO that alternates between using informed and uninformed kernels. This allows us to explore grasp adjustments online, while exploiting simulation-based information and feedback from previous grasp attempts. To inform the kernel for our BO algorithm, we propose a task-oriented grasp model. This further contribution is a deep neural network architecture that maps raw visual observation of an object to task-oriented grasp scores. The network is trained on large amounts of synthetic data. This allows us to compute task-oriented grasp candidates for previously unseen objects from online perceptual information. Simulation-based knowledge provides additional guidance for online search, making it more sample-efficient. Overall, our approach aims to emulate the human-like strategy of attempting several adjustments until a successful grasp is accomplished; Figure 1 gives a brief visual overview. We conduct experiments on an ABB Yumi robot with real sensor data, and demonstrate success of our approach to fulfill task requirements when grasping novel objects, despite high uncertainty over their physical properties.

2 Background and Related Work

2.1 Bayesian Optimization Informed By Simulation

Bayesian optimization (BO) is a data-efficient global search method (see [1] for an overview). Let $\boldsymbol{x}$ be a set of control parameters (e.g. for grasping: the approach direction and orientation of the end-effector). The problem is to find $\boldsymbol{x}$ that optimizes a given objective/cost function $f(\boldsymbol{x})$. The objective function encodes characteristics of the desired outcome: e.g. low score for failing to grasp an object, high score for grasping and completing the desired task. BO starts with a prior expressing uncertainty over $f(\boldsymbol{x})$. After each real-world trial, BO constructs a posterior based on data obtained so far. It then uses an auxiliary function (*acquisition function*) to choose the next $\boldsymbol{x}$ to evaluate. The acquisition function selects points for which the posterior estimate of the objective f is promising. It takes into account both the posterior mean and covariance. Gaussian Process (GP) is commonly used to model the cost function: $f(\boldsymbol{x}) \sim GP(\mu(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}'))$. Its kernel function captures similarity between inputs: if $k(\boldsymbol{x}, \boldsymbol{x}')$ is large for $\boldsymbol{x}, \boldsymbol{x}'$, then $f(\boldsymbol{x})$ has high influence on $f(\boldsymbol{x}')$. Squared Exponential (SE) kernel is frequently the default choice (its hyperparameters σ_k^2, $\boldsymbol{\ell}$ can be optimized automatically):

$$k_{SE}(\boldsymbol{x}, \boldsymbol{x}') = \sigma_k^2 \exp\big(-\tfrac{1}{2}(\boldsymbol{x}-\boldsymbol{x}')^T \operatorname{diag}(\boldsymbol{\ell})^{-2}(\boldsymbol{x}-\boldsymbol{x}')\big), \tag{1}$$

BO with standard kernels (SE or, more generally, Matérn kernels) is effective in cases with low noise, smooth objective/cost functions or limited search space. In contrast, robotics problems exhibit high noise, sharp transitions between low- and high-cost regions, inability to limit the search space a priori without excluding optimal regions. Previous work explored using kernels informed by simulation [2, 3] and by the domain dynamics [4], in some cases allowing to learn kernel transforms automatically [5]. However, these assumed access to high-fidelity simulators/models. For many areas of robotics such simulators are not available. High anticipated simulation-reality mismatch prevents us from using simulation-based information in standard ways, like constructing the prior of the mean function for the Gaussian Process in BO. As noted in [6], in practice it can be challenging to specify prior mean effectively. Recent experiments in robotics show that performance of 'prior-based' BO can degrade in cases with significant simulation-reality mismatch [7]. Hence, embedding simulation-based information into the kernel could be a more robust alternative.

2.2 Task-oriented Grasping

Grasping simulators [8] can help generating a set of grasps labeled with quality measures, which can be used to select the best grasp for execution on a real robot. However, these quality measures often rely on access to full 3D geometry of an object, which is often not obtainable in real-world scenarios. To overcome this, recent work investigated learning to predict grasp success from partial visual inputs [9], [10], [11]. Vision alone often does not provide enough information about important object properties: it is difficult to infer inertial and, even more so, friction properties. Hence, several works proposed learning adaptively with feedback from real robot trials [12, 13, 14]. Some proposed constructing priors from simulation, but as noted in the BO background subsection, directly adding prior points or using a fixed prior mean is problematic with high simulation-reality mismatch. In the case of grasping: only medium- to low-fidelity simulators are generally available; their results only align with reality if crucial parameters, like friction, are measured and modelled meticulously. We do not assume access to estimates of friction or inertial properties of objects, and do not require modeling these in simulation either. Our main goal is to develop a method whose performance does not degrade due to incorporating data from highly inaccurate simulations. Hence, we draw inspiration from approaches that embed simulation-based information into the kernel (e.g. [15], though this prior work investigated convergence of MABs with a large number of pick-and-place simulated grasps across objects, while we emphasize finding successful task-oriented grasps with only a few BO trials on the real robot and adapt to each object separately).

Approaches from prior work described above can be successful for grasping an object in a stable manner. However, in most scenarios the ultimate goal is to allow an execution of a manipulation task, for example cutting or pouring. This problem is known as Task-oriented Grasping (TOG) [16], [17] and is vastly more challenging than just stable grasping. The first challenge is to find grasps that are stable and also task compliant. For example, when grasping a mug most of the stable grasps for smaller parallel grippers are on the rim. However, if we want to use the mug for pouring, fingers should avoid the opening area and instead aim for the handle. This is challenging when vision is imperfect, since small task-relevant areas of the object could be obscured or distorted. The second challenge is that a robot must be able to reason about geometric properties of an object and how they relate to tasks (e.g. openings & pouring, blades & cutting). This is know as *affordance* learning and many authors have addressed this problem in the past. While some works focus solely on reasoning about object affordances, others utilize them for purpose of TOG. In [18] authors trained a CNN to predict part affordances, which are used to formulate task constraints, namely the ones on the location and orientation of the gripper. These constraints are given to optimization-based grasp planner, which executes task-oriented grasps. Similarly, [19] trained task-oriented CNNs to identify the regions a robot is allowed to contact to fulfill a task; they use a part-based approach that finds object parts with shape that is compatible with the gripper. In [20] authors proposed training TOG Network to optimize for task-oriented grasp and manipulation policy. They decomposed the problem into 1) finding task-agnostic grasps for which they use Dex-Net 2.0 [10] and 2) finding task-oriented grasps for which they train a CNN. Our work differs from this in several ways. We train a network that jointly predicts grasp stability and task suitability. Furthermore, although we could use a single score (like Dex-Net 2.0) for predicting stability, we chose to incorporate three different metrics. This provides a richer signal for BO: different metrics could be of varying importance for different tasks (e.g. one metric could score grasps based on the distance from the center of mass of the object, while another might be most sensitive to positions of the finger joints). Moreover, we train our network on a large data set of realistic 3D objects and six different tasks, while [20] use procedurally generated objects based on shape primitives and consider only two tasks (pounding and sweeping). Hence, in our case the complexity of learning object-task-grasp relationships is significantly higher.

3 Proposed Method

3.1 Bayesian Optimization with Bernoulli Alternation Kernel

Our aim is to develop an approach robust to high simulation-reality mismatch. This motivates us to look beyond solutions that put simulation-based information into GP prior or rely on simulation-based kernels alone. We first note that using a sum of kernels could be beneficial. Let $k_{sum}(\boldsymbol{x}, \boldsymbol{x}') = k_{SE}(\boldsymbol{x}, \boldsymbol{x}') + k_{\phi}(\boldsymbol{x}, \boldsymbol{x}')$ be a kernel comprised of k_{SE} (Equation 1) and $k_{\phi} = k_{SE}(\phi(\boldsymbol{x}), \phi(\boldsymbol{x}'))$, where $\phi(\cdot)$ is akin to a warping function. Recall that in our case $\boldsymbol{x}$ is a vector of control parameters. To obtain $\phi(\boldsymbol{x})$ we could execute controls $\boldsymbol{x}$ in simulation and output relevant characteristics of the result (e.g. stability metrics for a grasp). It is useful to embed ϕ into the kernel, because we can collapse the space of unsuccessful controls. For example, ϕ could give near zero stability scores to grasps that miss or barely touch objects. This would indicate that all such failed points/controls are similar to each other, but dissimilar from successful regions of control parameters. BO can then quickly learn to neglect the non-promising regions, even though they could be far away from each other in the original space of control parameters. However, when ϕ fails to provide high-quality information, k_{sum} could be adversely impacted by the k_{ϕ} component.

To offer a more robust alternative, we propose an approach that takes contributions from both k_{SE} and k_{ϕ}, but ensures BO can not be mislead by a poor choice of ϕ. At each BO iteration/trial, we randomize the choice of whether k_{SE} or k_{ϕ} is used. For this, we define a probability distribution over kernels and draw a kernel function to be used at each BO trial independently. This defines k_{bak} as:

$$k_{bak}(\boldsymbol{x}, \boldsymbol{x}') \sim \mathbb{1}_{\{\theta \leq 0.5\}} k_{SE}(\boldsymbol{x}, \boldsymbol{x}') + \mathbb{1}_{\{\theta > 0.5\}} k_{\phi}(\boldsymbol{x}, \boldsymbol{x}'); \quad \theta \sim Uniform(0, 1) \tag{2}$$

Note that after each iteration/trial n, the data for computing the GP posterior consists of all the points sampled so far: $\boldsymbol{D}_n = \{(\boldsymbol{x}_i, f(\boldsymbol{x}_i)) | i = 1, ..., n\}$. In BO, posterior is usually re-computed after each new sample. This aspect of BO allows us to make a choice of the kernel function for each iteration separately. So, after n iterations/trials, we first pick a kernel function k_n using Equation 2. We then compute GP posterior mean and covariance (notation from [6]):

$$\begin{aligned} mean(f_*) &= \boldsymbol{k}_{*_n}^T (K_n + \sigma_{noise}^2 I)^{-1} \boldsymbol{y}_n \\ cov(f_*) &= k_n(\boldsymbol{x}_*, \boldsymbol{x}_*) - \boldsymbol{k}_{*_n}^T (K_n + \sigma_{noise}^2 I)^{-1} \boldsymbol{k}_{*_n} \end{aligned} \tag{3}$$

where $\boldsymbol{x}_*$ is a new point whose cost we want to predict; $f_* := f(\boldsymbol{x}_*)$; K_n is $n \times n$ matrix with $K_{ij} = k_n(\boldsymbol{x}_i, \boldsymbol{x}_j)$; $\boldsymbol{k}_{*_n} \in \mathbb{R}^n$ is a vector of covariances between $\boldsymbol{x}_*$ and each $\boldsymbol{x}_i$, $i=1, ..., n$; $\boldsymbol{y}_n$ is a vector of evaluations for the sampled points: $[\boldsymbol{y}_n]_i = f(\boldsymbol{x}_i)$. Algorithm 1 gives a concise summary.

For an intuitive insight, note that points sampled for trials when k_ϕ is used could provide fast guidance towards useful parts of the search space. These points are included in the data for subsequent posterior computations, enabling even trials in which k_{SE} is used to propose better next choices. This is important if probability of discovering a promising region is small, which is frequent in robotics. If k_ϕ is not useful, or even misleading, choices made on the trials that use k_{SE} are not impacted by poor ϕ, since with our approach BO's acquisition function makes its decisions using only one of the kernels at a time.

Algorithm 1: BO-BAK

```
sample $\boldsymbol{x}_1$ randomly, get $y_1 = f(\boldsymbol{x}_1)$ from real world
initialize: $\boldsymbol{D}_1 = \{(\boldsymbol{x}_1, y_1)\}$
for $n = 1, 2, ...$ do
    sample kernel function $k_n$ using Equation 2
    get posterior GP mean & cov using $\boldsymbol{D}_n$ & Eq.3
    select $\boldsymbol{x}_{n+1}$ by optimizing acquisition function:
        $\boldsymbol{x}_{n+1} = \arg\max_{\boldsymbol{x}} \alpha(\boldsymbol{x}; \boldsymbol{D}_n)$
    get $y_{n+1} = f(\boldsymbol{x}_{n+1})$ from real world
    augment data $\boldsymbol{D}_{n+1} = \boldsymbol{D}_n \cup \{(\boldsymbol{x}_{n+1}, y_{n+1})\}$
```

In Section 4.1 we evaluate on analytic functions that exhibit challenges similar to our target domain. We confirm that informed kernels can provide significant improvement over conventional BO. k_ϕ and k_{sum} appear somewhat sensitive to the quality of the warping function ϕ, while k_{bak} appears to be more robust empirically. As a preliminary theoretical comment, we observe that k_{bak} retains optimality guarantees of the conventional BO. Intuitively, this is because in expectation we perform N/2 trials using only SE to choose the next point. This is no worse than using conventional BO in half of the trials, and we anticipate that satisfactory consistency and regret bounds could be derived. We leave this theoretical analysis to future work. In this work, we investigate whether k_{bak} is beneficial in complex real-world robotics scenarios. These pose a further challenge, since often the assumptions made to obtain consistency and regret bounds for conventional BO do not hold in practice.

3.2 Task-oriented Grasping, Network Architecture and Training

The objective of our TOG CNN is to learn to output several grasp stability scores ξ_i and a task suitability score ζ, given as input a task t and a grasp $\boldsymbol{x}$ on an object. A grasp is parameterized by $\boldsymbol{x} := (\boldsymbol{p}, \boldsymbol{n}, \psi, d) \in \mathbb{R}^8$, where $\boldsymbol{p} \in \mathbb{R}^3$ is a point on the surface of the object and $\boldsymbol{n} \in \mathbb{R}^3$ is its corresponding unit normal, ψ is a gripper roll and d is an offset from the surface of the object. A task t is defined as a single manipulation action that starts with a grasp. To fully exploit the power of 3D representations we use volumetric architecture in convolutional layers. For this, we scale an object mesh to fit inside $50 \times 50 \times 50$ binary voxel grid (for learning, we also scale the grasping points). The actual input to the network is then: a voxel grid of an object ($\boldsymbol{V}$), a grasp ($\boldsymbol{x}$), and a task (t, encoded as 1×6 one-hot vector). For details of network architecture see Figure 2. To handle noisy data that can be encountered in the real-world, the network is trained with dropout (ratio of 0.5) in

Task	Object Class	Grasp Requirements
handover	all objects	leave handle(s) clear
screw	screwdrivers	avoid the shaft
cut	knife and scissors	avoid blade(s)
pour	bottle, can, mug, wine glass	avoid the opening area
support	pan, spatula, spoon, fork	avoid the supporting area
pound	hammer	avoid a hammer head

Figure 2: Objects, tasks and grasps requirements in our dataset.

the first layer. Furthermore, for training we randomly rotate objects to account for possible orientations for tabletop grasping. The training was performed using Tensorflow on a Titan X GPU.

A single CNN is trained for all object categories and tasks (no need to classify the object explicitly). The kernel for BO is constructed "on the fly" with a forward pass on the trained CNN. To grasp a new object the steps are: 1) vision processes the object (point cloud $\rightarrow$ mesh $\rightarrow$ voxel grid); 2) voxel grid & task id are fed as input to CNN; 3) we get as output predicted stability & task scores for any sets of grasp parameters, which enables computing kernel distances quickly during BO. Once the CNN is trained, we have quick access to a kernel for any object type and task that are included in CNN training. BO is run for each object instance "from scratch", but with informed kernel incorporated into the search. In collaborative industrial settings (where the same objects could be used for many days) our method could identify optimal task-specific grasping parameters and these can be utilized for a given tool without re-optimizing. If it is unlikely that the robot will deal with the same tool/object again, BO would run each time for each new object instance. This relieves us from making restrictive assumptions about object properties that can't be inferred from vision only.

3.3 Data Generation

To obtain learning targets for our network, we simulate grasping various objects in OpenRave [8] with a parallel gripper. We provide the simulator with a "free-floating" 3D model of the gripper and objects. The objects dataset consists of 605 mesh models from ShapeNET [21] and ModelNet40 [22], containing objects from 13 different categories. For training we align each mesh with a reference frame that coincides with object's principal axes and meshes are scaled to real-world size based on object's category.

We label the objects with target task scores by assigning positive/negative labels to parts of the object suitable/unsuitable for the task. Figure 2 shows the tasks we consider and relations to applicable objects. We execute 4500 grasps on each object with grasp parameters as follows:

- **point & normal** $(\boldsymbol{p}, \boldsymbol{n})$: randomly sample 500 points on object's surface (along with the corresponding normal directions)

- **gripper roll** ψ: try angles $\psi \in \{0, \pi/2, \pi/4\}$
- **offset** d: try offsets $d \in \{0, 2\text{cm}, 4\text{cm}\}$

Grasps are simulated by approaching a point on an object along the normal direction and closing the fingers. Once the fingers close, we extract scores for three of the grasp stability metrics recently analyzed by [23]:
ξ_1 – Grasp Isotropy Index [24];
ξ_2 – Distance from Center of Mass [25, 26, 27, 28];
ξ_3 – Grasp Finger Posture [29, 30, 31].

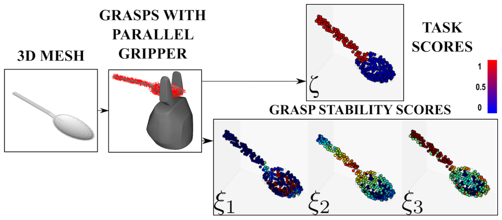


Figure 3: Data generation for *support* on spoon (with $\psi = 0$, $d = 0$). For task scores, red area denotes points suitable for grasping.

4 Experiments

4.1 Performance of BO Variants on Synthetic Benchmarks

To anticipate the challenges of running optimization on a real-world robotics system, we first test the performance of BO variants on synthetic benchmarks. From the commonly used optimization test functions [32] we select 3 settings which exemplify the main challenges that frequently occur in robotics: numerous shallow local optima, small low-cost region and sharp cost function drops/rises, deep local optima difficult to overcome when searching for global optimum. Figure 4 visualizes the settings we consider. Figure 4a shows the *Ackley* function with numerous shallow local minima, defined as:

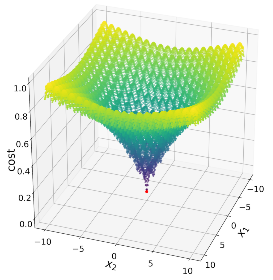
$$f_{AC}(x) = \text{-}a \cdot \exp\left(\text{-}b\sqrt{\tfrac{1}{d}\textstyle\sum_{i=1}^{d} x_i^2}\right) - \exp\left(\tfrac{1}{d}\textstyle\sum_{i=1}^{d}\cos(cx_i)\right) + a + \exp(1)$$
$$a = 20, b = 0.2, c = 2\pi, x \in \mathbb{R}^d$$

For a further challenge we consider this function on a much larger domain $[-100, 100]$. This is similar to considering Easom function, which is commonly used to test robustness to steep ridges/drops in the search space. Figure 4b shows the Ackley function on this larger domain.

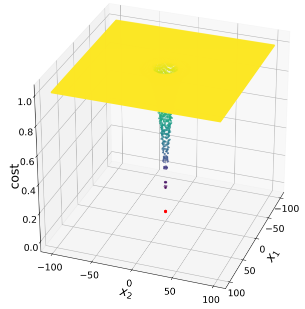
We are interested in emulating a setting where simulation could provide coarse guidance, but also could be limiting when searching for the global optimum. For the synthetic setting we pick ϕ with components that capture information from two first addends of the Ackley function:

$$[\phi(x)]_1 = \text{-}b\sqrt{\tfrac{1}{d}\textstyle\sum_{i=1}^{d} x_i^2} \quad ; \quad [\phi(x)]_2 = \tfrac{1}{d}\textstyle\sum_{i=1}^{d}\cos(cx_i)$$

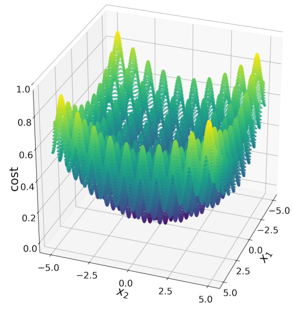
This gives a kind of ‘collapsing’ of the relevant features, akin to simulation that aims to capture most salient features needed to approximate real-world output.



(a) Ackley function with challenging local minima



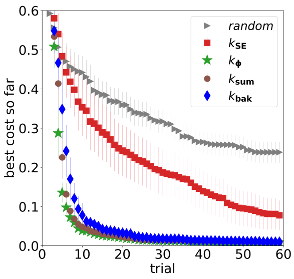
(b) Ackley on a larger domain: only 1% of the space is <0.9



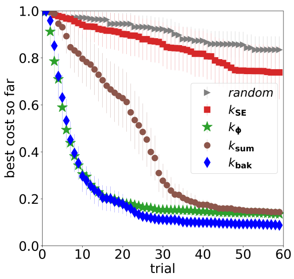
(c) Rastrigin: highly multi-modal, deep local minima

Figure 4: High-dimensional analytic functions for testing optimization algorithms. Ackley and Rastrigin functions present challenges similar to those most frequent in optimization for robotics: (a) shallow local optima, (b) small low cost region & sharp cost changes, (c) deep local optima difficult to overcome. Figures show functions in 2D, with values (costs) normalized to $[0, 1]$. We test on 2- to 10-dimensional versions of these.

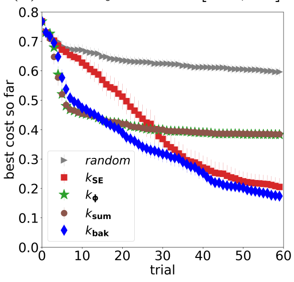
Figure 5 shows comparisons of BO variants on 2D and 10D versions of the Ackley function. In 2D, using ϕ gives a significant advantage to all the informed versions of BO (with k_{ϕ}, k_{sum} and k_{bak} kernels). When the low-cost region is small, the gains are even more striking. In this case the performance of uninformed BO with SE kernel degrades to random search, while the informed versions get close to the optimum in less than 40 trials. However, the hint of limitations induced by using ϕ is already visible: in the case of k_{ϕ} and k_{sum} the improvement stagnates after 50 trials. This stagnation is even more striking when 10-dimensional version of the Ackley function is optimized. There, even the uninformed BO with SE kernel improves over k_{ϕ} and k_{sum} after 30 trials. In contrast, k_{bak} is able to 'recover' and outperform both informed and uninformed kernels.



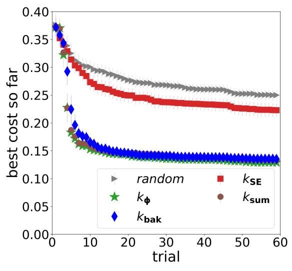
(a) Ackley 2D on [-10, 10]



(b) Ackley 2D on [-100, 100]



(c) Ackley 10D on [-10, 10]



(d) Rastrigin 10D on [-5, 5]

Figure 5: BO on analytic functions. Plots of mean over 40 runs for each kernel type, 95% CIs. Comparison with random search ('random' in the legend) shows relative difficulty of each setting.

To test robustness to deep local minima we use *Rastrigin* function:

$$f_{RA}(x) = \textstyle\sum_i^d x_i^2 - \sum_i^d a\cos(c\pi x_i) + a \cdot d;\;\; a{=}10, c{=}2, x{\in}\mathbb{R}^d$$

Figure 4c shows a 2D version of the function. As before, ϕ summarizes the first two addends:

$$[\phi(x)]_1 = \textstyle\sum_i^d x_i^2;\;\; [\phi(x)]_2 = \sum_i^d a\cos(c\pi x_i)$$

Figure 5d shows results for the BO variants in 10D. In this case, all informed kernels improve over uninformed SE kernel. Overall, the results on test benchmarks appear promising. Therefore we proceed with experiments on hardware, since we believe this step is crucial for validating applicability of our approach to robotics problems.

4.2 Experimental Setup for Task-oriented Grasping

To evaluate performance on a real-world setting we construct Everyday Objects Dataset (EOD), then do task-oriented grasping with an ABB Yumi robot. The objects (shown in Figure 7) are from seven categories and can be used for six different tasks. Our selection of objects is constrained by limitations of the robot: maximum payload of 500g (lower in practice), rigid plastic parallel gripper, no force-torque or tactile sensing. Once an object is placed on the table, we use Microsoft Kinect to get a dense point cloud of the scene and segment out the object. We then generate a mesh representation from the partial point cloud and attach a coordinate frame to the object[2]. Then we compute grasp points, discarding those in collision with the table. The object is voxelized and together with grasp and task representation fed through the CNN to get stability and task score estimates.

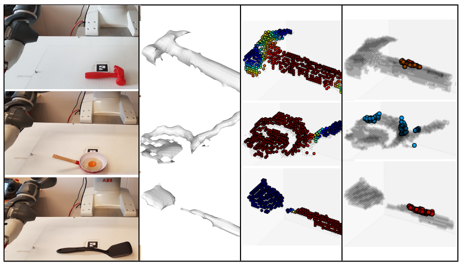


Figure 6: Three objects from EOD and visualization of task & stability scores. From left to right: object in the workspace, partial mesh, ζ task scores, ξ_2 stability metric for top 100 grasps with $\zeta{>}0.5$.

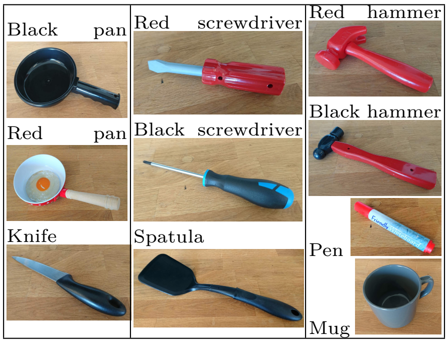


Figure 7: Ten objects from our EOD (Everyday Objects Dataset).

[2]Object coordinate frame is positioned in the center of the mesh bounding box and has the same orientation in the world coordinate frame as the april tag which we use to segment the object, for details see [33].

We execute grasps on the robot and report the results using the following scores:

[**0.0-0.99**] **No plan**: planning failed or returned partial plan

[**1.0**] **No lift**: failed to grasp the object

[**2.0**] **Lift failed**: robot grasped the object but failed to lift it (object slipped)

[**3.0**] **Wrong grasp**: grasped a wrong part of object (unsuitable for the task)

[**4.0**] **Success**: task-appropriate grasp was successful

After each execution, operator re-positions the object in the workspace if needed and starts next trial.

Before starting experiments on hardware, for an initial test of our CNN we evaluated its performance on a synthetic test set. For task suitability: we report task-specific and overall MaxF1 and MAP scores in Table E.1 (top). Overall, the network learns to recognize parts of objects that are suitable for grasping given a task. The most challenging task is *cut*, since blade and handle parts of a knife can have very similar appearance. For grasp stability: in Table E.1 (bottom) we report RMSE for each of the stability metrics. Since we are interested in using the predicted scores for task-oriented grasping, we also report precision at top 100 for task-appropriate points ($\zeta > 0.5$).

Task	**maxF1**	**MAP**
handover	0.892	0.930
screw	0.988	0.990
cut	0.764	0.877
pour	0.874	0.963
support	0.952	0.964
pound	0.940	0.948
overall	0.909	0.943

Stability scores	ξ_1	ξ_2	ξ_3
RMSE	0.184	0.173	0.206
Precision@100	0.651	0.902	0.688

Table E.1: CNN evaluation on synthetic testset.

4.3 Hardware Experiments for Top-k Grasps

To test the performance of our CNN in a real-world setting, we did task-oriented grasping for objects in our EOD. We generated 4500 grasps (as in Section 3.3), then removed those in collision with the table. For each object-task pair, 10 grasps with highest stability according to score ξ_2 (and task scores $\zeta > 0.5$) were executed on the robot. We did not perform *handover* for screwdrivers and knife, because their blades were not adequately visible to our camera. Furthermore, although some high-scoring points were found on the spatula for *handover*, they were out of reach for the robot. For all other object-task pairs multiple successful task-oriented grasps were found.

Task	Scores (for each trial)
	Spatula
support	4 4 4 4 4 2 4 4 4 1
	White pan
handover	4 4 4 4 4 4 3 3 4 3
support	4 4 4 4 4 1 4 1 4 4
	Black pan
handover	1 2 2 1 4 2 4 4 4 4
support	1 1 1 1 4 4 1 2 1 4
	Red hammer
handover	1 2 4 4 2 4 2 4 4 1
pound	4 4 4 4 4 1 4 1 4 4

Table E.2: Top-10 grasps on EOD.

Tables E.2&E.3 report scores for each trial (scoring described in Section 4.2). Our CNN mostly outputs high scores for grasp points that are on suitable parts of the object, both from task and stability perspectives. Of course not all of these grasps succeed in practice, since the CNN is trained using simulations that do not model friction or mass distribution. In reality, the most stable grasps were on spatula for *support*, white pan for both tasks and red hammer for task *pound*. The most unstable grasps where on mug *handover*, likely because of smooth surface, high mass, severely degraded partial mesh from vision.

Task	Scores (for each trial)
	Black hammer
handover	4 4 4 4 2 4 4 1 1 2
pound	4 4 4 2 4 4 1 1 1 4
	Mug
handover	2 2 4 2 4 2 2 4 2 2
pour	4 4 2 1 4 3 4 4 4 4
	Red screwdriver
screw	4 4 4 1 4 4 2 4 4 2
	Black screwdriver
screw	2 2 4 2 1 4 2 2 2 2
	Knife
cut	1 2 4 2 4 2 2 4 2 4
	Pen
handover	4 2 4 2 4 2 4 4 2 2

Table E.3: Top-10 grasps on EOD

For the above experiments, we reduced maximum gripper standoff from 4cm to 3cm. Since the tabletop was not present during training, CNN was likely to learn high scores for grasps with small standoff. These would frequently collide with the table. The alternative grasps with 4cm standoff were still successful in simulation, but slipped off the object in reality. While we could fix this issue by simply reducing the maximum standoff in this case, such simulation-reality mismatch might be harder to fix in general. This motivates experiments with adaptive search like BO, which we describe next.

4.4 Hardware Experiments for Bayesian Optimization

We ran experiments using the proposed BO approach from Section 3.1. k_ϕ kernel was constructed using the stability scores: $\phi(\boldsymbol{x}) = [\xi_1, \xi_2, \xi_3]$, which were obtained from the trained CNN. The vector of control parameters $\boldsymbol{x} = (\boldsymbol{p}, \boldsymbol{n}, \psi, d)$ contained: 3D coordinates for a point on the object, approach direction, gripper roll and offset (as in Section 3.2). Choice for $\boldsymbol{p}$ was constrained to the surface of the object, but other control parameters were limited only by a choice of [min,max] values. Points on the object were sampled from the output of the vision system, those with low task scores were filtered out (for a given task) using the trained CNN. Our implementation of BO was based on [34, 35]. After each BO trial, $f(\boldsymbol{x})$ evaluation (expressing the success of executing grasp with parameters $\boldsymbol{x}$) was given as described in Section 4.2.

BO usually starts with a random trial, but in our experiments we instead execute one top choice from each of the 3 stability metrics. For challenging objects this does not yield any promising points. However, from this BO can infer which regions are not promising. *Handover* task for spatula provides a clear example of this. Left side of Figure 8 shows executing top choice according to each of the 3 stability metrics (after filtering out planning failures). These choices are not successful, moreover executing top 20 choices does not yield any successful grasps. Most stable grasps are not reachable due to robot's joint limits or the need to approach too close to the table. We obtain a successful handover on the 4th trial (1st candidate from BO), shown in the right part of Figure 8. The grasp is just above the handle part (the handle needs to be clear for handover). It is labeled as acceptable for task completion, but is unlikely to be among top k offline choices. This is because task and stability scores are higher in other parts of the object, but those parts are inaccessible.

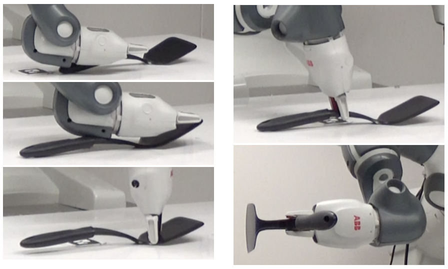


Figure 8: Left: top choices for handover task from 3 stability metrics; Right: subsequent BO trial.

Left side of Figure 9 shows success of BO on the 4th trial for mug handover (1st candidate from BO). Right side shows successful pan handover on the 5th trial (2nd candidate from BO). In contrast, top k grasps computed offline attempted to grasp the middle and outer part of the handle, which for this pan resulted in either slippage or tilting. The above object-task pairs presented the toughest challenges for the top-10 approach, while BO proposed successful task-oriented grasps in the first few trials.

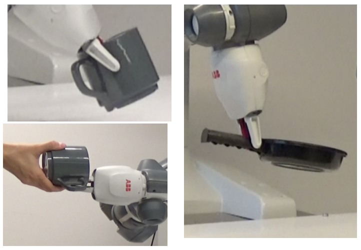


Figure 9: Left: A trial from BO for mug handover task. Right: A trial from BO for pan support task.

Overall, we did 15 runs with BO: 6 full runs with 10 trials each, 9 partial runs that we stopped early after 5 trials. We stopped a run early if the top 3 choices from CNN already suggested successful approach points, or if multiple successful grasps were executed in the first 5 trials. We obtained multiple successful task-oriented grasps for spatula, mug, hammers, pans, screwdrivers. We could not complete BO on screwdriver handover task, knife and pen, because these required getting very close to the surface of the table (within 1-3mm). BO repeatedly proposed such approaches, but the planning library rejected these as near-collisions. For the full BO runs we focused on object-task pairs that were challenging for top-10 approach (e.g. ran spatula *handover* twice to ensure repeated success, since top-10 approach failed for this task). Figure 10 summarizes the results, showing quick significant improvement of BO over the initial top choices from the 3 stability metrics. Qualitatively, the benefits we observed from using BO were: 1) exploring various parts of the object systematically and efficiently; 2) sampling a variety of successful controllers that further improve over a merely acceptable controller.

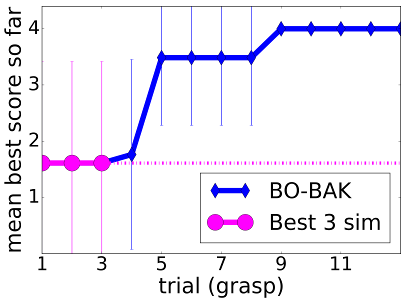


Figure 10: BO for challenging objects.

5 Conclusion and Future Work

We proposed a variant of online global search suitable for simulation-informed optimization. Our focus was on validating this approach on a challenging robotics task. We plan to extend evaluation to task-oriented grasping scenarios in clutter, external disturbances, or real-time requirements for task completion. Our data generation does not assume a tabletop scenario, since objects are simulated without a support surface. This is a strength, since there is no need to re-run offline training when the workspace properties change. However, we need to improve our planning pipeline to avoid needless planning failures when grasping very small objects from tabletop. On the theory side, it would be interesting to explore theoretical properties of the proposed informed BO, investigate which guarantees could be obtained. It would be useful to put an emphasis on retaining realistic assumptions: no bounds on simulation-reality mismatch a-priori, non-smooth objective/cost functions.

Acknowledgments

This research was supported in part by the Knut and Alice Wallenberg Foundation.

References

[1] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas. Taking the Human Out of the Loop: A Review of Bayesian Optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

[2] A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret. Robots that can adapt like animals. *Nature*, 521(7553):503–507, 2015.

[3] A. Rai, R. Antonova, S. Song, W. Martin, H. Geyer, and C. G. Atkeson. Bayesian Optimization Using Domain Knowledge on the ATRIAS Biped. In *Robotics and Automation (ICRA), 2018 IEEE International Conference on*, 2018.

[4] A. Marco, P. Hennig, S. Schaal, and S. Trimpe. On the design of LQR kernels for efficient controller learning. In *56th IEEE Annual Conference on Decision and Control, CDC*, 2017.

[5] R. Antonova, A. Rai, and C. G. Atkeson. Deep kernels for optimizing locomotion controllers. In *Conference on Robot Learning*, pages 47–56, 2017.

[6] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005. ISBN 026218253X.

[7] A. Rai, R. Antonova, F. Meier, and C. G. Atkeson. Using simulation to improve sample-efficiency of bayesian optimization for bipedal robots. *arXiv preprint arXiv:1805.02732*, 2018.

[8] R. Diankov and J. Kuffner. Openrave: A planning architecture for autonomous robotics. *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-08-34*, 79, 2008.

[9] I. Lenz, H. Lee, and A. Saxena. Deep learning for detecting robotic grasps. *The International Journal of Robotics Research*, 34(4-5):705–724, 2015.

[10] J. Mahler, J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, J. A. Ojea, and K. Goldberg. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. *arXiv preprint arXiv:1703.09312*, 2017.

[11] P. Schmidt, N. Vahrenkamp, M. Wächter, and T. Asfour. Grasping of unknown objects using deep convolutional neural networks based on depth images. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[12] O. Kroemer, R. Detry, J. Piater, and J. Peters. Combining active learning and reactive control for robot grasping. *Robotics and Autonomous systems*, 58(9):1105–1116, 2010.

[13] L. Montesano and M. Lopes. Active learning of visual descriptors for grasping using non-parametric smoothed beta distributions. *Robotics and Autonomous Systems*, 60 (3):452–462, 2012.

[14] J. Oberlin and S. Tellex. Autonomously acquiring instance-based object models from experience. In *Robotics Research*, pages 73–90. Springer, 2018.

[15] J. Mahler, F. T. Pokorny, B. Hou, M. Roderick, M. Laskey, M. Aubry, K. Kohlhoff, T. Kröger, J. Kuffner, and K. Goldberg. Dex-net 1.0: A cloud-based network of 3d objects for robust grasp planning using a multi-armed bandit model with correlated rewards. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 1957–1964. IEEE, 2016.

[16] D. Song, C. H. Ek, K. Huebner, and D. Kragic. Task-based robot grasp planning using probabilistic inference. *IEEE transactions on robotics*, 31(3):546–561, 2015.

[17] L. Antanas, P. Moreno, M. Neumann, R. P. de Figueiredo, K. Kersting, J. Santos-Victor, and L. De Raedt. High-level reasoning and low-level learning for grasping: A probabilistic logic pipeline. *arXiv preprint arXiv:1411.1108*, 2014.

[18] M. Kokic, J. A. Stork, J. A. Haustein, and D. Kragic. Affordance detection for task-specific grasping using deep learning. In *Humanoid Robotics (Humanoids), 2017 IEEE-RAS 17th International Conference on*, pages 91–98. IEEE, 2017.

[19] R. Detry, J. Papon, and L. Matthies. Taskoriented grasping with semantic and geometric scene understanding. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2017.

[20] K. Fang, Y. Zhu, A. Garg, A. Kurenkov, V. Mehta, L. Fei-Fei, and S. Savarese. Learning task-oriented grasping for tool manipulation from simulated self-supervision. *arXiv preprint arXiv:1806.09266*, 2018.

[21] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

[22] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015.

[23] C. Rubert, D. Kappler, A. Morales, S. Schaal, and J. Bohg. On the relevance of grasp metrics for predicting grasp success. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 265–272. IEEE, 2017.

[24] B.-H. Kim, S.-R. Oh, B.-J. Yi, and I. H. Suh. Optimal grasping based on non-dimensionalized performance indices. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 2, pages 949–956. IEEE, 2001.

[25] E. Chinellato, A. Morales, R. B. Fisher, and A. P. Del Pobil. Visual quality measures for characterizing planar robot grasps. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 35(1):30–41, 2005.

[26] J. Ponce and B. Faverjon. On computing three-finger force-closure grasps of polygonal objects. *IEEE Transactions on robotics and automation*, 11(6):868–881, 1995.

[27] J. Ponce, S. Sullivan, A. Sudsang, J.-D. Boissonnat, and J.-P. Merlet. On computing four-finger equilibrium and force-closure grasps of polyhedral objects. *The International Journal of Robotics Research*, 16(1):11–35, 1997.

[28] D. Ding, Y.-H. Lee, and S. Wang. Computation of 3-d form-closure grasps. *IEEE Transactions on Robotics and Automation*, 17(4):515–522, 2001.

[29] J. Cornella and R. Suárez. Fast and flexible determination of force-closure independent regions to grasp polygonal objects. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 766–771. IEEE, 2005.

[30] Y.-H. Liu. Computing n-finger form-closure grasps on polygonal objects. *The International journal of robotics research*, 19(2):149–158, 2000.

[31] Y. Li, Y. Yu, and S. Tsujio. An analytical grasp planning on given object with multifingered hand. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 4, pages 3749–3754. IEEE, 2002.

[32] S. Surjanovic and D. Bingham. Virtual library of simulation experiments: test functions and datasets. *Simon Fraser University, Burnaby, BC, Canada*, 13:2015, 2013.

[33] J. Varley, C. DeChant, A. Richardson, J. Ruales, and P. Allen. Shape completion enabled robotic grasping. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 2442–2447. IEEE, 2017.

[34] C. E. Rasmussen and H. Nickisch. Gaussian processes for machine learning (gpml) toolbox. *J. Mach. Learn. Res.*, 11:3011–3015, Dec. 2010.

[35] J. R. Gardner, M. J. Kusner, Z. E. Xu, K. Q. Weinberger, and J. Cunningham. Bayesian Optimization with Inequality Constraints. In *ICML*, pages 937–945, 2014.

Variational Auto-Regularized Alignment for Sim-to-Real Control

Martin Hwasser, **Danica Kragic**, **Rika Antonova**

Northvolt, Sweden EECS, KTH, Stockholm, Sweden

Abstract

General-purpose simulators can be a valuable data source for flexible learning and control approaches. However, training models or control policies in simulation and then directly applying to hardware can yield brittle control. Instead, we propose a novel way to use simulators as regularizers. Our approach regularizes a decoder of a variational autoencoder to a black-box simulation, with the latent space bound to a subset of simulator parameters. This enables successful encoder training from a small number of real-world trajectories (10 in our experiments), yielding a latent space with simulation parameter distribution that matches the real-world setting. We use a learnable mixture for the latent prior/posterior, which implies a highly flexible class of densities for the posterior fit. Our approach is scalable and does not require restrictive distributional assumptions. We demonstrate ability to recover matching parameter distributions on a range of benchmarks, challenging custom simulation environments and several real-world scenarios. Our experiments using ABB YuMi robot hardware show ability to help reinforcement learning approaches overcome cases of severe sim-to-real mismatch.

Published in the proceedings of the 2020 IEEE International Conference on Robotics and Automation (ICRA). Available at https://ieeexplore.ieee.org/document/9197130

Analytic Manifold Learning: Unifying and Evaluating Representations for Continuous Control

Rika Antonova[‡] **Maksim Maydanskiy**
Danica Kragic[‡] **Sam Devlin**[§] **Katja Hofmann**[§]
[‡]EECS, KTH, Stockholm, Sweden
[§]Microsoft Research

Abstract

We address the problem of learning reusable state representations from streaming high-dimensional observations. This is important for areas like Reinforcement Learning (RL), which yields non-stationary data distributions during training. We make two key contributions. First, we propose an evaluation suite that measures alignment between latent and true low-dimensional states. We benchmark several widely used unsupervised learning approaches. This uncovers the strengths and limitations of existing approaches that impose additional constraints/objectives on the latent space.

Our second contribution is a unifying mathematical formulation for learning latent relations. We learn analytic relations on source domains, then use these relations to help structure the latent space when learning on target domains. This formulation enables a more general, flexible and principled way of shaping the latent space. It formalizes the notion of learning independent relations, without imposing restrictive simplifying assumptions or requiring domain-specific information. We present mathematical properties, concrete algorithms for implementation and experimental validation of successful learning and transfer of latent relations.

1 Introduction

In this work, we address the problem of learning reusable state representations from streaming high-dimensional observations. Consider the case when a deep reinforcement learning (RL) algorithm is trained on a set of source domains. Low-dimensional state representations could be extracted from intermediate layers of

https://arxiv.org/abs/2006.08718

RL networks, but they might not be reusable on a target domain with different rewards or dynamics. To aid transfer and ensure non-degenerate embeddings, it is common to add unsupervised learning objectives. However, the quality of resulting representations is usually not evaluated rigorously. Moreover, constructing and prioritizing such objectives is done manually: auxiliary losses are picked heuristically and hand-tuned for transfer to a new set of domains or tasks.

As the first part of our contribution, we provide a set of tools and environments to improve evaluation of learning representations for use in continuous control. We evaluate commonly used unsupervised approaches and explain new insights that highlight the need for critical analysis of existing approaches. Our evaluation suite provides tools to measure alignment between the latent state from unsupervised learners and the true low-dimensional state from the physics simulator. Furthermore, we introduce new environments for manipulation with multiple objects and ability to vary their complexity: from geometric shapes to mesh scans and visualizations of real objects. We show that, while alignment with true state is achieved on the simpler benchmarks, new environments present a formidable challenge: existing unsupervised objectives do not guarantee robust and transferable state representation learning.

The second part of our contribution is a formalization of learning latent objectives from a set of source domains. We describe the mathematical perspective of this approach as finding a set of functionally independent relations that hold for the data sub-manifold. We explain theoretical properties and guarantees that this perspective offers. Previous work constructed latent relations based on domain knowledge or algorithmic insights, e.g. using continuity [1], mutual information with prior states [2], consistency with a forward or inverse model (see [3] for a survey). Our formulation offers a unified view, allowing to leverage known relations, discover new ones and incorporate relations into joint training for transfer to target domains. We describe algorithms for concrete implementation and visualize the learned relations on analytic and physics-based domains. In our final set of experiments, we show successful transfer of relations learned from source domains with simple geometric shapes to target domains that contain objects with real textures and 3D scanned meshes. We also show that our approach obtains improved latent space encoder mappings with smaller distortion variability.

2 Evaluation Suite for Unsupervised Learning for Continuous Control

Reinforcement learning (RL) has shown strong progress recently [4], and RL for continuous control is particularly promising for robotics [5]. However, training for each robotics task from scratch is prohibitively expensive, especially for high-dimensional observations. Unsupervised learning could help obtaining low-dimensional latent representations, e.g. with variational autoencoder (VAE) [6] variants. However, evaluation of these mostly focused on datasets, with a limiting assumption that the training data distribution is stationary [7]. Moreover, advanced approaches usually

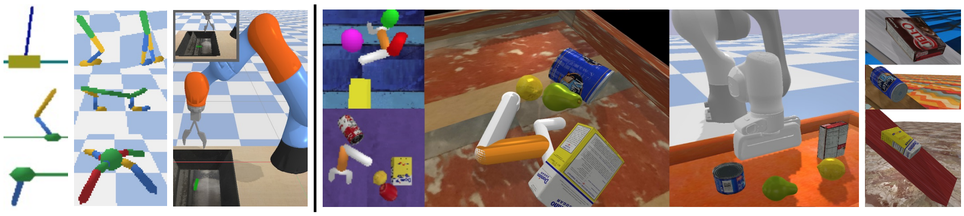


Figure 1: Evaluation suite environments. Left: Standard PyBullet envs for which our suite yields both pixels and low-dimensional state. Right: Proposed domains with YCB objects.

report best-case results, achieved only with exact parameters that the authors find to work for a given static dataset. Obtaining reconstructions that are clear enough to judge whether all important information is encoded in the latent state could still require days or weeks of training [8, 9]. These limitations severely impair the adoption of unsupervised representation learning in robotics. In stark contrast to the learning community, a vast majority of roboticists still need to rely on hand-crafted low-dimensional features.

We propose an evaluation suite that helps analyze the alignment between the learned latent state and true low-dimensional state. Unsupervised approaches receive frames that an RL policy yields during its own training: a non-stationary stream of RGB images. The alignment of the learned latent state and the true state is measured periodically as training proceeds. For this, we do a regression fit using a small fully-connected neural network, which takes latents as inputs and is trained to produce low-dimensional states as outputs (position & orientation of objects; robot joint angles, velocities, contacts). The quality of alignment is characterized by the resulting test error rate. This approach helps quantify latent space quality without the need for detailed reconstructions. To connect our suite to existing benchmarks, we extend the OpenAI gym interface [10] of widely used robotics domains so that both pixel- and low-dimensional state is reported. We use an open source simulator: PyBullet [11]. Simulation environments are parallelized, ensuring scalability. We introduce advanced domains utilizing meshes from 3D scans of real objects from the YCB dataset [12]. This yields realistic object appearances and dynamics. Our *RearrangeYCB* domain models object rearrangement tasks, with variants for using realistic vs basic robot arms. The *RearrangeGeom* domain offers an option with simple geometric shapes instead of object scans. The *YCB-on-incline* domain models objects sliding down an incline, with options to change friction and apply external forces; *Geom-on-incline* offers a variant with simple single-color geometric shapes. Figure 1 gives an overview.

2.1 Benchmarking Latent State Alignment of Unsupervised Approaches

To demonstrate usage and benefits of the suite we evaluated several widely used and recently proposed unsupervised learning approaches. Unsupervised approaches get 64x64 pixel images sampled from replay buffers, filled by PPO RL learners [13].

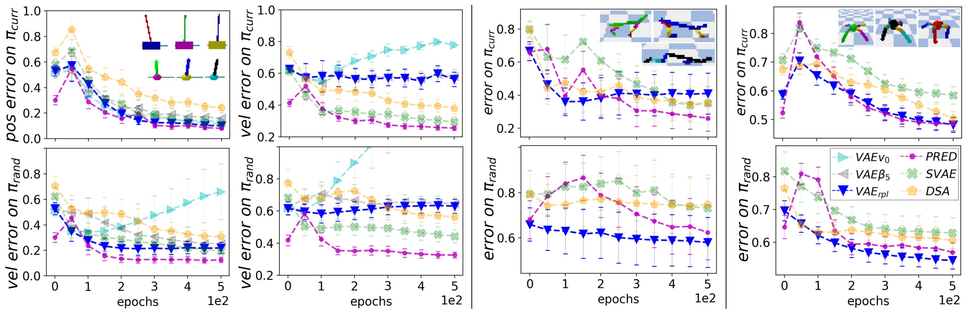


Figure 2: Benchmarking alignment with true low-dimensional state. Plots show mean test error of NN regressors trained with current latent codes as inputs and true states (robot positions, velocities, contacts) as outputs. 90% confidence intervals over 6 training runs for each unsupervised approach are shown (>140 training runs overall). Training uses frames from replay buffers (1024 frames per batch; 10 batches per epoch, 50 for locomotion). Top row: performance on frames from current RL policy π_{curr}, middle row: random policy π_{rand}. 1st column shows results for *CartPole* and *InvertedPendulum* for position & angle; 2nd column: for velocity. 3rd column shows aggregated results for position, velocity and contacts for *HalfCheetah*; 4th column shows these for *Ant* domain.

Figures 2, 3 show results for the following unsupervised approaches (Appendix A gives more detailed descriptions and learning parameters): ***VAE$_{v_0}$*** [6]: a VAE with a 4-layer convolutional encoder and corresponding de-convolutional decoder; ***VAE$_{rpl}$***: a VAE with a replay buffer that retains 50% of frames from beginning of training (our modification of VAE for improved performance on a wider range of RL policies); ***β-VAE*** [14]: a VAE with β parameter to encourage disentanglement (we tried several β parameters and also included the replay enhancement from *VAE$_{rpl}$*); ***SVAE***: a sequential VAE that reconstructs a sequence of frames $x_1, ..., x_t$; ***PRED***: a VAE that, given a sequence of frames $x_1, ..., x_t$, constructs a predictive sequence $x_1, ..., x_{t+k}$; ***DSA*** [15]: a sequential autoencoder that uses structured variational inference to encourage separation of static and dynamic aspects of the latent state; ***SPAIR*** [16]: a spatially invariant and faster version of AIR [17] that imposes a particular structure on the latent state.

Figure 2 shows results on multicolor versions of *CartPole*, *InvertedPendulum*, *HalfCheetah* and *Ant* domains (multicolor to avoid learning trivial color-based features). We evaluated using two kinds of policies: a current RL learner policy π_{curr}, and a random policy π_{rand}. Success on π_{rand} is needed for transfer: when learning a new task, initial frames are more similar to those from a random policy than a final source task policy. *VAE$_{v_0}$* performed poorly on π_{rand}. We discovered that this can be alleviated by replaying frames from initial random policy. The resulting *VAE$_{rpl}$* offers good alignment for positions. Surprisingly, *β-VAE* offered no improvement over *VAE$_{rpl}$*. We used $\beta \in \{100, 20, 10, 5, 0.5\}$; the best ($\beta = 5$) performed slightly worse than *VAE$_{rpl}$* on pendulum domains (shown in Figure 2), the rest did significantly worse (omitted from plots). Sequential approaches *SVAE*,*PRED*,*DSA* offered significant gains when measuring alignment for velocity. Despite its simpler

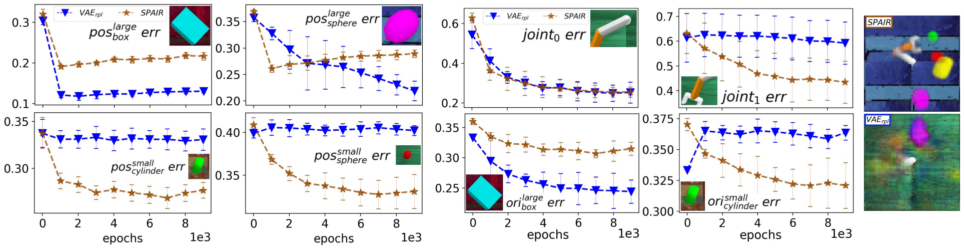


Figure 3: Evaluation on the *RearrangeGeom* domain (reconstructing YCB objects was difficult for existing approaches, so *RearrangeYCB* was too challenging). VAE_{rpl} encoded angle of the main robot joint, location & partly orientation (major axis) of the largest objects. *SPAIR* encoded (rough) locations quickly, but did not improve with longer training (bounding boxes not tight).

architecture, *PRED* performed best on pendulum domains. For aggregated performance on position, velocity and contacts (i.e. whether robot joints touch the ground) for locomotion: PRED outperformed VAE_{rpl} on π_{curr}, but was second-best on π_{rand}. Overall, this set of experiments was illuminating: simpler approaches were often better than more advanced ones.

For our newly proposed domains with multiple objects: the first surprising result was that all of the approaches we tested failed to achieve clear reconstructions of objects from the YCB dataset. This was despite attempts to use larger architectures, up to 8 layers with skip connections, similar to [18]. Figure 3 shows results for *SPAIR* vs VAE_{rpl}. *SPAIR* succeeded to reconstruct *RearrangeGeom*, while other approaches failed. This indicates that our multi-object benchmark is a highly needed addition to the current continuous control benchmark pool. While single-object benchmarks might still be challenging for control, they could be inherently simpler for latent state learning and reconstruction.

Overall, our analysis shows that structuring the latent space can be beneficial, but has to be done such that it does not impair the learning process and resulting representations. This is not trivial, since seemingly beneficial objectives that worked well in the past could be detrimental on new domains. However, forgoing structure completely can fail on more advanced scenes. Hence, in the following sections we show an alternative direction: a principled way to learn a set of general rules from source domains, then apply them to structure latent space of unsupervised learners on target domains.

3 Analytic Manifold Learning

We now motivate the need to unify learning latent relations, then provide a rigorous and general mathematical formulation for this problem. Let x_t denote a high-dimensional (observable) state at time t and s_t denote the corresponding low-dimensional or latent state. x_t could be an RGB image of a scene with a robot & objects, while s_t could contain robot joint angles, object poses, and velocities.

Consider an example of a latent relation: the continuity (slowness) principle [1, 19]. It postulates continuity in the latent states, implying that sudden changes are unlikely. It imposes a loss $L_{cont}(\mathcal{D}_x, \phi) = \mathbb{E}\big[||s_{t+1} - s_t||^2\big]$, with $D_x = \{x_t, x_{t+1}, ...\}$ and encoder $\phi(x) = s$. A related heuristic from [2] maximizes mutual information between parts of consecutive latent states. Such approaches may be viewed as postulating concrete latent relations: $g(s_t, s_{t+1}) = c_\epsilon$, where g is the squared distance between s_t and s_{t+1} for L_{cont}, and a more complicated relation for [2]. Ultimately, all these are heuristics coming from intuition or prior knowledge. However, only a subset of them might hold for a given class of domains. Moreover, it would be tedious and error-prone to manually compose and incorporate a comprehensive set of such heuristics into the overall optimization process.

We take a broader perspective. Let $g(\mathcal{D}_\tau) = 0$ define a relation that holds on a set of sequences $\mathcal{D}_\tau = \{\tau^{(i)}\}_{i=1}^{M}$. $\mathcal{D}_\tau$ could contain state sequences $\tau = [s_t, ..., s_{t+T}]$ from a set of source domains. We start by learning a relation g_1; then learn g_2 that differs from g_1; then learn g_3 different from $\{g_1, g_2\}$ and so on. Overall, we aim to learn a set of relations that are (approximately) independent, and we define independence rigorously. To understand why rigor is important here, recall the significance of the definition of independence in linear algebra: it is central to the theory and algorithms in that field. Extending the notion of independence to our more general nonlinear setting is not trivial, since naive definitions can yield unusable results. Our contribution is developing rigorous definitions of independence, and ensuring the result can be analyzed theoretically & used for practical algorithms.

3.1 Mathematical Formulation

Let $\mathbb{R}^N$ be the ambient space of all possible latent state sequences τ (of some fixed length). Let $\mathcal{M}$ be the submanifold of actual state sequences that a dynamical system from one of our domains could generate (under any control policy). A common view of discovering $\mathcal{M}$ is to learn a mapping that produces only plausible sequences as output (the 'mapping' view). Alternatively, a submanifold can be specified by describing all equations (i.e. relations) that have to hold for points in the submanifold.

We are interested in finding relations that are in some sense independent. In linear algebra, a dependency is a linear combination of vectors with constant coefficients. In our nonlinear setting the analogous notion is that of *syzygy.* A collection of functions $\mathfrak{f}^\ddagger = \{f_1, ..., f_k\}$ is called a *syzygy* if $\sum_{j=0}^{k} f_j g_j$ is zero. Observe that this sum is a linear combination of relations $g_1, ..., g_k$ with coefficients in the ring of functions. If there is no syzygy $\mathfrak{f}^\ddagger$ s.t. $\sum_{j=0}^{k} f_j g_j = 0$, then $g_1, ..., g_k$ are independent. However, this notion of independence is too general for our case, since it deems any g_1, g_2 dependent: $g_1 \cdot g_2 - g_2 \cdot g_1 = 0$ holds for any g_1, g_2. Hence, we define *restricted syzygies.*

Definition 3.1 (Restricted Syzygy). *Restricted syzygy for relations $g_1, ..., g_k$ is a syzygy with the last entry f_k equal to -1, i.e. $\mathfrak{f} = \{f_1, ..., f_{k-1}, f_k = -1\}$ with $\sum_{j=1}^{k} f_j g_j = 0$.*

Definition 3.2 (Restricted Independence). *g_k is independent from $g_1, ..., g_{k-1}$ in a restricted sense if the equality $\sum_{j=1}^{k} f_j g_j = 0$ implies $f_k \neq -1$, i.e. if there exists no restricted syzygy for $g_1, ..., g_k$.*

For $\mathfrak{f} = \{f_1, ..., f_{k-1}, f_k = -1\}$ we denote $\sum_{j=1}^{k} f_j(\tau) g_j(\tau)$ by $\mathfrak{f}(\tau, g_1, ..., g_k)$. Using the above definitions, we construct a practical algorithm (Section 3.2) for learning independent relations. The overall idea is: while learning g_ks, we are also looking for restricted syzygies $\mathfrak{f}(\tau, g_1, ..., g_k) = 0$. Finding them would mean g_ks are dependent, so we augment the loss for learning g_k to push it away from being dependent. We proceed sequentially: first learning g_1, then g_2 while ensuring no restricted syzygies appear for $\{g_1, g_2\}$, then learning g_3 and so on. Section 5 explains motivations for learning sequentially. For training g_ks we use *on-manifold* data: τ sequences from our dynamical system. Restricted syzygies $\mathfrak{f}$ are trained using *off-manifold* data: $\tau_{off} = \{s_{off_t}, s_{off_{t+1}}, ..., s_{off_T}\}$, because we aim for independence of g_ks on $\mathbb{R}^N$, not restricted to $\mathcal{M}$ (on $\mathcal{M}$ g_ks should be zero). τ_{off} do not lie on our data submanifold and can come from thickening of on-manifold data or can be random (when $\mathbb{R}^N$ is large, the probability a random sequence satisfies equations of motion is insignificant). Independence in the sense of Definition 3.2 is the same as saying that g_k does not lie in the *ideal* generated by $(g_1, ..., g_{k-1})$, with *ideal* defined as in abstract algebra (see Appendix B.1). Hence, the ideal generated by $(g_1, ..., g_{k-1}, g_k)$ is strictly larger than that generated by $(g_1, ..., g_{k-1})$ alone, because we have added at least one new element (the g_k). We prove that in our setting the process of adding new independent g_ks will terminate (proof in Appendix B.1):

Theorem 3.1. *When using Definition 3.2 for independence and real-analytic functions to approximate gs, the process of starting with a relation g_1 and iteratively adding new independent g_ks will terminate.*

If $\mathcal{M}$ is real-analytic (i.e. is cut out by a finite set of equations of type $h(\tau) = 0$ for some finite set of real-analytic hs), then after the process terminates, the set where all relations $g_1, .., g_k$ hold will be precisely $\mathcal{M}$. Otherwise, the process will still terminate, having learned all possible analytic relations that hold on $\mathcal{M}$. By a theorem of Akbulut and King [20] any smooth submanifold of $\mathbb{R}^N$ can be approximated arbitrarily well by an analytic set, so in practice the differences would be negligible.

To ensure that each new relation decreases the data manifold dimension, we could additionally prohibit $g_1, ..., g_k$ from having any syzygy $\{f_1, ..., f_k\}$ in which f_k itself is not expressible in terms of $g_1, ..., g_{k-1}$. With such definition (below) we could guarantee that a sequence of independent relations $g_1, ..., g_k$ restricts the data to a submanifold of codimension at least k (Theorem 3.2, which we prove in Appendix B.1).

Definition 3.3 (Strong Independence). *g_k is strongly independent from $g_1, ..., g_{k-1}$ if the equality $\sum_{j=1}^{k} f_j g_j = 0$ implies that f_k is expressible as $f_k = h_1 \cdot g_1 + ... + h_{k-1} \cdot g_{k-1}$.*

Theorem 3.2. *Suppose $g_1, \ldots, g_k$ is a sequence of analytic functions on B, each strongly independent of the previous ones. Denote by $\mathcal{M}_{\mathring{B}} = \{x \in \mathring{B} | g_j(x) = 0 \text{ for all } j\}$ the part of the learned data manifold lying in the interior of B. Then dimension of $\mathcal{M}_{\mathring{B}}$ is at most $N - k$.*

In addition, we construct an alternative approach with similar dimensionality reduction guarantees, which ensures that the learned relations differ to first order. For this we use a notion of independence based on *transversality*, with the following definition and lemmas (with proofs in Appendix B.1):

Lemma 3.1. *Dependence as in Definition 3.2 implies $\nabla_\tau g_k$ and $\nabla_\tau g_1, ..., \nabla_\tau g_{k-1}$ are dependent.*

Definition 3.4 (Transversality). *If for all points $\tau^{(i)} \in \mathcal{M}$ the gradients of $g_1, .., g_k$ at τ, i.e. $\nabla_\tau g|_{\tau^{(i)}}$, are linearly independent, we say that g_k is transverse to the previous relations: $g_k \pitchfork g_1, ..., g_{k\text{-}1}$.*

Using transversality, we deem g_k to be independent from $g_1, ..., g_{k-1}$ if the gradients of g_k do not lie in the span of gradients of $g_1, ..., g_{k-1}$ anywhere on $\mathcal{M}$. With this, g_k that only differs from previous relations in higher-order terms would be deemed as 'not new'. This formulation is natural from the perspective of differential geometry. Let H_{g_j} be the hypersurface defined by g_j: the set of points where $g_j = 0$. Each $H_{g_1}, ..., H_{g_k}$ contains $\mathcal{M}$. If gradients of g_k are linearly independent from gradients of $g_1, ..., g_{k-1}$, then the corresponding hypersurfaces intersect transversely along $\mathcal{M}$.

Lemma 3.2. *For once differentiable $(g_1, .., g_k)$ s.t. H_{g_j}s are transverse along their common intersection H, this intersection H is a submanifold of $\mathbb{R}^N$ of dimension $N - k$.*

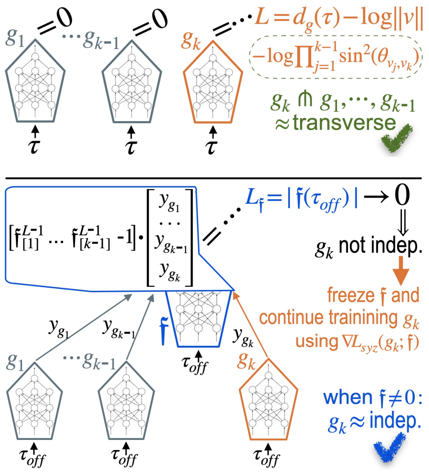
The notion of independence defined via transversality is infinitesimal and symmetric w.r.t. permuting g_ks. This is useful in settings where many relations could be discovered, because it is then better to find relations whose first order behavior differs. In cases where guaranteed decrease in dimension is not needed, using restricted syzygies could allow a flexible search for more expressive relations.

3.2 Learning Latent Relations

Here we describe the algorithm with relations g_k and restricted syzygies $\mathfrak{f}$ approximated by neural networks. Each g is represented by a neural network (NN) that takes a sequence of latent/low-dimensional states $\tau = [s_t, s_{t+1}, ..., s_T], \tau \in \mathbb{R}^N$ as input. The output of g is a scalar. We use g to denote both the relation and the NN used to learn it. If g outputs 0 for on-manifold data, this implies g has learned

Algorithm 1: Analytic Manifold Learning

```
$\{\tau^{(i)}\}_{i=1}^{d} \leftarrow$ rollouts from RL actors
train $g_1$ with loss $L = g_d(\tau) - \log\|v\|$ (Eq.1)
for $k = 2, 3, \ldots,$ do
    if aiming_for_transversality then
        train $g_k$ with loss $L_{tr}$ from Eq.2
    else // using syzygies
        train $g_k$ with loss $L$ from Eq.1
        for $j = 1, 2, \ldots,$ do
            generate $\tau_{off}, \tau_{off}^{test}$
            train $\mathfrak{f}_j$ with $L_{\mathfrak{f}} = |\mathfrak{f}_j(\tau_{off})|$
            if $\mathfrak{f}_j \neq 0$ on $\tau_{off}^{test}$ then break
             //$g_k \approx$indep.
            while $\mathfrak{f}_j(\tau_{off}^{test}) \approx 0$ do
                freeze $\mathfrak{f}_j$; train $g_k$ w.$L_{syz}$ Eq.3
```



$g_1 = 0$
$\cdots g_{k\text{-}1} = 0$
g_k
$L = d_g(\tau) - \log\|v\|$
$-\log\prod_{j=1}^{k-1}\sin^2(\theta_{v_j,v_k})$
$g_k \pitchfork g_1, \cdots, g_{k\text{-}1}$
≈transverse
τ
$[\mathfrak{f}^{L\text{-}1}_{[1]} \cdots \mathfrak{f}^{L\text{-}1}_{[k\text{-}1]}\ \text{-}1] \cdot \begin{bmatrix} y_{g_1} \\ \cdots \\ y_{g_{k-1}} \\ y_{g_k} \end{bmatrix}$
$L_{\mathfrak{f}} = |\mathfrak{f}(\tau_{off})| \rightarrow 0$
$\Downarrow$
g_k not indep.
freeze $\mathfrak{f}$ and continue training g_k using $\nabla L_{syz}(g_k; \mathfrak{f})$
y_{g_1}
$y_{g_{k\text{-}1}}$
$\mathfrak{f}$
y_{g_k}
g_1
$\cdots g_{k\text{-}1}$
τ_{off}
g_k
when $\mathfrak{f} \neq 0$: $g_k \approx$ indep.

Figure 4: Left: algorithm for learning latent relations. Top right: using transversality. Bottom right: training with syzygy $\mathfrak{f}$ to uncover if g_k is dependent, then using $\mathfrak{f}$ to modify g_k's loss. Orange & blue denotes NNs whose weights are being trained. Gray denotes learned relations whose NNs are frozen.

a function $g(\tau) = 0$, which captures a relation between states of the underlying dynamical system. g is trained on minibatches of size b of on-manifold data points $\tau^{(i)}$ using loss gradients: $\nabla L = \sum_{i=1}^{b} \nabla_g \left[L(\tau^{(i)})\right]$, where ∇_g means gradient w.r.t NN weights of g. We need to make $g \rightarrow 0$ for on-manifold data, while avoiding trivial relations (e.g. all NN weights ≈ 0). Hence, in the loss we minimize $d_g(\tau) = \frac{|g(\tau)|}{\|v\|}$, where v is the gradient of g with respect to input points $\tau^{(i)}$: $v = \nabla_\tau(g)|_{\tau^{(i)}}, v \in \mathbb{R}^N$. The gradient norm $\|v\|$ is the maximal 'slope' of the linearization of g at τ, so $d_g(\tau)$ is the distance from τ to the nearest point where this linearization vanishes ($d_g(\tau) =$ height/slope $=$ distance). Hence, $d_g(\tau)$ is a proxy for the distance from τ to the vanishing locus of g. This measure of vanishing avoids scaling problems (see Appendix B.2). We also maximize $\log\|v\|$ to further regularize g. Equation 1 summarizes our loss for g:

$$L(g) = d_g(\tau) - \log\|v\| \ \ ; \quad d_g(\tau) = |g(\tau)| / \|v\| \ \ ; \quad v = \nabla_\tau(g)|_\tau \qquad (1)$$

We proceed sequentially: first learn g_1, then g_2, and so on. Suppose that so far we learned (approximately) independent relations $g_1, \ldots, g_{k-1}$. We then keep their NN weights fixed and learn an initial version of the next relation g_k. To obtain g_k that is transverse to $g_1, .., g_{k-1}$ (Definition 3.4), we augment the loss as follows. We compute gradients of each $g_1, \ldots g_{k-1}$ w.r.t input τ. For example, for g_1 we denote this as $v_1 = \nabla_\tau(g_1)|_\tau$. Making g_k transverse to $g_1, \ldots g_{k-1}$ means ensuring that v_k is linearly independent of $v_1, \ldots, v_{k-1}$. We optimize a computationally efficient numerical measure of this: maximize the angles between v_k and all the

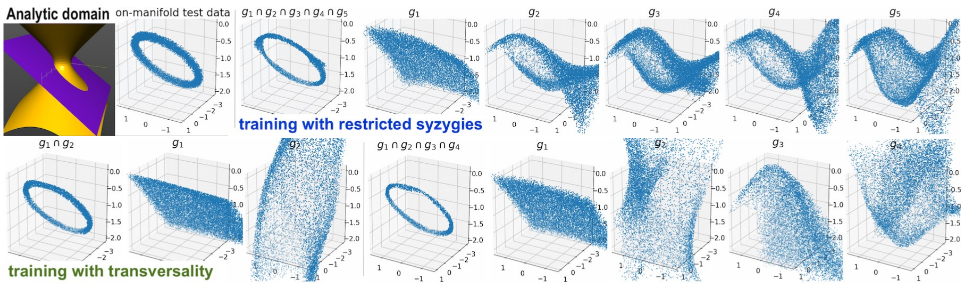


Figure 5: Learning relations $g_1, .., g_k$ on a noisy version of the analytic domain.

previous $v_1, .., v_{k-1}$. Such measure encourages transversality of subsets of relations and strongly discourages small angles. Our overall measure of transversality is the product of sines of pairwise angles, with log for stability (Appendix B.3.1 gives further discussion):

$$L_{tr}(g_k) = d_{g_k}(\tau) - \log \|v_k\| - \log \prod_{j=1}^{k-1} \sin^2(\theta_{v_j, v_k}) \tag{2}$$

For independence based on Definition 3.2, we instead learn a restricted syzygy $\mathfrak{f}(\tau_{off}, g_1, ..., g_k) = 0$. Training data for $\mathfrak{f}$ is comprised of: 1) τ_{off} (defined in Section 3.1) and 2) $y_{g_1}{=}g_1(\tau_{off}), ..., y_{g_k}{=}g_k(\tau_{off})$, i.e. outputs from $g_1, ...g_k$ with τ_{off} fed as inputs. $y_g s$ are passed directly to the next-to-last layer, which we denote as $\mathfrak{f}^{L\text{-}1} \in \mathbb{R}^{k-1}$. The last layer of $\mathfrak{f}$ computes a dot product of $\left[\mathfrak{f}^{L\text{-}1}_{[1]}, ..., \mathfrak{f}^{L\text{-}1}_{[k\text{-}1]}, \text{-}1\right]$ and $[y_{g_1}, ..., y_{g_k}]$. We use a simple L1 loss for training $\mathfrak{f}$. If $\mathfrak{f}$ outputs 0 at convergence: g_k is not independent. In this case, we freeze the weights of $\mathfrak{f}$ and continue to train g_k with augmented loss. We use gradients passed through $\mathfrak{f}$ to push g_k away from a solution that made it possible to learn $\mathfrak{f}$:

$$\nabla L_{syz}(g_k; \mathfrak{f}) = \nabla L(g_k) - \nabla_{g_k}\Big[\big|\mathfrak{f}(\tau_{off}, g_1, ..., g_k)\big|\Big] \tag{3}$$

L_{syz} encourages adjusting g_k such that it makes the outputs of (frozen) $\mathfrak{f}$ non-zero. Once $L_{syz}(g_k; \mathfrak{f})$ is minimized, we can attempt to learn another syzygy $\mathfrak{f}_2$, and so on, until we cannot uncover any new dependencies. Then g_k can be declared (approximately) independent of $g_1, ...g_{k-1}$ and we can proceed to learn g_{k+1}. All g_ks, $\mathfrak{f}$s, Ls are in latent space, so networks are small & quick to train.

An additional benefit of our formulation is that prior knowledge can be incorporated without restricting the hypothesis space. g_ks can be pre-trained in a supervised way: to output values that a prior heuristic produces on- and off-manifold. Then, g_ks can be further trained using on-manifold data, and if prior knowledge is wrong, then g_k would move away from the wrong heuristic during further training.

4 Evaluating Analytic Manifold Learning (AML)

We evaluate our AML approach with 3 sets of experiments: 1) learning on an analytic domain and visualizing relations in 3D; 2) handling dynamics with friction and drag on a *block-on-incline* domain; 3) employing learned relations to get improved representations on the *YCB-on-incline* target domain.

For our analytic domain on-manifold data comes from an intersection of a hyperboloid and a plane. The top row of Figure 5 shows results using restricted syzygies. We visualize $g_1 \cap g_2 \cap ... \cap g_5$: the intersection of the learned relations $g_1, ..., g_5$ (i.e. the intersection of the zero-level sets of these relations). The zero-level sets of individual relations are shown next. On the second row, we show training with transversality: $g_1 \cap g_2$ has two simple relations – a plane and a hollow cylinder; $g_1 \cap g_2 \cap g_3 \cap g_4$ includes smoothed cones. Transversality allows capturing information with a small number of general relations. In contrast, relations found using syzygies have more complicated shapes and can be similar in some regions, as expected. This could be useful when we need to avoid large changes, e.g. for fine-tuning or for flexible partial transfer using subsets of relations.

Next, we evaluate AML on a physics domain: a block sliding down an incline. The block is given a random initial velocity; gravity, friction and drag forces then determine its further motion. On-manifold data consists of noisy position & velocity of the block at the start and end of trajectories. Figure 6 shows AML with transversality (Appendix B.3.3 gives results with syzygies). We visualize phase space plots: arrows show change in position & velocity after 1*sec* of sliding (scaled to fit). The left plots show the case of a 45° incline and demonstrate generalization. AML is only given training data with start position & velocity $\in [0, 0.2]$, but is able to generalize to $[0, 0.4]$. The middle plots show high friction on a 35° incline. The right plots show high drag on a 10° incline. Overall, these results show that AML can generalize beyond training data ranges and capture non-linear dynamics.

Lastly, we show transfer to *YCB-on-incline* domain (rightmost in Figure 1) and compare AML to the leading approaches from our earlier experiments: *PRED* and VAE_{rpl}. We note that while *SPAIR* did ok on *RearrangeGeom*, it had significant problems reconstructing existing benchmarks. Decoding *RearrangeYCB* was problematic for all approaches (see Appendix A). Even supervised decoder training failed (with true states as training input). Decoder design is outside the scope of this work. Hence, we evaluate AML transfer using *YCB-on-incline*, which has challenging dynamics & images, but still tractable for decoding. First, AML learns relations from *Geom-on-incline*. Incline angle, friction and object pose are initialized randomly. Actions are random forces that push objects along the incline. AML is given incline, position & velocity at two subsequent steps, and the applied action.

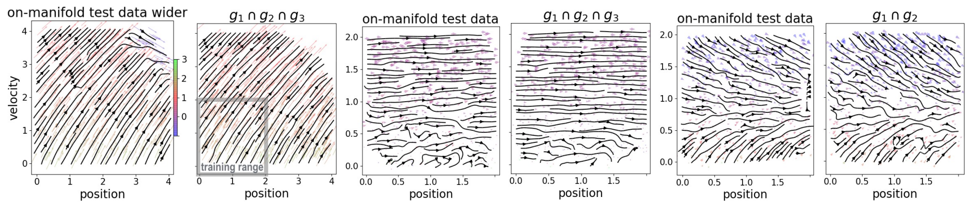


Figure 6: Phase space plots for on-manifold data and relations learned with AML for *block-on-incline*.

Then, we train an unsupervised learner (*PRED*) on the target *YCB-on-incline* domain. PPO RL drives the distribution of RGB frames. RL gets high rewards for pushing objects to stay in the middle of the incline. We impose AML relations by extending the latent part of an ELBO-based loss (with $z_{t,t+1}$ as encoder outputs):

$$\mathcal{L} = -\big[\underbrace{\log p(x|z_{t,t+1}) - KL\big(q(z_{t,t+1}) \| \mathcal{N}(0,1)\big)}_{\text{standard } ELBO \text{ for } PRED \text{ version of } VAE}\big] + \underbrace{\textstyle\sum_{k=1}^{K} \big|g_k(z_{t,t+1}, a_t)\big|}_{\text{impose } AML \text{ relations}}$$

The resulting $\text{AML}_{\text{trnsv}}$ (AML_{syz} when using syzygies) gets a better latent state alignment for object position compared to VAE_{rpl} and *PRED* without AML relations imposed (see the top plot in Figure 7).

Another important quality measure of a latent space mapping is how much it distorts the true data manifold. We quantify this as follows (on 10K test points): take pairs of low-dimensional representations τ_1^{true}, τ_2^{true} and the corresponding pixel-based representations x_1, x_2, then compute distortion coefficient $\rho_{distort} = \log\big[d_{L2}\big(\phi_{enc}(x_1), \phi_{enc}(x_2)\big) / d_{L2}\big(\tau_1^{true}, \tau_2^{true}\big)\big]$, with d_{L2} as Euclidean distance. An encoder that yields low variance of these coefficients better preserves the geometry of the low-dimensional manifold (up to overall scale). This measure is related to approaches surveyed in [21, 22] (see Appendix B.3.2). The bottom plot in Figure 7 confirms that AML helps achieving lower distortion variability.

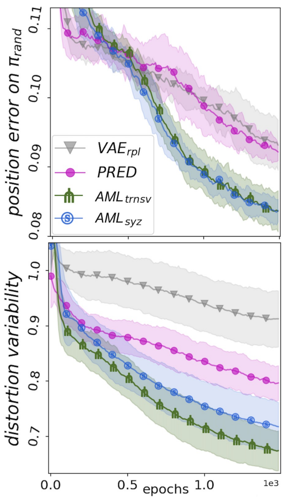


Figure 7: *YCB-on-incline*: mean of 6 training runs, shaded areas show 1 STD.

Results presented in Figure 7 show that imposing AML relations helps improve the latent space mapping of *PRED* when training on RGB frames. The distribution of the frames is non-stationary, since they are sampled using the current (changing) policy of an RL learner. Overall, this above setup aims to demonstrate the potential for sim-to-real transfer. In this case, *Geom-on-incline* plays a role of a simulator, while frames from *YCB-on-incline* act as surrogates for 'real' observations. Note that YCB objects have realistic visual appearances and their dynamics is dictated by meshes obtained from the 3D scans of real objects. Hence, there is a non-trivial mismatch between the dynamics of the simple shapes of *Geom-on-incline* domain vs realistic shapes of the *YCB-on-incline* domain.

5 Related Work

Scalable simulation suites for continuous control [23, 24, 25] bolstered progress in deep RL. However, advanced benchmarks for unsupervised learning from non-stationary data are lacking, since the community mainly focused on dataset-oriented evaluation. [2] provides such a framework for ATARI games, but it is not aimed at continuous control. [26] includes a limited set of robotics domains and 3 metrics for measuring representation quality: KNN-based, correlation, RL reward. We incorporate more standard benchmarks, introduce a variety of objects with realistic appearances (fully integrated into simulation) and measure alignment to latent state in a complimentary way (highly non-linear, but not RL-based). In future work, it would be best to create a combined suite to support both games- and robotics-oriented domains, and offer a comprehensive set of RL-based and RL-free evaluation.

Our formulation of learning latent relations is in the general setting of representation learning. This is a broad field, so in this work we focus on formalization of learning independent/modular relations that capture the true data manifold. We also provide a way to transfer relations learned on source domains to target domains. Unlike meta-learning, we do not assume access to a task distribution and do not view target task reward as the main focus. Our sequential approach to learning $g_1, ..., g_k$ has conceptual parallels with a functional Frank-Wolfe algorithm [27], but without convex optimization. Learning sequentially helps avoid instabilities, e.g. from training flexible NN mixtures with EM [8]. There is prior work for learning algebraic (meaning polynomial) relations, but its criterion for relation simplicity is based on polynomial degree. Such approaches are based on computational algebra and spectral methods from linear algebra. This line of work was initiated by [28, 29, 30], with extensions [31, 32, 33, 34, 35], applications [36, 37] and learning theory analysis [38, 39]. Our formulation is more general, since we learn analytic relations and approximate them with neural networks. We summarize the main differences & point out potential connections in Appendix B.2.

Conclusion and Future Work

We proposed a suite for evaluation of latent representations and showed that additional latent space structure can be beneficial, but could stifle learning in existing approaches. We then presented AML: a unified approach to learn latent relations and transfer them to target domains. We offered a rigorous mathematical formalization, algorithmic variants & empirical validation for AML.

We showed applications of AML to physics & robotics domains. However, in general AML does not assume that source or target domains are from a certain field, such as robotics, or have particular properties, such as continuity in the adjacent latent states or existence of an easy-to-learn transition model. As as long some relation exists between the subsequences of latent states – AML would attempt to learn it, and would succeed if a chosen function approximator is capable of representing it. Moreover, AML relations can be learned on the latent space of any unsupervised learner trained on the source domain. In this case, AML would capture abstract relations that encode the regularities embedded in the latent representation learned on the source domain. Imposing these relations during transfer could help to preserve (i.e. carry over) these regularities. This alternative could be better than starting from scratch and better than fine-tuning. Starting from scratch is not data-efficient. Fine-tuning is prone to getting stuck in local optima, causing permanent degradation of performance, especially in case of a non-trivial mismatch between the source and target domains.

AML can build a modular representation of relations encoded in the latent/low-dimensional space. Hence, AML can enable a dynamic partial transfer and thus help recover from negative transfer in cases of large source-target mismatch. In our follow-up work, we intend to dynamically adjust the strength of imposing each latent relation on the target domain. For this, we would combine the learned relations $g_1, ..., g_k$ using prioritization weights $w_1, ..., w_k$. These weights would be optimized by propagating the gradients of the RL loss w.r.t. the latent state representation (that these weights would influence). Further extensions could include, for example, lifelong learning: we could gradually expand the set of learned relations and discard relations whose weights decay to zero as the lifelong learning proceeds. Another promising option would be to learn policy representations (rather than state representations). If AML could be used to learn policies that are in some sense independent, then we could provide a way to learn a *portfolio* of policies that are complementary. Then, we could construct algorithms for learning diversified portfolios, such that a system capable of executing any policy in a portfolio could provide robustness to uncertainty and changes in the environment.

Acknowledgments

This research was supported in part by the Knut and Alice Wallenberg Foundation. This work was also supported by an "Azure for Research" computing grant. We would like to thank Yingzhen Li, Kamil Ciosek, Cheng Zhang and Sebastian Tschiatschek for helpful discussions regarding unsupervised & reinforcement learning and variational inference.

Appendices

A Evaluation Suite for Unsupervised Learning

A.1 Benchmarking Alignment : Algorithm Descriptions and Further Evaluation Details

In this section, we include more detailed descriptions of the existing approaches we evaluated, describe parameters used for evaluation experiments, and give examples of reconstructions. Code for the evaluation suite environments can be obtained at: `https://github.com/contactrika/bulb`

– VAE_{v_0} [6]: a VAE with a 4-layer convolutional encoder and corresponding de-convolutional decoder (same conv-deconv stack is also used for all the other VAE-based methods below).

– VAE_{rpl}: a VAE with a replay buffer that retains 50% of initial frames from the beginning of training and replays them throughout training. This is our modification of the basic VAE to ensure consistent performance on frames coming from a wider range of RL policies. We included this replay strategy into the rest of the algorithms below, since it helped improve performance in all cases.

– *β-VAE* [14]: a VAE with an additional β parameter in the variational objective that encourages disentanglement of the latent state. To give *β-VAE* its best chance we tried a range of values for β.

– *SVAE*: a sequential VAE that is trained to reconstruct a sequence of frames $x_1, ..., x_t$ and passes the output of the convolutional stack through LSTM layer before decoding. Reconstructions for this and other sequential versions were also conditioned on actions $a_1, ..., a_t$.

– *PRED*: a VAE that, given a sequence of frames $x_1, ..., x_t$, constructs a predictive sequence $x_1, ..., x_{t+k}$. First, the convolutional stack is applied to each x_i as before; then, the t output parts are aggregated and passed through fully connected layers. Their output constitutes the predictive latent state. To decode: this state is chunked into $t + k$ parts, each fed into deconv stack for reconstruction.

– *DSA* [15]: a sequential autoencoder that uses structured variational inference to encourage separation of static vs dynamic aspects of the latent state. It uses LSTMs in static and dynamic encoders. To give *DSA* its best chance we tried uni- and bidirectional LSTMs, as well as replacing LSTMs with GRUs, RNNs, convolutions and fully connected layers.

– *SPAIR* [16]: a spatially invariant and faster version of AIR [17] that imposes a particular structure on the latent state. *SPAIR* overlays a grid over the image (e.g. 4x4=16, 6x6=36 cells) and learns 'location' variables that encode bounding boxes of objects detected in each cell. 'Presence' variables indicate object presence in a particular cell. A convolutional backbone first extracts features from the overall image (e.g. 64x64 pixels). These are passed on to further processing to learn 'location','presence' and 'appearance' of the objects. The 'appearance' is learned by object encoder-decoder, which only sees a smaller region of the image (e.g. 28x28 pixels) with a single (presumed) object. The object decoder also outputs transparency alphas, which allow rendering occlusions.

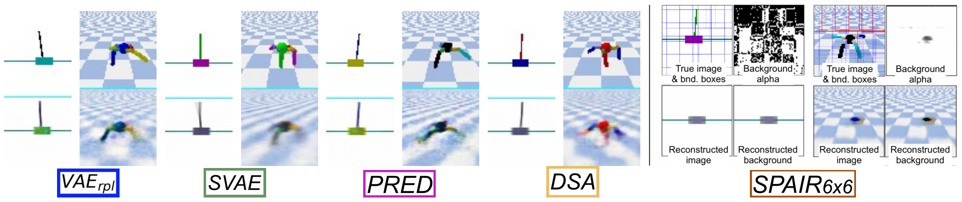


Figure 1: Unseen frames (top) and reconstructions (bottom) after 500 training epochs.

Neural network architectures and training parameters:

In our experiments, unsupervised approaches learn from 64x64 pixel images, which are rendered by the simulator. All approaches (except *SPAIR*) first apply a convolutional stack with 4 hidden layers, (with [64,64,128,256] conv filters). The decoder has analogous de-convolutions. Fully-connected and recurrent layers have size 512. Using batch/weight normalization and larger/smaller network depth & layer sizes did not yield qualitatively different results. The latent space size is set to be twice the dimensionality of the true low-dimensional state. For VAE we also tried setting it to be the same, but this did not impact results. *PRED*, *SVAE*, *DSA* use sequence length 24 for pendulums & 16 for locomotion (increasing to 32 yields similar results). *SPAIR* parameters and network sizes are set to match those in [16]. We experimented with several alternatives, but only the cell size had a noticeable effect on the final outcome. We report results for 4x4 and 6x6 cell grids, which did best.

To decouple the number of gradient updates for unsupervised learners from the simulator speed: frames for training are re-sampled from replay buffers. These keep 5K frames and replace a random subset with new observations collected from 64 parallel simulation environments, using the current policy of an RL learner. Training hyperparameters are the same for all settings (e.g. using Adam optimizer [40] with learning rate set to 1e-4). Since different approaches need different time to perform gradient updates, we equalize resources consumed by each approach by reducing the batch size for the more advanced/expensive learners. VAE_{v_0}, VAE_{rpl}, β-*VAE* get 1024 frames per batch; for sequential approaches (*SVAE*, *PRED*, *DSA*) we divide that by the sequence length; for *SPAIR* we use 64 frames per batch (since *SPAIR*'s decoding process is significantly more expensive).

Reconstructions for benchmarks and the new multi-object domains:

Reconstruction for benchmark domains (e.g. *CartPole*, *InvertedPendulum*, *HalfCheetah*, *Ant*) was tractable for *VAE*, *SVAE*, *PRED*, *DSA*. Decoded images were sharp when these algorithms were trained on a static dataset of frames. However, then trained on streaming data with a changing RL policy, decoding was more challenging. It took longer for colors to emerge, especially for *SVAE* and *DSA*. Sometimes robot links were missing, especially for poses that were seen less frequently.

We attempted to run *SPAIR* on these benchmark domains as well. However, it had difficulties with reconstruction. The thin pole in *CartPole* domain was completely lost, and *SPAIR* mistook the cart base as a part of background. For

HalfCheetah and *Ant*: a bounding box was detected around the robot, signifying that *SPAIR* did separate it from the background. However, cheetah robot was reconstructed only as a faint thin line, and legs of the *Ant* were frequently missing. Right set of plots in Figure 1 shows examples of reconstructions, red bounding boxes show detected foreground regions; blue boxes indicate inactive boxes. *SPAIR* is not specifically designed for domains like this, since its strengths are best seen in identifying/tracking separate objects. Thin object parts and dynamic backgrounds in the benchmark domains are not the best match for *SPAIR*'s strongest sides.

As we noted in the main paper, all existing approaches we tried had difficulties decoding *RearrangeYCB* domain. *SPAIR* did manage to produce reasonable reconstructions, albeit missing/splitting of objects was still common. Figure 2 shows example reconstructions after training for 10K epochs (≈32 hours) and after 100K epochs. Bounding boxes reported by *SPAIR* were not tight even after 100K epochs (up to 11 days of training overall on one NVIDIA GeForce GTX1080 GPU). We used PyTorch implementation from [41], which was tested in [42] to reproduce the original *SPAIR* results (and we added the capability to learn non-trivial backgrounds). An optimized Tensorflow implementation could potentially offer a speedup, but PyTorch has an advantage of being more accessible and convenient for research code.

VAE, *SVAE*, *PRED*, *DSA* did not achieve good reconstructions even on *RearrangeGeom* domain. Figure 3 shows example reconstructions. Hence, in the main paper, for analyzing alignment on *RearrangeGeom* domain we chose VAE_{rpl} and *SPAIR*. We focused on these, since VAE_{rpl} offered speed and simplicity, while *SPAIR* gave better reconstructions.

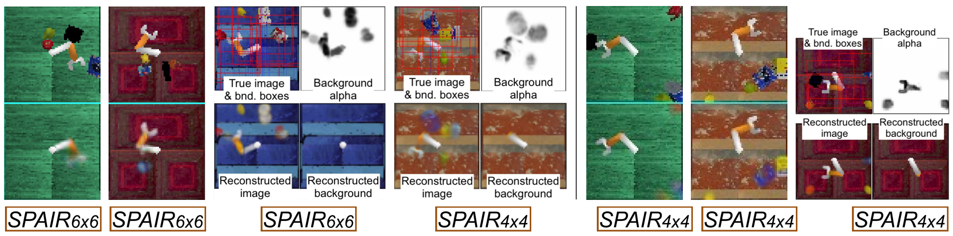


Figure 2: Left side: SPAIR *RearrangeYCB* results after 10K epochs. Right side: SPAIR after 100K epochs. True images are in the top row, reconstructions in the bottom. Thin red bounding boxes overlaid over true images (in the top row) show that bounding boxes did not shrink with further training. SPAIR 6x6 tended to split large objects into pieces (visible in the case with blue background). SPAIR 4x4 did not split objects and had better results for low-dimensional alignment.

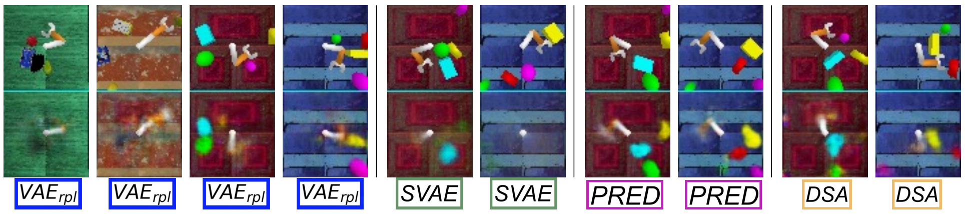


Figure 3: True images (top) and reconstructed images (bottom) after 10K training epochs.

B Analytic Manifold Learning

B.1 Proofs and Technical Background for Mathematical Formulation

Here we present an extended version of Section 3.1 from the main paper. This version contains proofs for all lemmas and theorems, provides relevant technical background from abstract algebra and geometry. We draw analogies with simpler settings from linear algebra to highlight connections with settings that are common in ML literature.

Let $\mathbb{R}^N$ be the ambient space of all possible latent state sequences τ (of some fixed length). Let $\mathcal{M}$ be the submanifold of actual state sequences that a dynamical system from one of our domains could generate (under any control policy)[3]. A common view of discovering $\mathcal{M}$ is to learn a mapping that would produce only plausible sequences as output (the 'mapping' view). Alternatively, a submanifold can be specified by describing all the equations (i.e. relations) that have to hold for the points in the submanifold. Recall an example from linear algebra, where a submanifold is linear, a.k.a. a vector subspace. This submanifold can be represented as an image of some linear map (the 'mapping' view), or as null space of some collection of linear functions, a.k.a. a system of linear equations. The latter is the 'relations' view: specifying which relations have to hold for a point to belong to the submanifold.

B.1.1 Definitions of Independence for Learning Independent Relations

We are interested in finding relations that are in some sense independent. One notion of independence is the *functional independence*. Relations $g_1, ..., g_k$ are said to be functionally independent if there is no (non-trivial) function $f : \mathbb{R}^k \to \mathbb{R}$ s.t. $f(g_1(\cdot), ..., g_k(\cdot)) = 0$. However, with such definition, $g(\tau)$ and $h(\tau)g(\tau)$ could be deemed independent[4], even when $h(\cdot)g(\cdot)$ does not provide an additional interesting relation, e.g. $g(\tau)$ vs $\sin(\tau)g(\tau)$. Hence, we need a stricter version of independence. To describe such a version we use the formalism of modules.

A *module* is the generalization of the concept of vector space, where the coefficients lie in a ring instead of a field. In our case, both elements of the module and elements of the ring are functions on $\mathbb{R}^N$. We observe that the set of functions that vanish on $\mathcal{M}$ is closed under the module operations '$+, -$' and multiplication by ring elements, hence it is a (sub-)module. Recalling the definition of independence for vectors of a vector space, we note that the default notion of independence for elements of a module is analogous. In this setting, a *syzygy* $\mathfrak{f}^{\ddagger}$ is a linear combination

[3]In this work, we use the term 'manifold' in the sense most commonly used in the machine learning literature, i.e. without assuming strict smoothness conditions.

[4]f can transform gs in any way, but does not have direct access to τ, so $h(\tau)$ can not be a 'coefficient'.

of relations $g_1, ..., g_k$ with coefficients $f_1, ..., f_k$ in the ring of functions. If there is no syzygy $\mathfrak{f}^\ddagger = \{f_1, .., f_k\}$ s.t. $\sum_{j=0}^{k} f_j g_j$ vanishes, then $g_1, ..., g_k$ are independent.

However, for our case the above notion of independence is now too strict, because it would deem any relations g_1, g_2 dependent: $g_1 \cdot g_2 - g_2 \cdot g_1 = 0$ holds for any g_1, g_2. We propose several strategies to avoid this problem. One option is to define *restricted syzygies*, presented below.

Definition 3.1 (Restricted Syzygy). *Restricted syzygy for relations $g_1, ..., g_k$ is a syzygy with the last entry f_k equal to -1, i.e. $\mathfrak{f} = \{f_1, ..., f_{k-1}, f_k = -1\}$ with $\sum_{j=1}^{k} f_j g_j = 0$.*

Definition 3.2 (Restricted Independence). *g_k is independent from $g_1, ..., g_{k-1}$ in a restricted sense if the equality $\sum_{j=1}^{k} f_j g_j = 0$ implies $f_k \neq -1$, i.e. if there exists no restricted syzygy for $g_1, ..., g_k$.*

For $\mathfrak{f} = \{f_1, ..., f_{k-1}, f_k = -1\}$ we denote $\sum_{j=1}^{k} f_j(\tau) g_j(\tau)$ by $\mathfrak{f}(\tau, g_1, ..., g_k)$.

Using definitions above, we construct a practical algorithm for learning an (approximately) independent set of relations. The overall idea is: while learning g_ks, we are also looking for restricted syzygies $\mathfrak{f}(\tau, g_1, ..., g_k) = 0$. Finding them would mean g_ks are dependent (in the sense of Definition 3.2), so we augment the loss for learning g_ks to push them away from being dependent. We proceed sequentially: first learning g_1, then learning g_2 while ensuring no restricted syzygies appear for $\{g_1, g_2\}$, then learning g_3 and so on.

For training g_ks we use *on-manifold* data: τ sequences come from our dynamical system (i.e. satisfying physical equations of motion, etc). Restricted syzygies $\mathfrak{f}$ are trained using *off-manifold* data: sequences that do not lie on our data submanifold. We denote such subsequences as $\tau_{off} = \{s_{off_t}, s_{off_{t+1}}, ..., s_{off_T}\}$. Off-manifold data is needed for $\mathfrak{f}$ since we aim for independence of g_ks on $\mathbb{R}^N$, not restricted to their output on data that lies on $\mathcal{M}$ (when restricted to $\mathcal{M}$ the g_ks are zero, and so are trivially dependent). τ_{off} do not lie on our data submanifold and can come from thickening of on-manifold data or can be random (when $\mathbb{R}^N$ is large, the probability a random sequence satisfies equations of motion is insignificant).

Observe that independence in the sense of Definition 3.2 is the same as saying that g_k does not lie in the *ideal* generated by $(g_1, ..., g_{k-1})$, with *ideal* defined as in abstract algebra[5]. Hence the ideal generated by $(g_1, \ldots, g_{k-1}, g_k)$ is strictly larger than that generated by $(g_1, \ldots, g_{k-1})$ alone, because we have added at least one new element (the g_k). Below we prove that in our setting the process of adding new independent g_ks will terminate.

[5]In the language of abstract algebra: we consider functions on $\mathbb{R}^N$ as module over itself. When a ring is viewed as a module over itself, a submodule of a ring is called an *ideal.* Thus the set of relations that hold on $\mathcal{M}$ is an ideal, called 'the ideal of $\mathcal{M}$', written $I(\mathcal{M})$. When considering only subsets of relations that hold on $\mathcal{M}$, we will also talk about the 'ideal generated by $(g_1, ...g_k)$', which is, by definition, the smallest ideal containing $g_1, ..., g_k$. One can show that this ideal consists of all linear combinations of $g_1, ..., g_k$ with functions as coefficients.

Theorem 3.1. *When using Definition 3.2 for independence and real-analytic functions to approximate gs, the process of starting with a relation g_1 and iteratively adding new independent g_ks will terminate.*

Proof. First, we assume that the values of each dimension d of τ lie between some minimum constants c_d and maximum C_d. This is to model actual data observations that are limited by real-world boundaries. This implies that instead of working with unrestricted ambient space, we will work with a compact box B, and the corresponding subset of the data manifold $\mathcal{M}_B = \mathcal{M} \cap B$. The precise values of c_ds, C_ds and even the rectangular shape of the box B are immaterial; what is needed is that B is compact and is cut out by a collection of analytic inequalities. In technical terms: we require that B is compact and *real semi-analytic*. To avoid boxes with pathological shapes we require in addition that B is the closure of its interior $\mathring{B}$. Possible Bs include a closed ball, or an arbitrary convex polytope.

We consider the case of using neural networks for approximating relations $g_1, \ldots, g_k$. For networks with real-analytic activation functions (e.g. sigmoid, tanh), the gs and relations between them would be real-analytic (recall that a function is analytic if it is locally given by a convergent power series). The g_k being independent in the sense of Definition 3.2 implies g_k is not in the ideal generated by $(g_1, ..., g_{k-1})$ inside the ring of real-analytic functions. This means that $(g_1), (g_1, g_2), \ldots, (g_1, \ldots, g_k)$ is a strictly increasing sequence of ideals inside the ring of real-analytic functions on the ambient space B. A theorem of J. Frisch [43, Théorème (I, 9)] says that the ring of analytic functions on a compact real semi-analytic space B is *Noetherian*, meaning that any growing chain of ideals in it will stabilize. This means that after a finite number of iterations we would be unable to learn a new independent g_k, meaning we would have found all analytic relations that hold on $\mathcal{M}_B$, thus terminating the process. □

If $\mathcal{M}$ itself is cut out by a finite set of equations of type $h(\tau) = 0$ for some finite set of real-analytic hs), then after the process terminates, the subset of B where all relations $g_1, .., g_k$ hold will be precisely $\mathcal{M}_B$. This is the same as saying that all the hs defining $\mathcal{M}$ will be in the ideal $(g_1, \ldots, g_k)$. If $\mathcal{M}$ is not cut out by global real-analytic relations, the process will still terminate, having learned all possible global analytic relations that hold on $\mathcal{M}_B$.

We remark that by a theorem of Akbulut and King [20] any smooth submanifold of $\mathbb{R}^N$ can be approximated arbitrarily well by $\mathcal{M}$ defined by a finite set of analytic equations $g(\tau) = 0$. The same is true even when $g(\tau)$ are restricted to be polynomial. This means that if one ignores the issues of complexity of the defining equations g, the differences between various categories of manifolds (smooth, analytic, or algebraic) could be ignored. The above may seem to suggest that methods based on polynomials may suffice. In practice, the polynomial relations needed may be of very high degree. Hence, using neural networks to learn (approximate) relations would be more suitable.

We further note that in practice we of course don't have access to $\mathcal{M}$ or even $\mathcal{M}_B$, but only to a finite sample of data points in $\mathcal{M}_B$. The fact that finding independent g_i's vanishing at these points will terminate is a (simpler) special case of the Theorem 3.1, which guarantees that even the more complicated idealized set of relations defining $\mathcal{M}_B$ can be learned in finite time.

Observe that if g_k is dependent on $g_1, \ldots, g_{k-1}$ then the set of points where g_k is zero contains the set of points where all the other $g_1, ..., g_{k-1}$ are zero. The converse is not true: while g_k is different from the previous relations in a non-trivial way, it might happen that adding g_k as a relation does not restrict the learned manifold to a smaller set. This arises because of the non-linearity in our setting[6].

To ensure that each new relation decreases the data manifold dimension, we could additionally prohibit $g_1, ..., g_k$ from having any syzygy $\{f_1, ..., f_k\}$ in which f_k itself is not expressible in terms of $g_1, ..., g_{k-1}$. This is encoded in the definition below.

Definition 3.3 (Strong Independence)**.** *g_k is strongly independent from $g_1, ..., g_{k-1}$ if the equality $-f_k g_k = f_1 \cdot g_1 + ... + f_{k-1} \cdot g_{k-1}$ implies that f_k is expressible as $f_k = h_1 \cdot g_1 + ... + h_{k-1} \cdot g_{k-1}$.*

In Theorem 3.2 we will show that imposing relations $(g_1, \ldots, g_k)$, such that each new relation is strongly independent from the previous ones, restricts data to a submanifold of codimension at least k. Since we don't assume that $\mathcal{M}$ has to be smooth, the notion of dimension needs to be defined precisely. Thus, before embarking on a formal statement and a proof of Theorem 3.2, we give such a definition and discuss related notions needed in the proof.

B.1.2 Definitions of Dimension in Geometry and Algebra

For smooth manifolds, which are locally homeomorphic to some $\mathbb{R}^n$, the dimension is simply defined to be n, and the invariance of dimension theorem of Brouwer (see [44, Theorem 2.26]) ensures that this is unambiguous (which, in light of Cantor's proof that all $\mathbb{R}^n$s have the same number of points and Peano's construction of space-filling curves is not as obvious as it may seem a priori).

For arbitrary subsets X of $\mathbb{R}^n$ one can then analogously define $\dim X = d$ if and only if X contains an open set homeomorphic to an open ball in R^n, but not an open set homeomorphic to an open ball in $\mathbb{R}^e$, for $e > d$. We will call this *geometric dimension*. This is the definition we will use when referring to dimension of $\mathcal{M}$.

Now suppose X is a semi-analytic subset of $\mathbb{R}^n$, meaning a subset locally defined by a system of analytic equations and inequalities[7]. While X is in general not smooth, it admits a decomposition into smooth parts. Then, the definition of geometric dimension $\dim X$ given above coincides with just taking the largest dimension of any part (see, for example, [47, Proposition 2.10 and Remark 2.12]). This definition

[6]This is in contrast to linear algebra, where adding an independent linear equation necessarily decreases the dimension of the subspace of solutions.

[7]The manifolds we are learning are actually much nicer: they are globally defined by analytic equations. This means, by definition, that they are C-analytic sets (an abbreviation of Cartan real analytic sets; see [45, Definition 1.5] and [46], particularly the Paragraphe 11).

is *local*, meaning that if we define dimension of X at a point $\tau \in X$, denoted by $\dim_\tau X$, to be dimension of $X \cap U$ for all sufficiently small open neighborhoods U of τ, then $\dim X = \sup_\tau(\dim_\tau X)$. Of course one also has that $Y \subset X$ implies $\dim Y \leq \dim X$. See [48, II.1.1] for all this and more.

In order to relate this dimension to properties of the relations g_i that define $\mathcal{M}$, we need to connect $\dim \mathcal{M}$ to dimensions of algebraic objects arising from g_is. These will be rings of various kinds. Thus, we need theory of dimensions of rings.

In commutative algebra the standard way to define a dimension of a ring is due to Krull. It says that dimension of a ring R, denoted krdim R, is the length d of the longest chain of prime ideals $I_d \subsetneq I_{d-1} \subsetneq ... \subsetneq I_0$ in R. Note that this has some resemblance to the fact that dimension of a vector space is equal to the length d of the longest chain of subspaces $V = V_d \supsetneq V_{d-1} \supsetneq ... \supsetneq V_0$. For an ideal $I \subset R$ the Krull dimension is defined as $\operatorname{krdim} I := \operatorname{krdim}(R/I)$, where R/I is the quotient ring. See [49, Chapter 8 and onwards].

B.1.3 Statement and Proof of Theorem 3.2

Theorem 3.2. *Suppose $g_1, \dots, g_k$ is a sequence of analytic functions on B, each strongly independent of the previous ones. Denote by $\mathcal{M}_{\mathring{B}} = \{\tau \in \mathring{B} | g_j(\tau) = 0$ for all $j\}$ the part of the learned data manifold lying in the interior of B. Then dimension of $\mathcal{M}_{\mathring{B}}$ is at most $N - k$.*

Proof outline: Strong independence (Definition 3.3) is directly related to the definition of regular sequences. The proof ultimately aims to use Proposition 18.2 in [49], which ensures that ideals defined by regular sequences have low dimension. To deduce that $\mathcal{M}$ has low dimension, we need to relate the Krull dimension of the ideal $(g_1, ..., g_k)$ to the geometric dimension of $\mathcal{M}$. To do this we pass through a number of intermediate stages. First we localize, and complexify. This allows us to equate the dimension of the local complexified ideal $(g_1, ..., g_k)$ to that of the local complexified ideal of $\mathcal{M}$, which we do by using local analytic Nullstellensatz. We also equate the common dimension of these two ideals to the (local complex) geometric dimension of $\mathcal{M}$. Then, we relate this to local real dimension of $\mathcal{M}$. Finally, we get a bound on the (global) dimension of $\mathcal{M}$ itself.

Proof. The Definition 3.3 is equivalent to saying that g_k is not a zero divisor in the ring of functions modulo the ideal generated by $(g_1, \dots, g_{k-1})$. To see this, we argue as follows. By definition, in any ring, an element g is not a zero divisor if $fg = 0$ implies that $f = 0$. Equality $fg = 0$ in the quotient ring means that, in the ring of functions, we have: $fg = \sum_{i=1}^{k-1} f_i g_i$. Thus if g is not a zero divisor in the quotient ring, then $fg = \sum_{i=1}^{k-1} f_i g_i$ implies f is zero in the quotient ring, that is to say $f = h_1 g_1 + ... + h_{k-1} g_{k-1}$, for some functions $h_1, ..., h_{k-1}$.

Thus, a sequence $(g_1, g_2, \dots, g_{k-1}, g_k)$ where each g_i is strongly independent from the previous ones is a *regular sequence*, see [49, Sections 10.3, beginning of Section 17].

Let τ be a point in $\mathcal{M}_{\mathring{B}}$. We will denote by $\mathcal{O}_\tau$ the ring of germs[8] of real-analytic functions defined near τ, which is isomorphic to the ring of convergent power series centered at τ. We will denote the complex version of this ring by $\mathcal{O}_\tau^{\mathbb{C}}$.

The *localization* ring L_τ of the ring of analytic functions on $\mathring{B}$ at a point τ is defined as the set of equivalence classes of pairs of analytic functions (f, h) s.t. $h(\tau) \neq 0$, with the equivalence relation $(f_1, h_1) \sim (f_2, h_2) \iff f_1 \cdot h_2 = f_2 \cdot h_1$. This is a formal way of introducing fractions f/h. One also has the *localization map* from the original ring to the localization ring. It sends f to the equivalence class represented by the pair $(f, 1)$, where 1 is the constant function. In our setting, if one identifies the set of equivalence classes with germs, this map performs a 'type conversion' from an analytic function f to its germ at τ. In fact, the localized ring L_τ is a subring of the ring of germs $\mathcal{O}_\tau$. Indeed, a fraction $\frac{f}{h}$ defines an analytic function on some open neighborhood of τ and the corresponding germ depends only on the equivalence class, thus giving a map $L_\tau \to \mathcal{O}_\tau$. Clearly the germ is zero only when f is zero, so this map is an injection, and L_τ is a subring of $\mathcal{O}_\tau$.

However, L_τ is not all of $\mathcal{O}_\tau$, since not every function analytic at τ is a ratio of two functions analytic on all of $\mathring{B}$. To remedy this, we consider *completions* of both L_τ and $\mathcal{O}_\tau$, denoted $\hat{L}_\tau$ and $\hat{\mathcal{O}}_\tau$ with respect to the maximal ideal of germs vanishing at τ. A completion is perhaps most familiar as a procedure that gives real numbers from rational ones, by means of equivalence classes of Cauchy sequences. In the present situation, a sequence of germs is deemed Cauchy if the difference of any two elements with sufficiently high indexes vanishes to arbitrarily high order (this is known as Krull topology). The completion (of either L_τ or $\mathcal{O}_\tau$) is then isomorphic to the ring of formal power series centered at τ. Indeed, just taking Cauchy sequences of germs of polynomial functions we get that the completion contains all formal power series centered at τ; and any Cauchy sequence (in either L_τ or $\mathcal{O}_\tau$) is equivalent to one made up of polynomials, and converges to a formal power series.

We now argue as follows. Since the localization procedure commutes with taking quotients, and the localization map takes non-zero divisors to non-zero divisors ([50, Section 15.4]), we conclude that for each τ the sequence of germs of $g_1, \ldots, g_k$ is a regular sequence in L_τ. On the other hand, by [51, Lemma 10.67.5 and Lemma 10.96.2] (as cited in proof of [51, Lemma 23.8.1.]) a sequence is regular in a local ring if and only if it is regular in the completion, so $g_1, ..., g_k$ is regular in $\hat{L}_\tau = \hat{\mathcal{O}}_\tau$, and so also in $\mathcal{O}_\tau$.

We claim that the corresponding complexified germs form a regular sequence in $\mathcal{O}_\tau^{\mathbb{C}}$ as well. Indeed, if $f_{j+1}g_{j+1} = \sum_{l=1}^{j} f_l g_l$ on neighborhood U of τ in $\mathbb{C}^n$, then restricting to $U^{\mathbb{R}} = U \cap \mathbb{R}^n$ and taking real and imaginary parts we see (denoting by $re(f)$ and $im(f)$ the real and imaginary parts of any complex-valued function f) that on $U^{\mathbb{R}}$ we have $re(f_{j+1})g_{j+1} = \sum_{l=1}^{j} re(f_l)g_l$ and $im(f_{j+1})g_{j+1} = \sum_{l=1}^{j} im(f_l)g_l$.

[8]A *germ* of a function at a point τ is an equivalence class of functions defined near τ, where f_1, f_2 are considered equivalent if there exists an open neighbourhood of τ s.t. restrictions of f_1 and f_2 to that neighborhood coincide.

Since g_l's form a regular sequence we must have $re(f_{j+1}) = \sum_{l=1}^{j} a_l g_l$, $im(f_{j+1}) = \sum_{l=1}^{j} b_l g_l$, so that denoting $c_l = a_l + ib_l$ we have $f_{j+1} = \sum_{l=1}^{j} c_l g_l$ on an open neighborhood of τ in $\mathbb{R}^N$. Then the same is true on an open neighborhood of τ in $\mathbb{C}^N$, and so g_{j+1}s form a regular sequence in $\mathcal{O}_\tau^{\mathbb{C}}$, as wanted.

Thus the depth of the ideal $I = (g_1, \ldots, g_k)$ in $\mathcal{O}_\tau^{\mathbb{C}}$ (defined as the maximal length of a regular sequence of elements in I [49, Section 17.2]) is at least k. Moreover, depth of the radical of $J = \sqrt{I}$ is the same ([49, Corollary 17.8]), and by the local complex-analytic Nullstellensatz (for example, [52, Theorem 7, Section III.A]) J is the ideal of germs of complex-analytic functions vanishing on the zero-locus of the g_js near τ. By Proposition 18.2 in [49], codimension of J is is at least k (by definition codimension it is the supremum of lengths of chains of primes descending from J, see [Chapter 9][49]), so $\dim J + \operatorname{codim} J \leq n$, and so the (Krull) dimension of J is at most $N - k$.

Local structure theorem for analytic sets implies that this is also the local (complex) geometric dimension of $\mathcal{M}^{\mathbb{C}}$ (see [48, Proposition 1, IV.4.3] or [52, Section IIIA]). Near τ the real vanishing locus $\mathcal{M}$ is then of real dimension at most $N - k$ ([46, Proposition 5]). Thus $\mathcal{M}$ is of local dimension of at most $N - k$ at all points of $\mathring{B}$, and hence $\dim \mathcal{M}_{\mathring{B}} \leq N - k$ as wanted. □

B.1.4 Learning Transverse Relations

It is also possible to define an alternative approach that would provide similar dimensionality reduction guarantees, while also ensuring that the learned relations differ to first order. To this end we utilize a notion of independence based on *transversality* as follows.

Lemma 3.1. *Dependence as in Definition 3.2 implies $\nabla_\tau g_k$ and $\nabla_\tau g_1, ..., \nabla_\tau g_{k-1}$ are dependent.*

Proof. Suppose $g_1, ..., g_k$ are dependent in the sense of Definition 3.2, i.e. $g_k = f_1 \cdot g_1 + ... + f_{k\text{-}1} \cdot g_{k\text{-}1}$. We take gradients w.r.t coordinates of $\mathbb{R}^N$ (the ambient space) and obtain:

$$\nabla_\tau g_k = \nabla_\tau [f_1 \cdot g_1] + ... + \nabla_\tau [f_{k\text{-}1} \cdot g_{k\text{-}1}] = \sum_{j=1}^{k\text{-}1} \Big(f_j \nabla_\tau g_j + g_j \nabla_\tau f_j \Big)$$

Restricting to points in $\mathcal{M}$ and observing that $g_j = 0$ on $\mathcal{M}$, we get $\nabla_\tau g_k = \sum_{j=1}^{k-1} f_j \nabla_\tau g_j$. □

Definition 3.4 (Transversality)**.** *If for all points $\tau^{(i)} \in \mathcal{M}$ the gradients of $g_1, .., g_k$ at τ, i.e. $\nabla_\tau g|_{\tau^{(i)}}$, are linearly independent, we say that g_k is transverse to the previous relations: $g_k \pitchfork g_1, ..., g_{k\text{-}1}$.*

Using transversality, we deem g_k to be independent from $g_1, ..., g_{k-1}$ if the gradients of g_k do not lie in the span of gradients of $g_1, ..., g_{k-1}$ anywhere on $\mathcal{M}$. With this, g_k that only differs from previous relations in higher-order terms would still be deemed as 'not new'. This stronger notion of independence would be useful for settings where many relations could be discovered, because it is then better to find relations whose first order behavior differs. This formulation is natural from the perspective of differential geometry. Let H_{g_j} be the hypersurface defined by g_j: the set of points where $g_j = 0$. Each hypersurface $H_{g_1}, ..., H_{g_k}$ contains $\mathcal{M}$. If gradients of g_k are linearly independent from gradients of $g_1, ..., g_{k-1}$, then the corresponding hypersurfaces are said to intersect transversely along $\mathcal{M}$.

Lemma 3.2. *For once differentiable $(g_1, .., g_k)$ s.t. H_{g_j}s are transverse along their common intersection H, this intersection H is a submanifold of $\mathbb{R}^N$ of dimension $N-k$.*

Proof. Consider the map $G : \mathbb{R}^N \to \mathbb{R}^k$ given by $G = (g_1, \ldots, g_k)$. The fact that H_{g_j}s are transverse along H means that the derivative $DG(p)$ has rank k for any $p \in H$. This means that we can pick k linearly independent columns of $DG(p)$. We renumber the coordinates of $\mathbb{R}^N$ so that the ones corresponding to these columns become the first k and apply the Implicit Function Theorem, e.g. [53, Theorem 9.28. p.224]. We can conclude that a neighborhood of $p \in H$ is diffeomorphic to an open set in $\mathbb{R}^{N-k}$. Since this holds near each $p \in H$, we conclude that H is a manifold of dimension $N-k$, as wanted[9]. □

The notion of independence defined via transversality is infinitesimal and symmetric w.r.t. permuting g_ks. This is useful in settings where many relations could be discovered, because it is then better to find relations whose first order behavior differs. In cases where guaranteed decrease in dimension is not needed, using restricted syzygies could allow a flexible search for more expressive relations.

B.2 Related Work in Algebraic Ideal Learning

There exists prior work on learning relations carried out in the algebraic setting. Some of this work aims to find simple polynomial relations that hold on the data manifold. The criterion for simplicity is the polynomial degree. Most of these works find either all relations or all relations of a given degree at the same time. This is in contrast to our approach, which finds relations one by one, making it amenable to finding as many relations as desired. Moreover, since we aim to use neural networks for learning relations, the class of polynomial relations is not suitable for our purposes. Hence, we consider a substantially different setting of learning analytic relations. The notion of degree is not defined for analytic functions, making work based on this notion not directly applicable to our setting. In contrast to the algebraic case, our notion of simplicity is implicit in the expressivity of the networks. However, some of the issues that arise in our setting have parallels in the algebraic setting. Below we give a brief overview, pointing out these connections.

The problem of learning relations that hold approximately on a given dataset was brought to the machine learning community by [28]. This paper introduced

[9]Our proof is a variation on Preimage Theorem [54, p. 21], and can also be deduced from it: H is the preimage of $\mathbf{0}$ under the map G, and $\mathbf{0}$ is a regular value because H_{g_j}s are transverse along H, implying the lemma. Though note that the Preimage Theorem is itself a direct consequence of the Implicit Function Theorem.

the algorithm called Vanishing Component Analysis (VCA). The VCA algorithm depends on a parameter ε, and in the limit $\varepsilon = 0$ finds a set of generators for the ideal of polynomials that vanish on a data set. The algorithm builds up this set of generators degree by degree, starting with linear ones (if such exist). For general ε it finds polynomials P_i such that the (Euclidean) norm of the vector of values of P_i is at most ε. The specifics of which of these polynomials it finds depend on the inner workings of the algorithm, which is based on SVD. Being a linear algebra based algorithm, it finds all of those polynomials of the specific degree at the same time.

The same problem of learning relations that hold approximately on a given dataset has been considered before in mathematics literature. [29] introduced p-approximate ideals of accuracy ε, and [30] introduced ε-approximate vanishing ideals; these are two related but different objects aiming at capturing such approximate relations. One of the differences between them is how they normalize the polynomials. The issue at hand is that if one considers only values of a function P on the data set, then a sufficiently small rescaling of *any* P will have values that are small, and so will be deemed an approximate relation. Such a rescaled function will have small values in a lot of places, not just near the data set itself, and so would be a 'trivial' or 'spurious' relation. To avoid this problem of 'spurious approximate relations' one needs to normalize g itself. [29] considers L_p norms of the coefficient vector (hence the p in the name), while [30] considers only the L_2 norm. In [34], it is observed that the VCA algorithm from [28] does produce such 'spurious' small-coefficient polynomials. The authors introduce a modification to VCA in which the values of f on the data set are traded off against its norm, like in [29] and [30]. By default [34] also uses the Euclidean norm of the vector of coefficients of P, or some modification (such as truncation) of it. In a follow up paper [35], the 'norm' is now given by (the norms of) the gradient vectors of P on the data points.

In an alternative formulation of approximate vanishing, which is geometric and avoids the spurious relation problem: one looks for relations whose vanishing loci pass near the data points (rather than the ones which take small values exactly at the data). This approach is more challenging for the algebraic methods but has been attempted in [31, 32]. We note that it is in fact related to gradient normalization, and this relation underlies a part of our approach.

Observe that, while the setting of our work is very different, the need to decide which relations 'hold approximately' on a data set in the presence of rescalings is common to both settings. The norm of the coefficient vector is obviously unavailable in our setting. On the other hand, we employ a transversality framework for multiple relations, which places an emphasis on the on-manifold gradients as its core principle. As a special case, this produces the 'singleton-transversality' approach: comparing on-manifold values of g to the on-manifold gradients of g (similar to one used in [35] except that in our case it is formulated as a component of NN loss). More precisely, we use the ratio of the absolute value of g $\big(\text{i.e.}|g(\tau)|\big)$ to the norm of the gradient of g at a data point τ $\big(\text{i.e.}\|\nabla_{\mathcal{T}}(g)|_\tau\|\big)$: $d_g(\tau) = \frac{|g(\tau)|}{\|\nabla_{\mathcal{T}}(g)|_\tau\|}$. This has the following interpretation: the norm of the gradient measures the maximal rate of change of the linearization of g at τ, meaning the maximal 'slope'. So $d_g(\tau)$ is the distance from τ to the nearest point where this linearization vanishes ($d_g(\tau)=$ height/slope = distance). This serves as a proxy for the distance from τ to the vanishing locus of g itself. In this way, our approach unifies the gradient based and distance-to-vanishing-set based measures of approximate vanishing that have appeared in the prior work cited above. In addition to gradient measures, we also assess the vanishing of g by comparing values on-manifold and off-manifold, which is related to gradients in spirit (gradients tell you how much the value nearby differs from the values at the point you start with), but requires only evaluations of the relation itself. In our case we use this comparison to formulate a stopping criterion for learning relations: stopping when on-manifold values are sufficiently smaller than off-manifold ones.

Learning of algebraic manifolds has been considered in learning theory works, e.g. [38, 39]. It would be interesting to investigate whether analytic manifolds that we consider, which are less rigid than algebraic ones (but more rigid than smooth, as illustrated by Theorem 3.1, for example), give another reasonable alternative.

On the applications side, [36] search for a low-dimensional manifold (variety) cut out by polynomials of bounded degree, and show a proof of concept for data modeling in a robotics setting. VCA algorithm has been applied to pattern recognition by several works, e.g. [37, 55].

B.3 Further Details, Results, Illustrations for Evaluating AML

B.3.1 Further Algorithmic and Implementation Details

Here, we start by giving a further mathematical interpretation for our implementation of AML with transversality, i.e. the more detailed motivation for Equation 2 in the main paper. Then, we give a summary of implementation details for AML.

Motivation for our approach to computing transversality:

Recall that, to obtain g_k that is transverse to $g_1,..,g_{k-1}$ (Definition 3.4), we compute gradients of each $g_1,...g_{k-1}$ w.r.t the input. For example, for g_1 we denote this as $v_1 = \nabla_\tau(g_1)|_\tau$. Making g_k transverse to $g_1,...g_{k-1}$ means ensuring that v_k is linearly independent of $v_1,...,v_{k-1}$. Hence, we need to choose a (computationally tractable) numerical measure of this linear independence. To that end, we design our measure to maximize the angles between v_k and all the previous $v_1,..,v_{k-1}$. When the number of relations is lower than the dimensionality of the ambient space ($k \leq N$), this is maximized by any vector that is perpendicular to all the previous ones. Such a measure also encourages transversality of subsets of relations. Furthermore, we want to discourage small angles, which can be achieved by a measure that involves a product of pairwise measures.

Hence, we use the product of sines of pairwise angles as our measure of transversality (with log for computational stability):

$$L_{tr}(g_k) = d_{g_k}(\tau) - \log \|v_k\| - \log \prod_{j=1}^{k-1} \sin^2(\theta_{v_j,v_k}) \tag{2}$$

The last two terms give (up to weighting constants) the log of products of areas of parallelograms formed by v_k and each of the previous $v_1, ..., v_{k-1}$. In principle, the k-dimensional volume of the parallelepiped spanned jointly by $v_1, ..., v_k$ could serve as a measure of transversality. It could be computed as a product of singular values of the matrix with columns $v_1, ..., v_k$, e.g. requiring SVD. However, it would not be suitable for low-dimensional cases ($N < k$), since this volume would be 0.

AML implementation details:

We implemented AML in PyTorch [56]. To represent g_k relations and restricted syzygies $\mathfrak{f}$ we used fully connected networks with 3 hidden layers. For our experiments we used a setup that starts with small networks (e.g. 3 layers, 4 hidden units per layer for g_1) and doubles the number of hidden units for subsequent g_ks (e.g. 8 hidden units per layer for g_2, 16 for g_3, and so on, with a maximum of 256). For syzygies we started with 32 nodes per layer. We also experimented with simply having 32 units in all g_ks, but did not see a significant difference.

The first term in Equations 1,2,3 in the main paper dictates whether g_k is close to 0 for on-manifold data. Since there are no further weighting terms in these equations, we note that one needs to take care that the other terms do not overwhelm the contribution from the 1st term. For this, we clip the loss from other components if it is more than twice the magnitude of the 1st term. This simply means: the overall loss encourages g_k outputs to be small for on-manifold data, regardless of what other parts dictate.

When learning with transversality: we usually used a fixed weight $\beta = 1e3$ for the transversality terms instead of loss clipping ($\beta \in [1e2, 1e4]$ worked as well). When using the variant with restricted syzygies: we always included the second term from Equation 3, i.e $\nabla_{g_k} \left[|\mathfrak{f}(\tau_{off}, g_1, ..., g_k)| \right]$, even if $\mathfrak{f}$ did not reach output close to zero during its training. This implies that we implemented a soft (incremental) version, rather than mandating syzygies to be always learned exactly.

In theory, g_k being 0 for on-manifold data means getting an output of exactly 0. But in practice we need to choose a way to tell whether the output of g_k or $\mathfrak{f}$ is essentially 0 for all practical purposes. So, as our stopping criterion, we look at the difference between on-manifold and off-manifold outputs. When the mean absolute value of off-manifold values is more than 5 times that of on-manifold values: we say we are done learning g_k (or $\mathfrak{f}$). We record the mean outputs for on- and off-manifold data when we save the learned relations. With that, when AML relations are loaded for subsequent use, we can check if the output of g_k is 'close to 0': simply check whether it is close to the expected on-manifold output magnitude. To make this more concrete, below is an example of such expected values, printed when a learned set of relations is loaded:

```
20:52:07 AML loaded 2 relations, 0 syzygies, on/off means:
20:52:07 [0.0047 0.0029]
20:52:07 [0.0518 0.025 ]
```

Top row shows mean expected on-manifold output for g_1, g_2. Bottom row shows expected off-manifold output means. Note that off-manifold values are $\approx$10 times higher than on-manifold ones. The relative magnitude is what matters, not whether the values are 'small' in some absolute sense.

B.3.2 Distortion Measure

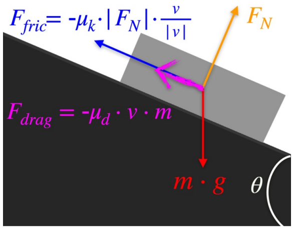
We use the following measure of distortion of a map f: take pairs of inputs τ_1, τ_2 and the corresponding outputs $f(\tau_1), f(\tau_2)$, then compute distortion coefficient $\rho_{distort}$:

$$\rho_{distort}(f)|_{\tau_1,\tau_2} = \log \frac{d_{L2}\big(f(\tau_1), f(\tau_2)\big)}{d_{L2}(\tau_1,\tau_2)}.$$

Here, d_{L2} is the Euclidean distance. A map that yields low variance of these coefficients would better preserve the geometry of the domain (up to overall scale). This measure is related to approaches in [21, 22] and has the same desirable properties as σ-distortion described in [21], but in log space. Note that, if h is a composition $h = f \circ g$, then: $\rho_{distort}(h)|_{\tau_1,\tau_2} = \rho_{distort}(g)|_{\tau_1,\tau_2} + \rho_{distort}(f)|_{g(\tau_1),g(\tau_2)}$. This additivity of individual ρs is appealing. It makes the variance measure defined from them extendable to a distortion covariance measure for composable maps (with inverse maps maximally anti-correlated). This measure is also related to Hilbert distance on rays, which appears in the work of Birkhoff on Perron-Frobenius theory [57]. We plan to further investigate this in future work.

B.3.3 Additional Illustrations of AML Results

Here, we provide additional illustrations of learning relations with AML in the *block-on-incline* domain (Figure 1). On-manifold data is comprised of noisy position and velocity observations from simulation of a block with mass 1kg sliding down an incline. Figure 2 illustrates learning with transversality. Figure 3 shows the corresponding results when using syzygies. True dynamics and learned relations are visualized using phase space plots: arrows indicate change in position & velocity after 1*sec* of sliding (scaled to fit).


$F_{fric} = -\mu_k \cdot |F_N| \cdot \frac{v}{|v|}$
F_N
$F_{drag} = -\mu_d \cdot v \cdot m$
$m \cdot g$
θ
Figure 1: Block-on-incline

For each row in Figures 2 & 3 (on the next page) we show: on-manifold data visualizing the actual dynamics; part of the space where the intersection $g_1 \cap g_2 \cap g_3$ of the learned relations holds (i.e. all $g_1, ..., g_k$ output values close to 0); individual preimages of 0 for each relation separately. Top row in Figures 2 & 3 shows training on a limited range when a block slides on a 45° incline. The intersection $g_1 \cap g_2 \cap g_3$ generalizes far beyond the training data ranges. It misses only the part capturing stopping at the end of the incline (blue arrows in top right of 'on-manifold test data' plot), which is not possible to extrapolate, since training does not contain examples of running into the end of the incline. Middle row shows results for a 35° incline with high friction coefficient $\mu_k = 0.8$. Bottom row shows results when using a high drag coefficient $\mu_d = 2.0$; in this case we train on a range of incline angles $\theta \in [\frac{\pi}{20}, \frac{\pi}{2.5}]$ and visualize results for a 10° incline.

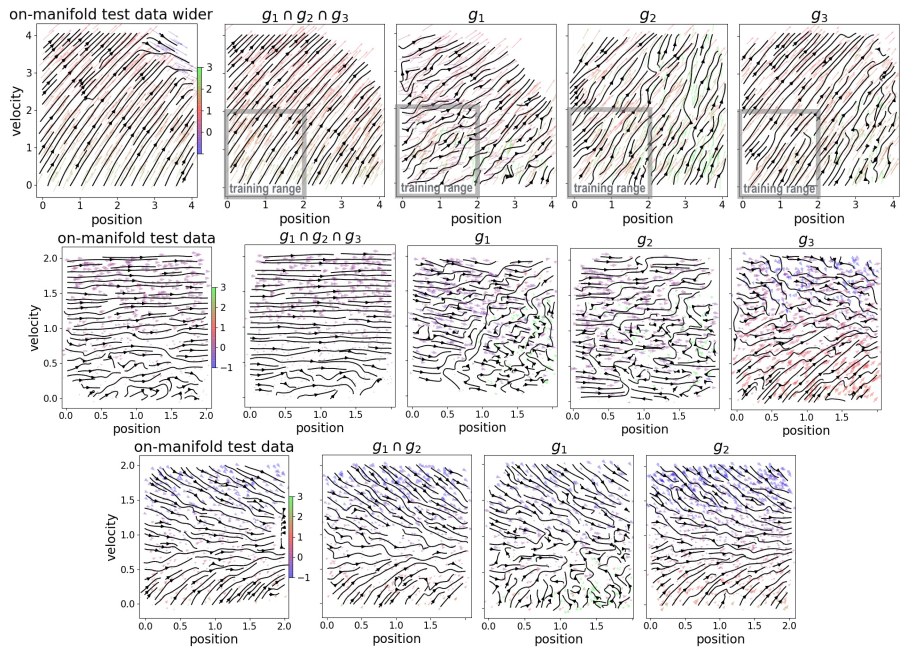


Figure 2: Illustrations of learning relations on the block-on-incline domain with transversality. This is a more detailed version of Figure 6 from the main paper.

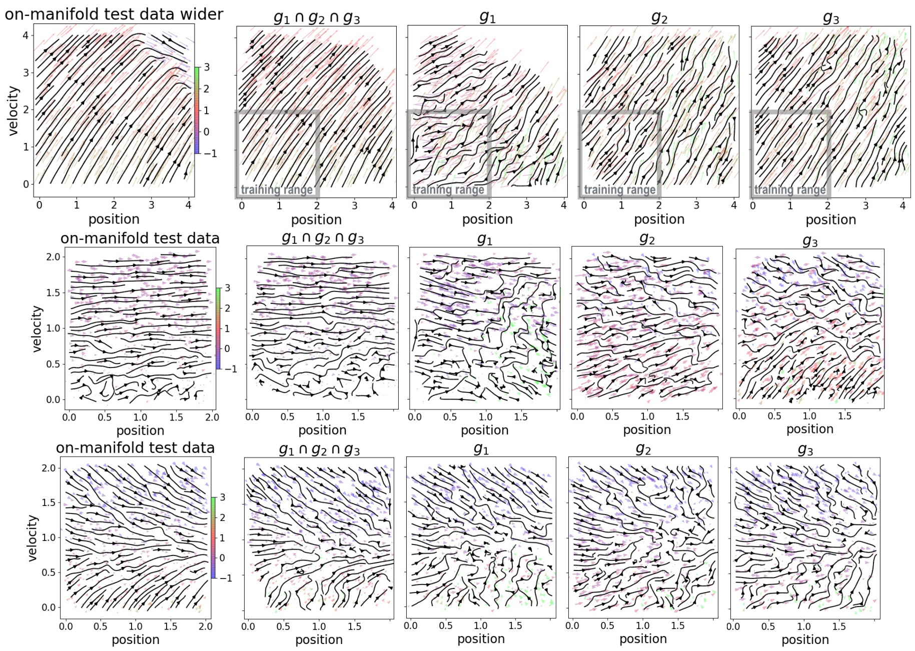


Figure 3: Illustrations of learning relations on the block-on-incline domain with syzygies.

Overall, both training with transversality and with syzygies gives us the ability to generalize and capture non-trivial dynamics. We can see that intersections $g_1 \cap g_2 \cap g_3$ look very similar to on-manifold phase space plots, which means AML correctly captures information about the data manifold. As expected, zero-level sets of individual images do not resemble on-manifold plots, since individual relations g_k capture different parts/aspects of on-manifold data properties.

References

[1] L. Wiskott and T. J. Sejnowski, "Slow feature analysis: Unsupervised learning of invariances," *Neural computation*, vol. 14, no. 4, pp. 715–770, 2002.

[2] A. Anand, E. Racah, S. Ozair, Y. Bengio, M.-A. Côté, and R. D. Hjelm, "Unsupervised state representation learning in ATARI," in *Advances in Neural Information Processing Systems 32*, 2019, pp. 8766–8779.

[3] T. Lesort, N. Díaz-Rodríguez, J.-F. Goudou, and D. Filliat, "State representation learning for control: An overview," *Neural Networks*, vol. 108, pp. 379–392, 2018.

[4] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, J. Pineau *et al.*, "An introduction to deep reinforcement learning," *Foundations and Trends® in Machine Learning*, vol. 11, no. 3-4, pp. 219–354, 2018.

[5] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[6] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[7] N. Sünderhauf, O. Brock, W. Scheirer, R. Hadsell, D. Fox, J. Leitner, B. Upcroft, P. Abbeel, W. Burgard, M. Milford, and P. Corke, "The limits and potentials of deep learning for robotics," *The International Journal of Robotics Research*, vol. 37, no. 4-5, pp. 405–420, 2018.

[8] K. Greff, R. L. Kaufman, R. Kabra, N. Watters, C. Burgess, D. Zoran, L. Matthey, M. Botvinick, and A. Lerchner, "Multi-object representation learning with iterative variational inference," in *International Conference on Machine Learning*, 2019.

[9] A. Heljakka, A. Solin, and J. Kannala, "Pioneer networks: Progressively growing generative autoencoder," in *Asian Conference on Computer Vision.* Springer, 2018, pp. 22–38.

[10] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[11] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2019.

[12] B. Calli, A. Singh, A. Walsman, S. Srinivasa, P. Abbeel, and A. M. Dollar, "The YCB object and model set: Towards common benchmarks for manipulation research," in *International Conference on Advanced Robotics (ICAR).* IEEE, 2015, pp. 510–517.

[13] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[14] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, "beta-vae: Learning basic visual concepts with a constrained variational framework." in *International Conference on Learning Representations*, 2017.

[15] L. Yingzhen and S. Mandt, "Disentangled sequential autoencoder," in *International Conference on Machine Learning*, 2018, pp. 5670–5679.

[16] E. Crawford and J. Pineau, "Spatially invariant unsupervised object detection with convolutional neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 3412–3420.

[17] S. M. A. Eslami, N. Heess, T. Weber, Y. Tassa, D. Szepesvari, k. kavukcuoglu, and G. E. Hinton, "Attend, infer, repeat: Fast scene understanding with generative models," in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds., 2016.

[18] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove, "DeepSDF: Learning continuous signed distance functions for shape representation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 165–174.

[19] V. R. Kompella, M. Luciw, and J. Schmidhuber, "Incremental slow feature analysis: Adaptive and episodic learning from high-dimensional input streams," *arXiv preprint arXiv:1112.2113*, 2011.

[20] S. Akbulut and H. King, "On approximating submanifolds by algebraic sets and a solution to the nash conjecture," *Inventiones mathematicae*, vol. 107, no. 1, pp. 87–98, 1992.

[21] L. Chennuru Vankadara and U. von Luxburg, "Measures of distortion for machine learning," in *Advances in Neural Information Processing Systems 31*, 2018, pp. 4886–4895.

[22] Y. Bartal, N. Fandina, and O. Neiman, "Dimensionality reduction: theoretical perspective on practical measures," in *Advances in Neural Information Processing Systems*, 2019, pp. 10 576–10 588.

[23] OpenAI, *Roboschool*, 2017. [Online]. Available: https://github.com/openai/roboschool

[24] Y. Tassa, Y. Doron, A. Muldal, T. Erez, Y. Li, D. d. L. Casas, D. Budden, A. Abdolmaleki, J. Merel, A. Lefrancq *et al.*, "Deepmind control suite," *arXiv preprint arXiv:1801.00690*, 2018.

[25] B. Ellenberger, *Pybullet gymperium*, 2018. [Online]. Available: https://github.com/benelot/pybullet-gym

[26] A. Raffin, A. Hill, R. Traoré, T. Lesort, N. Díaz-Rodríguez, and D. Filliat, "S-rl toolbox: Environments, datasets and evaluation metrics for state representation learning," *arXiv preprint arXiv:1809.09369*, 2018.

[27] M. Jaggi, "Revisiting frank-wolfe: Projection-free sparse convex optimization." in *Proceedings of the 30th international conference on machine learning*, no. CONF, 2013, pp. 427–435.

[28] R. Livni, D. Lehavi, S. Schein, H. Nachliely, S. Shalev-Shwartz, and A. Globerson, "Vanishing component analysis," in *International Conference on Machine Learning*, 2013, pp. 597–605.

[29] T. Sauer, "Approximate varieties, approximate ideals and dimension reduction," *Numerical Algorithms*, vol. 45, no. 1-4, pp. 295–313, 2007.

[30] D. Heldt, M. Kreuzer, S. Pokutta, and H. Poulisse, "Approximate computation of zero-dimensional polynomial ideals," *Journal of Symbolic Computation*, vol. 44, no. 11, pp. 1566–1591, 2009.

[31] C. Fassino, "Almost vanishing polynomials for sets of limited precision points," *Journal of symbolic computation*, vol. 45, no. 1, pp. 19–37, 2010.

[32] C. Fassino and M.-L. Torrente, "Simple varieties for limited precision points," *Theoretical Computer Science*, vol. 479, pp. 174–186, 2013.

[33] H. Kera and Y. Hasegawa, "Noise-tolerant algebraic method for reconstruction of nonlinear dynamical systems," *Nonlinear Dynamics*, vol. 85, no. 1, pp. 675–692, 2016.

[34] H. Kera and Y. Hasegawa, "Spurious vanishing problem in approximate vanishing ideal," *IEEE Access*, vol. 7, pp. 178 961–178 976, 2019.

[35] H. Kera and Y. Hasegawa, "Gradient boosts the approximate vanishing ideal," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019.

[36] R. Iraji and H. Chitsaz, "Principal variety analysis," in *Conference on Robot Learning*, 2017, pp. 97–108.

[37] H. Yan, Z. Yan, G. Xiao, W. Wang, and W. Zuo, "Deep vanishing component analysis network for pattern classification," *Neurocomputing*, vol. 316, pp. 240–250, 2018.

[38] E. Hazan and T. Ma, "A non-generative framework and convex relaxations for unsupervised learning," in *Advances in Neural Information Processing Systems*, 2016, pp. 3306–3314.

[39] A. Globerson, R. Livni, and S. Shalev-Shwartz, "Effective semisupervised learning on manifolds," in *Conference on Learning Theory*, 2017, pp. 978–1003.

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

[41] P. Shi, *SPAIR in Pytorch*, 2020. [Online]. Available: https://github.com/yonkshi/SPAIR_pytorch

[42] P. Shi, “Faster unsupervised object detection for symbolic representation,” *Master’s Thesis, KTH Royal Institute of Technology*, 2020.

[43] J. Frisch, “Points de platitude d’un morphisme d’espaces analytiques complexes,” *Inventiones mathematicae*, vol. 4, no. 2, pp. 118–138, 1967.

[44] A. Hatcher, *Algebraic Topology.* Cambridge University Press, 2002.

[45] F. Acquistapace, F. Broglia, and J. F. Fernando, “Some results on global real analytic geometry,” in *Ordered Algebraic Structures and Related Topics: International Conference on Ordered Algebraic Structures and Related Topics, October 12-16, 2015, Centre International de Rencontres Mathématiques (CIRM), Luminy, France*, vol. 697. American Mathematical Soc., 2017, p. 1.

[46] H. Cartan, “Variétés analytiques réelles et variétés analytiques complexes,” *Bulletin de la Société Mathématique de France*, vol. 85, pp. 77–99, 1957.

[47] E. Bierstone and P. D. Milman, “Semianalytic and subanalytic sets,” *Publications Mathématiques de l’IHÉS*, vol. 67, pp. 5–42, 1988.

[48] S. Łojasiewicz, *Introduction to complex analytic geometry.* Springer, 1991.

[49] D. Eisenbud, *Commutative Algebra: with a view toward algebraic geometry.* Springer-Verlag, 1995, vol. 150.

[50] D. S. Dummit and R. M. Foote, *Abstract algebra.* Wiley Hoboken, 2004, vol. 3.

[51] T. Stacks project authors, “The stacks project,” https://stacks.math.columbia.edu, 2020.

[52] R. C. Gunning and H. Rossi, *Analytic functions of several complex variables.* Prentice-Hall, 1965.

[53] W. Rudin, *Principles of mathematical analysis.* McGraw-hill New York, 1976.

[54] V. Guillemin and A. Pollack, *Differential topology.* Prentiee-Hall, 1974.

[55] Y.-G. Zhao and Z. Song, “Hand posture recognition using approximate vanishing ideal generators,” in *2014 IEEE International Conference on Image Processing (ICIP).* IEEE, 2014, pp. 1525–1529.

[56] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “Pytorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems 32*, 2019, pp. 8024–8035.

[57] G. Birkhoff, “Extensions of Jentzsch’s theorem,” *Transactions of the American Mathematical Society*, vol. 85, no. 1, pp. 219–227, 1957.