Physical Derivatives: Computing policy gradients by physical forward-propagation

Anonymous authors
Paper under double-blind review

Abstract

Model-free and model-based reinforcement learning are two ends of a spectrum. Learning a good policy without a dynamic model can be prohibitively expensive. Learning the dynamic model of a system can reduce the cost of learning the policy, but it can also introduce bias if it is not accurate. We propose a middle ground where instead of the transition model, the sensitivity of the trajectories with respect to the perturbation of the parameters is learned. This allows us to predict the local behavior of the physical system around a set of nominal policies without knowing the actual model. We assay our method on a custom-built physical robot in extensive experiments and show the feasibility of the approach in practice. We investigate potential challenges when applying our method to physical systems and propose solutions to each of them.

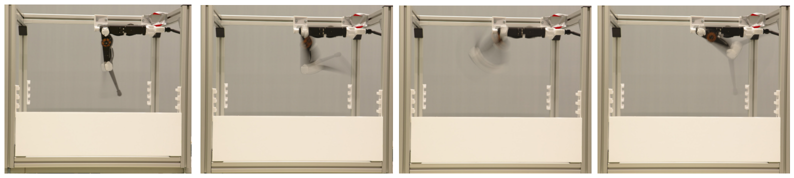

(a) (b) (c) (d)

Figure 1: Physical finger platform in action with different policies.

1 Introduction

Traditional reinforcement learning crucially relies on *reward* Sutton & Barto (2018). However, reward binds the agent to a certain task for which the reward represents success. Aligned with the recent surge of interest in unsupervised methods in reinforcement learning (Baranes & Oudeyer, 2013; Bellemare et al., 2016; Gregor et al., 2016; Houthooft et al., 2016; Gupta et al., 2018; Hausman et al., 2018; Pong et al., 2019; Laskin et al., 2020; 2021; He et al., 2021) and previously proposed ideas (Schmidhuber, 1991a; 2010), we argue that there exist properties of a dynamical system which are not tied to any particular task, yet highly useful, leveraging them can help solve other tasks more efficiently. This work focuses on the sensitivity of the produced trajectories of the system with respect to the policy so-called *Physical Derivatives*. The term *physical* comes from the fact that it uses the physics of the system rather than any idealized model. We learn a map from the directions in which policy parameters change to the directions in which every state of the trajectory changes. In general, our algorithm learns the Jacobian matrix of the system at every time step through the trajectory. The training phase consists of physically calculating directional derivatives by the finite difference after applying perturbed versions of a nominal policy (a.k.a. controller). After training, the learned directional derivatives are used to guide the controller to achieve the desired behaviour. Due to the difficulty of computing the Jacobian matrix by the finite difference in higher dimensions, we use random controllers joint with probabilistic learning methods to obtain a robust estimate of the Jacobian matrix at each instant of time along a trajectory. The generalization to unseen perturbations is possible because the trajectories produced by physical systems live on an intrinsically low-dimensional manifold and change slowly with respect to perturbations in the system (Koopman, 1931). This assumption holds as long as the system is not chaotic or close to a bifurcation condition (Khalil, 2002) (See Appendix A.4 for a detailed literature review).

Preliminaries. We consider a closed-loop dynamical system represented by the state vector $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^d$ and the policy function $\mathbf{u} = \pi(\mathbf{x}; \boldsymbol{\theta})$ that emits the $q-$dimensional control signal

$\mathbf{u} \in \mathcal{U} \subseteq \mathbb{R}^q$. Let $r : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ denote the reward function and $R : \Pi(\Theta) \to \mathbb{R}$ be the cumulative reward (*return*). For parametric policies, the space of feasible parameters Θ has a one-to-one correspondence to the policy space Π. The agent who takes on the policy π from state $\mathbf{x}_0$ produces the trajectory $\mathcal{T} \in \mathbb{T}$ where $\mathbb{T}$ is the space of possible trajectories. The expected return becomes a function of the policy as $J(\pi_{\boldsymbol{\theta}}) = \mathbb{E}_{\mathcal{T}}\{R(\mathcal{T})\}$ where the expectation is taken with respect to the probability distribution $P(\mathcal{T}|\pi_{\boldsymbol{\theta}})$. Traditionally in reinforcement learning, the goal is to perturb the policy to improve the expected return:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \left. \frac{\partial J(\pi_{\boldsymbol{\theta}})}{\partial \boldsymbol{\theta}} \right|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t} . \tag{1}$$

The gradient $\partial J(\pi_{\boldsymbol{\theta}})/\partial \boldsymbol{\theta}$ can be written as an integral

$$\frac{\partial J(\pi_{\boldsymbol{\theta}})}{\partial \boldsymbol{\theta}} = \int_{\mathbb{T}} \frac{\partial p(\mathcal{T}|\pi_{\boldsymbol{\theta}})}{\partial \boldsymbol{\theta}} R(\mathcal{T}) \, \mathrm{d}\mathcal{T} \tag{2}$$

which is hard compute in practice. In model-free RL, the policy is updated so that the mode of $p(\mathcal{T}|\pi_{\boldsymbol{\theta}})$ aligns with the modes of $R(\mathcal{T})$. In model-based RL, the dynamical model that generates the trajectory $\mathcal{T}$ is learned and then used to estimate $\frac{\partial J(\pi_{\boldsymbol{\theta}})}{\partial \boldsymbol{\theta}}$. In this work, we take a middle-ground approach to estimate an in-between quantity and show how the estimated map can guide the policy towards desired behaviour.

What is Physical Derivative. In this paper, we investigate the feasibility of learning a less explored unsupervised quantity, the so-called *Physical Derivative* which is computed directly from the physical system. In abstract terms, we perturb the policy and learn the effect of its perturbation on the resulting trajectory. The difference from traditional RL, whose algorithms are based on Equation (1), is the absence of a specified reward function. Instead, we generate samples from $\partial p(\mathcal{T}|\pi_{\boldsymbol{\theta}})/\partial \boldsymbol{\theta}$ of Equation (2) that makes it possible to compute $\partial J(\pi_{\boldsymbol{\theta}})/\partial \boldsymbol{\theta}$ for an arbitrary return function R. If the exact model of the system is known, control theory has a full set of tools to intervene on the system with stability and performance guarantees. When the system is unknown, one could identify the system as a preliminary step followed by a normal control synthesis process from control theory (Ljung, 2001). Otherwise, the model and the policy can be learned together in a model-based RL (Sutton, 1996) or in some cases adaptive control (Sastry & Bodson, 2011). We argue that learning physical derivatives is a middle ground. It is not model-based in the sense that it does not assume knowing the exact model of the system. For example, assume we are interested in going from the current trajectory $\mathcal{T}(\boldsymbol{\theta})$ to the target trajectory $\mathcal{T}^*$. The distance between these trajectories is reduced by perturbing the policy parameters in the direction $-\partial\|\mathcal{T}(\boldsymbol{\theta}) - \mathcal{T}^*\|/\partial \boldsymbol{\theta}$. This direction is already available since we have direct access to $\partial \mathcal{T}(\boldsymbol{\theta})/\partial \boldsymbol{\theta}$ as a physical derivative.

Our contributions— In summary, the key contributions of the current paper are as follows:

- A method to generate training pairs to learn the map from the policy perturbations to the resulting changes in the trajectories. Learning the mentioned map as a probabilistic function and showing that it generalizes to unseen perturbations in the policy. Using the inverse of the above map to perturb the policy in the desired direction to achieve certain goals without conventional RL methods.
- Provide a detailed description of the use of physical derivatives in different applications, i.e., *adversarial system identification*, *safety*, and *robust control* in Appendix A.3.
- Use a physical custom-built robotic platform to test the method and propose solutions to deal with the inherent issues of the physical system to ensure the practicality of the method (see Figure 1 for images of the platform and Appendix A.1 for technical details). The supplementary materials for the paper, including code, and the videos of the robot in action, can be found in `https://sites.google.com/view/physicalderivatives/`.

2 Estimating Physical Derivatives

We are interested in $\partial \mathcal{T}/\partial \boldsymbol{\theta}$ which denotes how a small change in the parameters $\boldsymbol{\theta}$ of the controller results in a different trajectory produced by the system. We normally consider a finite period of time $[0, T]$ and the trajectory is an ordered list of states $\mathcal{T} = [\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_T]$ where the subscript shows the time step. Therefore, having $\partial \mathcal{T}/\partial \boldsymbol{\theta}$ is equivalent to having $\partial \mathbf{x}_t/\partial \boldsymbol{\theta}$ for every $t \in \{1, \ldots, T\}$. Notice that the initial state $\mathbf{x}_0$ is chosen by us. Hence we can see it either as a constant or as a changeable parameter in $\boldsymbol{\theta}$. We kept it fixed in our experiments.

Assume $\mathbf{x}_t \in \mathbb{R}^d$ and $\boldsymbol{\theta} \in \mathbb{R}^m$. Hence, $\nabla_{\boldsymbol{\theta}} \mathbf{x}_t = \partial \mathbf{x}_t / \partial \boldsymbol{\theta} \in \mathbb{R}^{d \times m}$ where the t^{th} row of this matrix is $\nabla_{\boldsymbol{\theta}} x_{it} = (\partial x_{it} / \partial \boldsymbol{\theta})^{\mathsf{T}} \in \mathbb{R}^m$ showing how the i^{th} dimension of the state vector changes in response to a perturbation in $\boldsymbol{\theta}$. The directional derivative of x_{it} in the direction $\delta\boldsymbol{\theta}$ is defined as

$$\nabla_{\boldsymbol{\theta}}^{\delta\boldsymbol{\theta}} x_{it} = \langle \nabla_{\boldsymbol{\theta}} x_{it}, \frac{\delta\boldsymbol{\theta}}{|\delta\boldsymbol{\theta}|} \rangle. \tag{3}$$

If equation 3 is available for m linearly independent and orthonormal directions, $\{\delta\boldsymbol{\theta}^{(1)}, \delta\boldsymbol{\theta}^{(2)}, \ldots, \delta\boldsymbol{\theta}^{(m)}\}$, the directional derivative along an arbitrary $\delta\boldsymbol{\theta}$ can be approximated by

$$\nabla_{\boldsymbol{\theta}}^{\delta\boldsymbol{\theta}} x_{it} = \sum_{j=1}^{m} c_j \langle \nabla_{\boldsymbol{\theta}} x_{it}, \delta\boldsymbol{\theta}^{(j)} \rangle \tag{4}$$

where $c_j = \langle \delta\boldsymbol{\theta}, \delta\boldsymbol{\theta}^{(j)} \rangle$ is the coordinates of the desired direction in the coordinate system formed by the orthonormal bases.

In practice, m directions $\delta\boldsymbol{\theta}^{(j)}$ can be randomly chosen or can be along some pre-defined axes of the coordinate system. To compute $\langle \nabla_{\boldsymbol{\theta}} x_{it}, \delta\boldsymbol{\theta}^{(j)} \rangle$, the nominal policy parameters $\boldsymbol{\theta}$ are perturbed by $\delta\boldsymbol{\theta}^{(j)}$ as $\boldsymbol{\theta}^{(j)} \leftarrow \boldsymbol{\theta} + \delta\boldsymbol{\theta}^{(j)}$ and the derivative is computed as

$$\langle \nabla_{\boldsymbol{\theta}} x_{it}, \delta\boldsymbol{\theta}^{(j)} \rangle = \lim_{h \to 0} \frac{x_{it}(\boldsymbol{\theta} + h\delta\boldsymbol{\theta}^{(j)}) - x_{it}(\boldsymbol{\theta})}{h}. \tag{5}$$

This quantity is often approximated by finite difference where h takes a small nonzero value. By perturbing the parameters $\boldsymbol{\theta}$ along m orthonormal directions $\delta\boldsymbol{\theta}^{(j)}$ and computing the approximate directional derivative by equation 5, $\nabla_{\boldsymbol{\theta}}^{\delta\boldsymbol{\theta}} x_{it}$ can be computed along every arbitrary direction $\delta\boldsymbol{\theta}$, meaning that we can compute $\nabla_{\boldsymbol{\theta}} x_{it}$ by evaluating it along any direction, which is the aim of this paper. In the matrix form for $\mathbf{x} \in \mathbb{R}^d$, we can compute $\nabla_{\boldsymbol{\theta}}^{\delta\boldsymbol{\theta}^{(j)}} \mathbf{x} = [\nabla_{\boldsymbol{\theta}}^{\delta\boldsymbol{\theta}^{(j)}} x_1, \nabla_{\boldsymbol{\theta}}^{\delta\boldsymbol{\theta}^{(j)}} x_1, \ldots, \nabla_{\boldsymbol{\theta}}^{\delta\boldsymbol{\theta}^{(j)}} x_d]^{\mathsf{T}}$ in a single run by computing equation 5 for all d dimensions of the states. Let's define

$$\Delta_{\boldsymbol{\theta}} \mathbf{x} \triangleq [\nabla_{\boldsymbol{\theta}}^{\delta\boldsymbol{\theta}^{(1)}} \mathbf{x}, \nabla_{\boldsymbol{\theta}}^{\delta\boldsymbol{\theta}^{(2)}} \mathbf{x}, \ldots, \nabla_{\boldsymbol{\theta}}^{\delta\boldsymbol{\theta}^{(m)}} \mathbf{x}] \tag{6}$$

where $\Delta_{\boldsymbol{\theta}} \mathbf{x} \in \mathbb{R}^{d \times m}$ and let $\Lambda = [\delta\boldsymbol{\theta}^{(1)}, \delta\boldsymbol{\theta}^{(2)}, \ldots, \delta\boldsymbol{\theta}^{(m)}]$. Therefore, if $\Delta_{\boldsymbol{\theta}}^{\delta\boldsymbol{\theta}} \mathbf{x}$ shows the directional derivative of $\mathbf{x}$ along $\delta\boldsymbol{\theta}$, we can write it as:

$$\nabla_{\boldsymbol{\theta}}^{\delta\boldsymbol{\theta}} \mathbf{x} = \Delta_{\boldsymbol{\theta}} \mathbf{x} (\Lambda^{\mathsf{T}} \delta\boldsymbol{\theta}) \tag{7}$$

which is a vectoral representation of Equation (3). Even though the linear formula of Equation (7) requires only m directional derivatives, it has two major downsides. First, it does not give a clear way to incorporate more than m training directional physical derivatives. Second, the linear approximation remains valid only for very small $\delta\boldsymbol{\theta}$. We propose to use Gaussian Process (GP) as a nonlinear probabilistic function approximator (Rasmussen, 2003) to capture the maps $\hat{g}_t$ defined as

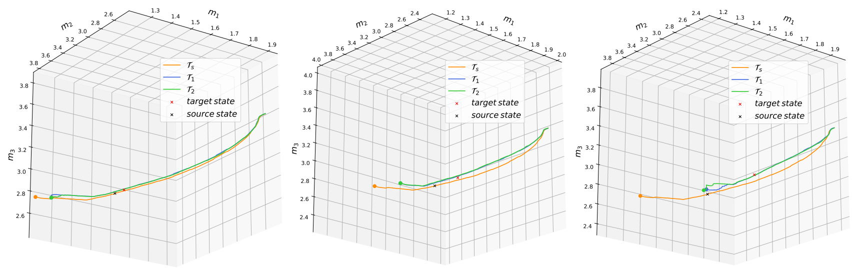
$$\hat{g}_t : \Theta \to \mathcal{X} \tag{8}$$

$$\hat{g}_t(\delta\boldsymbol{\theta}) = \delta\mathbf{x} \tag{9}$$

where subscript t shows the function that maps $\delta\boldsymbol{\theta}$ to the change of the states $\delta\mathbf{x}_t$ at time step t. We considered distinct functions for every time step. Taking into account the commonality among the function approximators corresponding to different time steps is deferred to future research. The required training data to learn these maps is produced by perturbing the parameters of the controller and recording the produced trajectories. We propose two perturbation methods called *Gaussian* and *Uniform* as illustrated in Figure 12. The effect of each of these sampling strategies is presented in Section 3. A major blockage against estimating physical derivatives is the inherent noise of the physical systems that may be mixed with the intentional perturbations which are applied to collect data to estimate physical derivatives. In Appendix A.5, we describe this issue concretely and elaborate on our solution to deal with it.

3 Experiments

In this section, we show how physical derivatives can be estimated in practice through several experiments. Notice that our work is different from computing gradients around the working point of a system by finite-difference. We aim to collect samples from such gradients by perturbing a grid of nominal values of the policy parameters and then generalize to unseen perturbations by Gaussian process as a probabilistic regression method. The experiments are designed to show each challenge separately and the efficacy of our proposed solution to it. These experiments subsume *linear open-loop controller*, *Nonlinear open-loop controller*, and *Feedback controller* elaborated thoroughly in Appendix A.7. Lastly, in the following section, we propose a *zero-shot planning task* and use our method to address it. Due to space constraints, details of the physical platform can be found in Appendix A.1. See[1] for videos of the robot while collecting data for different experiments and more backup materials.



(a) Short distance target (b) Medium distance target (c) Long distance target

Figure 2: Zero-shot planning with constraint satisfaction. The orange trajectory is the source produced by the nominal controller. The green and blue are two sampled trajectories produced by perturbing k_p to k_p^* by Equation (10).

Zero-shot planning task Our previous experiments showed that learning the physical derivative map is feasible for various types of controllers. In this section, we demonstrate an example of a constraint satisfaction task by means of the physical derivative map. In this experiment, the superscript (s) corresponds to the nominal trajectory, which is called *source*. Assume the system is controlled by a PD controller to reach a target state $\mathbf{x}^*$, i.e., the control torques are designed as $\mathbf{u} = k_p^{(s)}(\mathbf{x} - \mathbf{x}^*) + k_d^{(s)}\dot{\mathbf{x}}$. The controller does a decent job of reaching the target state given reasonable values for k_p and k_d. However, such a controller does not give us a clear way to shape the trajectory that starts from $\mathbf{x}_\circ$ and ends at $\mathbf{x}^*$. Assume it is desired that the nominally controlled trajectory $\mathcal{T}^{(s)}$ passes through an intermediate state $\mathbf{x}_t^*$ at time t on its way towards the target state $\mathbf{x}^*$ (we can equally assume that the system must avoid some regions of the state space because of safety reasons). The solution with physical derivatives is as follows . Assume $k_d^{(s)}$ is fixed and only $k_p^{(s)}$ is changeable. If the physical derivatives map is available, we have access to $\hat{g}_t(k_p^* - k_p^{(s)}) = (\mathbf{x}_t^* - \mathbf{x}_t^{(s)})/(k_p^* - k_p^{(s)})$. By simple algebraic rearrangement, we have

$$k_p^* = \frac{\mathbf{x}^* - \mathbf{x}_t^{(s)}}{\hat{g}_t(k_p^* - k_p^{(s)})} + k_p^{(s)}. \tag{10}$$

The new parameter of the policy is supposed to push the source trajectory $\mathcal{T}^{(s)}$ towards a target trajectory $\mathcal{T}^*$ that passes through the desired state $\mathbf{x}_t^*$ at time t. The result of this experiment on our physical finger platform is available in Figure 2.

4 Conclusions

In this paper, we present a method to learn how the trajectories of a physical real-world dynamical system change with respect to a change in the policy parameters. We tested our method on a custom-built platform called finger robot that allows testing a couple of controllers with various settings to show the applicability of our method for linear, nonlinear, open-loop, and feedback controllers. By estimating the physical derivative function, we showed that our method is able to push a controlled trajectory towards a target intermediate state. We investigate the real-world challenges when doing a fine sensitive task such as estimating physical derivatives on a real robot and proposed solutions to make our algorithm robust to inherent imperfection and noise in physical systems. We focused mainly on low-level issues of physical derivatives and showed the feasibility of estimating them robustly. We expect that physical derivatives will contribute to the research areas such as safety, control with constraint satisfaction and trajectory planning, robust, or safe control.

[1] `https://sites.google.com/view/physicalderivatives/`

REFERENCES

Adrià Puigdomènech Badia, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, Bilal Piot, Steven Kapturowski, Olivier Tieleman, Martín Arjovsky, Alexander Pritzel, Andew Bolt, et al. Never give up: Learning directed exploration strategies. *arXiv preprint arXiv:2002.06038*, 2020.

Akhil Bagaria, Jason K Senthil, and George Konidaris. Skill discovery for exploration and planning using deep skill graphs. In *International Conference on Machine Learning*, pp. 521–531. PMLR, 2021.

Adrien Baranes and Pierre-Yves Oudeyer. Active learning of inverse models with intrinsically motivated goal exploration in robots. *Robotics and Autonomous Systems*, 61(1):49–73, 2013.

Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pp. 1471–1479, 2016.

Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in neural information processing systems*, pp. 6571–6583, 2018.

Adam Gaier and David Ha. Weight agnostic neural networks. *CoRR*, abs/1906.04358, 2019. URL `http://arxiv.org/abs/1906.04358`.

Javier Garcıa and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

Michel Gevers, Alexandre S Bazanella, Xavier Bombois, and Ljubisa Miskovic. Identification and the information matrix: how to get just sufficiently rich? *IEEE Transactions on Automatic Control*, 54(ARTICLE):2828–2840, 2009.

Will Grathwohl, Ricky TQ Chen, Jesse Betterncourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

Karol Gregor, Danilo Jimenez Rezende, and Daan Wierstra. Variational intrinsic control. *arXiv preprint arXiv:1611.07507*, 2016.

Abhishek Gupta, Benjamin Eysenbach, Chelsea Finn, and Sergey Levine. Unsupervised meta-learning for reinforcement learning. *arXiv preprint arXiv:1806.04640*, 2018.

Jiequn Han, Qianxiao Li, et al. A mean-field optimal control formulation of deep learning. *arXiv preprint arXiv:1807.01083*, 2018.

Karol Hausman, Jost Tobias Springenberg, Ziyu Wang, Nicolas Heess, and Martin Riedmiller. Learning an embedding space for transferable robot skills. In *International Conference on Learning Representations*, 2018.

Shuncheng He, Yuhang Jiang, Hongchang Zhang, Jianzhun Shao, and Xiangyang Ji. Wasserstein unsupervised reinforcement learning. *arXiv preprint arXiv:2110.07940*, 2021.

Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Vime: Variational information maximizing exploration. *arXiv preprint arXiv:1605.09674*, 2016.

Frederic Kaplan and Pierre-Yves Oudeyer. Motivational principles for visual know-how development. 2003.

Hassan K Khalil. Nonlinear systems. *Upper Saddle River*, 2002.

Varun Raj Kompella, Marijn Stollenga, Matthew Luciw, and Juergen Schmidhuber. Continual curiosity-driven skill acquisition from high-dimensional video inputs for humanoid robots. *Artificial Intelligence*, 247:313–335, 2017.

Bernard O Koopman. Hamiltonian systems and transformation in hilbert space. *Proceedings of the National Academy of Sciences of the United States of America*, 17(5):315, 1931.

Michael Laskin, Aravind Srinivas, and Pieter Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. In *International Conference on Machine Learning*, pp. 5639–5650. PMLR, 2020.

Michael Laskin, Denis Yarats, Hao Liu, Kimin Lee, Albert Zhan, Kevin Lu, Catherine Cang, Lerrel Pinto, and Pieter Abbeel. Urlb: Unsupervised reinforcement learning benchmark. *arXiv preprint arXiv:2110.15191*, 2021.

Lennart Ljung. System identification. *Wiley Encyclopedia of Electrical and Electronics Engineering*, 2001.

Aleksandr Mikhailovich Lyapunov. The general problem of the stability of motion. *International journal of control*, 55(3):531–534, 1992.

Vitchyr H Pong, Murtaza Dalal, Steven Lin, Ashvin Nair, Shikhar Bahl, and Sergey Levine. Skew-fit: State-covering self-supervised reinforcement learning. *arXiv preprint arXiv:1903.03698*, 2019.

Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer School on Machine Learning*, pp. 63–71. Springer, 2003.

Samuel H Rudy, J Nathan Kutz, and Steven L Brunton. Deep learning of dynamics and signal-noise decomposition with time-stepping constraints. *Journal of Computational Physics*, 396:483–506, 2019.

Shankar Sastry and Marc Bodson. *Adaptive control: stability, convergence and robustness*. Courier Corporation, 2011.

Jürgen Schmidhuber. Curious model-building control systems. In *Proc. international joint conference on neural networks*, pp. 1458–1463, 1991a.

Jürgen Schmidhuber. A possibility for implementing curiosity and boredom in model-building neural controllers. In *Proc. of the international conference on simulation of adaptive behavior: From animals to animats*, pp. 222–227, 1991b.

Jürgen Schmidhuber. Formal theory of creativity, fun, and intrinsic motivation (1990–2010). *IEEE Transactions on Autonomous Mental Development*, 2(3):230–247, 2010.

Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis, Pieter Abbeel, Danijar Hafner, and Deepak Pathak. Planning to explore via self-supervised world models. In *International Conference on Machine Learning*, pp. 8583–8592. PMLR, 2020.

Krishnan Srinivasan, Benjamin Eysenbach, Sehoon Ha, Jie Tan, and Chelsea Finn. Learning to be safe: Deep rl with a safety critic. *arXiv preprint arXiv:2010.14603*, 2020.

Leonid Kuvayev Rich Sutton. Model-based reinforcement learning with an approximate, learned model. In *Proceedings of the ninth Yale workshop on adaptive and learning systems*, pp. 101–105, 1996.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

BCL Touwen, MS Hempel, and LC Westra. The development of crawling between 18 months and four years. *Developmental Medicine & Child Neurology*, 34(5):410–416, 1992.

Juyang Weng, James McClelland, Alex Pentland, Olaf Sporns, Ida Stockman, Mriganka Sur, and Esther Thelen. Autonomous mental development by robots and animals. *Science*, 291(5504): 599–600, 2001.

Anthony M Zador. A critique of pure learning and what artificial neural networks can learn from animal brains. *Nature communications*, 10(1):1–7, 2019.

Kemin Zhou and John Comstock Doyle. *Essentials of robust control*, volume 104. Prentice hall Upper Saddle River, NJ, 1998.

A Supplementary Material

A.1 Physical Platform

In this section, we introduce the physical robot on which we tested our method. The robot is called *finger platform* or simply *finger* throughout this paper. The range of movement for the motors are $[0, \pi]$, $[0, \pi]$, $[0, 2\pi]$ respectively. The axes of the plots throughout the paper are in radian. It consists of three articulated arms with three degrees of freedom in total (see Figure 3d). The motors $\{m_1, m_2, m_3\}$ are depicted in the figure. This naming remains consistent throughout this paper. Each arm is moved by a separate brushless DC motor and has one degree of freedom to swing in its own plane (see Figure 3a). Each arm is equipped with an encoder that measures its angle (see Figure 3b). The brushless motors are controlled by an electronic driver that receives torque values applied to each motor from a computer terminal via a CAN bus and applies the torques to the motors (see Figure 3c). Due to the imperfections of the arms, motors, and drivers, we did not use any model for the system, including the inertial matrix of the robot or the current-torque characteristic function of the motors. The low-cost and safe nature of this robot makes it a suitable platform to test the idea of physical derivatives that requires applying many different controllers in the training phase.

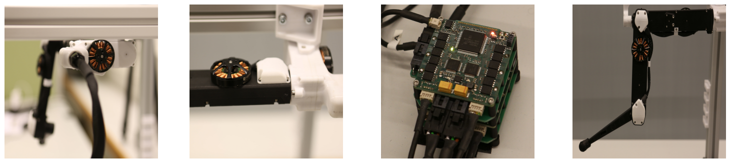

(a) Motor 1 (b) Encoder (c) Driver (d) Finger

Figure 3: Components of the physical finger platform

A.2 Additional Plots illustrating Real World Challenges (section A.5)

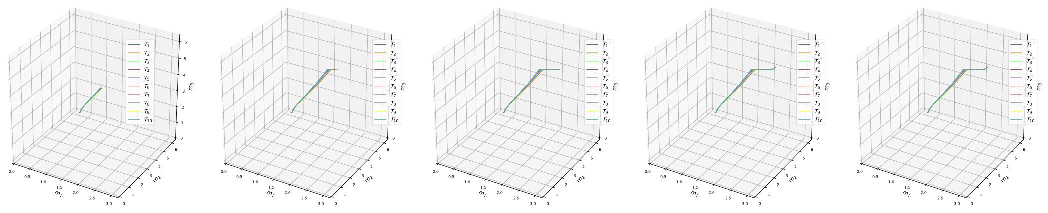

(a) $t = 200$ (b) $t = 400$ (c) $t = 600$ (d) $t = 800$ (e) $t = 1000$

Figure 4: Same controller applied for multiple runs. The trajectories are produced by the linear open-loop controller similar to those used in Appendix A.7.1. See the plot for a different set of nominal parameters of the controller in Figure 7 in the Appendix (Zooming is recommended).

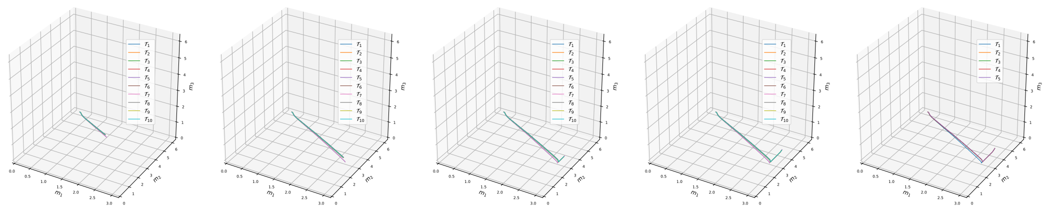
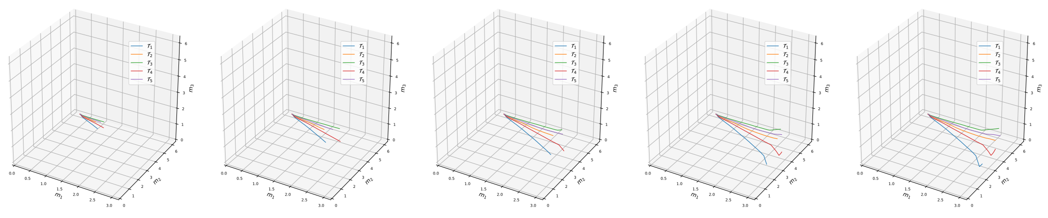
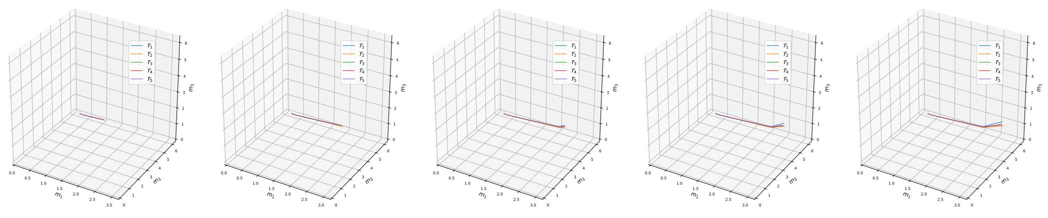

(a) $t = 200$ (b) $t = 400$ (c) $t = 600$ (d) $t = 800$ (e) $t = 1000$

Figure 5: The same as Figure 4 but for a different setting.



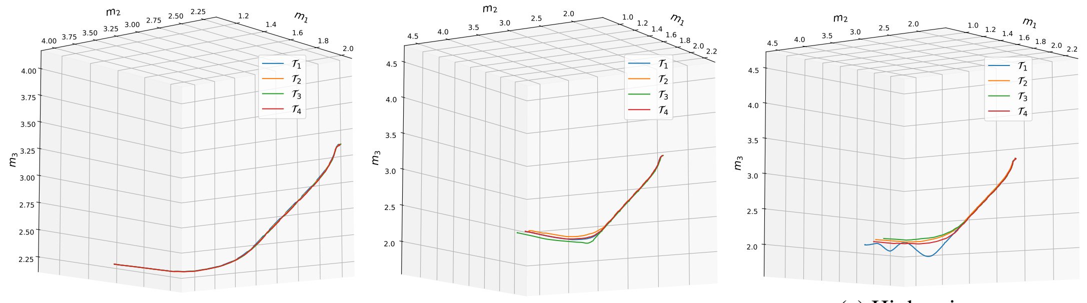
(a) $t = 200$ (b) $t = 400$ (c) $t = 600$ (d) $t = 800$ (e) $t = 1000$

Figure 6: Noisy linear open-loop controller.



(a) $t = 200$ (b) $t = 400$ (c) $t = 600$ (d) $t = 800$ (e) $t = 1000$

Figure 7: The same as Figure 6 but for a different nominal parameters of the policy.



(a) Low noise (b) Medium noise (c) High noise

Figure 8: PD controller with various noise intensities on K_p parameter.

A.3 Applications of physical derivatives

Supposing we know how the states of a trajectory change as a result of a change in the policy parameters, the policy can be easily updated to push the trajectory towards a desired one. For example, assume we are interested in going from the current trajectory $\mathcal{T}(\boldsymbol{\theta})$ to the target trajectory $\mathcal{T}^*$. The distance between these trajectories can get minimized by perturbing the policy parameters in the direction $-\partial\|\mathcal{T}(\boldsymbol{\theta}) - \mathcal{T}^*\|/\partial\boldsymbol{\theta}$. This direction is already available since we have estimated $\partial\mathcal{T}(\boldsymbol{\theta})/\partial\boldsymbol{\theta}$ as a physical derivative. As an exemplary case, we show this application of our method in practice in Section 3. Other applications of physical derivatives are in *robust control* and *safety*. In both cases, the physical derivative allows us to predict the behaviour of the system if the policy changes in a neighbourhood around a nominal policy. Then, it is possible to make sure that some performance or safety criteria will not be violated for the local perturbation in the policy. As a concrete example, for an autonomous driving system, there can be a calibration phase during which physical derivatives of the car are estimated by perturbing the controller parameters around different nominal policies which are likely to occur on real roads. The calibration must be done in a safe condition and before deploying the system. When deployed, the estimated physical derivatives can be used to predict the effect of a change of the policy on the behaviour of the system and neutralize the change if it would move the car towards unsafe regions of its state space. The command that changes the policy can be issued by a high-level controller (e.g., guidance system), and the safety is confirmed by a low-level mechanism through physical derivatives. This work focuses on the concept and the introduction of physical derivatives and direct applications would go significantly beyond the scope of this work.

Robust control. In control theory, robust control relates to the design of a controller whose performance is guaranteed for a range of systems and controllers belonging to a certain neighborhood around the nominal system (Zhou & Doyle, 1998). It is desired to have a controller that keeps the performance of the system at a certain good level even if the parameters of the controller are not fixed to the theoretical values. Assume the performance of the system is associated with some function of a trajectory $\mathcal{E}(\mathcal{T})$. Changing the parameters of the controller $\boldsymbol{\theta}$ results in a change in the trajectories. This allows us to compute $\partial\mathcal{T}/\partial\boldsymbol{\theta}$ that consequently gives us $\partial\mathcal{E}(\mathcal{T})/\partial\boldsymbol{\theta}$ by the chain rule. Roughly speaking, between two sets of parameters $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$, the set of parameters that gives the least $\partial\mathcal{E}/\partial\boldsymbol{\theta}$ is preferred. This means that by perturbing the parameters of the controller and assessing the performance of the system, an estimate of the curvature of the landscape of $\mathcal{E}(\mathcal{T}(\boldsymbol{\theta}))$ is obtained. We prefer flatter regions of this space where a small change in $\boldsymbol{\theta}$ does not cause a drastic change in the performance metric $\mathcal{E}$.

Safety. Safety refers to the situations in which the agent may hurt itself or the environment and causes irreversible damages if it freely takes arbitrary actions (Garcıa & Fernández, 2015). For a safety-critical system whose full physical models are hard to obtain, the physical gradients can assist in avoiding restricting the parameters of the robot to prevent unsafe behavior. The physical derivatives are learned in the Lab environment before the robot is deployed into the wild. For example, a rover whose mission is to safely explore an unknown environment often enjoys a learning loop that allows it to adapt to the new environment. Even though the learning in the new environment requires sufficient exploration, the physical derivatives can be used to give a rough simulation of the robot's next few states under a given update to its parameters. The potential harmful updates might be detected by such simulation and be avoided.

Adversarial system identification. System identification concerns learning about the governing equations of the system from observed trajectories. Most systems require some input excitation to show their behavior. Normally the input is designed rich enough to elicit the important behaviors from the system (Gevers et al., 2009). Assuming the system would not be harmed by the input signal, this traditional approach has the downside: we never know the input signal is rich enough to excite every mode of the system. Besides, there exist scenarios when the identification must be carried out with minimal control effort. Hence, the design of the input signal must not be agnostic to the dynamics of the system. However, estimating the dynamics is the initial goal of system identification. This seeming chain problem can be solved by repetitively updating the control input and the estimated model. This results in an adversarial game between the controller and the current best estimate of the system which can be described as the following min-max problem

$$\arg\min_{\boldsymbol{\phi}} \arg\max_{\boldsymbol{\theta},\mathbf{x}_0} \mathcal{L} = \frac{1}{T}\int_{t_0}^{t_0+T} \| \int_{t_0}^{t} \hat{f}(\mathbf{x}_m(t), \mathbf{u}(\mathbf{x}, t; \boldsymbol{\theta}); \boldsymbol{\phi})dt - \mathbf{x}(t; \boldsymbol{\theta})\|_2^2 \, \mathrm{d}t. \tag{11}$$

In this formula, $\mathbf{x}_m$ denotes the states of the model parameterized by $\boldsymbol{\phi}$ and the parameters of the estimated model while $\boldsymbol{\theta}$ shows the parameters of the controller. Notice that $\mathbf{x}$ is the state of the physical system, which is a function of the controller parameters too. The procedure proceeds as follows: A controller with parameters $\boldsymbol{\theta}$ is applied to both models and the physical system. The produced trajectories by the model and the physical system are compared. The model parameters are updated in the direction that minimizes this distance to give a better estimate of the system. The controller parameters, on the other hand, are updated to maximize this distance. This maximization ensures that the controller drives the system towards the regions of the state space for which the current model is not yet accurate.

A.4 Detailed literature review

There has been a recent surge of interest in unsupervised methods in reinforcement learning when a task-specific reward function is not the only driving force to train the agent (Baranes & Oudeyer, 2013; Bellemare et al., 2016; Gregor et al., 2016; Hausman et al., 2018; Houthooft et al., 2016; Badia et al., 2020; Sekar et al., 2020). A truly intelligent agent must behave intelligently in a range of tasks, not only in a single task associated with its reward function. This requires the agent to develop some sort of *general competence* that allows it to come up with solutions to new problems by combining some low-level primitive skills. This general competence is a key factor in animals to quickly and efficiently adapt to a new environment (Weng et al., 2001). By calling the traditional RL, *extrinsicially motivated RL*, the new framework is called *intrinsically motivated RL*. There have been many ideas in this line with various definitions for the terms *motivation* and *intrinsic*. Some researchers assume a developmental period in which the agent acquires some reusable modular skills that can be easily combined to tackle more sophisticated tasks (Kaplan & Oudeyer, 2003; Weng et al., 2001). Curiosity and confidence are other unsupervised factors that can be used to drive the agent towards unexplored spaces to achieve new skills (Schmidhuber, 1991b; Kompella et al., 2017; Bagaria et al., 2021). Interestingly, there are observations in neuroscience that dopamine, a known substance that controls one's motivation for extrinsic rewards, is also associated with intrinsic properties of the agent, such as novelty and curiosity. A novel sensory stimulus activates the dopamine cells the same way they are activated by extrinsic reward. Children build a collection of skills accumulatively while they engage in activities without a specific goal, e.g., hitting a ball repeatedly without a long-term target such as scoring a goal. The achieved skills contribute to their stability while handling objects (Touwen et al., 1992).

Another line of work concerns the fundamental constraints of the agent/environment and ensures those constraints are met while learning. For example, in many practical systems, learning episodes must halt if the system is likely to undergo an irreversible change. For example, the training episodes of a fragile robot must ensure the robot does not fall or will not be broken in any circumstance while acting under a certain policy. The general name *safe RL* embodies ideas to tackle such issues in current interactive learning algorithms (Garcıa & Fernández, 2015; Srinivasan et al., 2020). One major aspect of safety is *stability* that loosely means that states of the system converge to some invariant sets or remain within a certain bound (Lyapunov, 1992). Control theory enjoys a physical model of the system to guarantee stability (Khalil, 2002). When the physical model is not known in advance, the model is either learned along with the policy (model-based RL) or will be implicitly distilled in the value function (model-free RL) (Sutton & Barto, 2018). Stability can be categorized as an intrinsic motivation for the agent. No matter what task the agent aims to solve, it must remain stable all the time. Learning the transition model, which is the major concern of model-based RL, can also be seen as intrinsic motivation. The agent learns to predict the future step given the current state. The advantage of learning a model—even inaccurately—is twofold: the agent would know where to go and where not to go. It knows which regions of the state space are unsafe to explore and must be avoided. It also knows which regions are unexplored and might be informative to improve the model. This brings us to another view to intrinsic reward that encourages *diversity*.

Our work is also relevant to sensitivity analysis and its use in training the parameters of dynamical models. After Chen et al.'s NeuralODE on training neural networks by sensitivity analysis of the network parameters, the method was successfully applied to various tasks such as learning dynamics (Rudy et al., 2019), optimal control (Han et al., 2018), and generative models (Grathwohl et al., 2018). Our method can be seen as a mode-free sensitivity analysis in real-world systems. In NeuralODE, the gradient with respect to the parameters requires solving ODEs for both states and adjoint states that require a transition model. Since we work directly on the physical system, we don't need to calculate the integrals forward in time. The system itself acts as a physical ODE solver.

The importance of learning from unlabelled experiences is a known fact in animals. Many animals function efficiently soon after birth before being exposed to a massive labeled experience. Part of it might be due to unsupervised learning, but the major part of the story can be a genetic heritage after years of evolution that Zador called *genomic bottleneck*. The same idea turned out to be valid in statistical learning, where an automatically discovered neural network architecture performs surprisingly well with a shared random weight (Gaier & Ha, 2019). The embedded inductive bias in the neural network architectures could be analogous to the wiring of the brain of animal babies, which transfers from generation to generation by genes.

A.5 Tackling inherent noise

Inherent noise refers to a change in the system trajectory that is not caused by the intentional intervention on the controller. In our finger platform, we observed two different major sources of noise that are likely to occur in other physical systems too. We call them *temporal* and *spatial* noise for the reasons that come in the following.

Temporal noise. The temporal noise represented by $\mathbf{n}$ affects trajectories by shifting them in time

$$\mathbf{x}_t \leftarrow \mathbf{x}_{t+\mathbf{n}} \text{ for } t = 0, 1, \ldots, T. \tag{12}$$

Notice that the absence of subscript t in $\mathbf{n}$ shows that this noise is not time-dependent, i.e., the time shift does not change along the trajectory as time proceeds.

Spatial noise. The trajectories affected by spatial noise cannot be aligned with each other by shifting forward or backward in time. We can model this noise as a state-dependent influence on the state of the system at every time step.

$$\mathbf{x}_t \leftarrow \mathbf{x}_t + \mathbf{n}_{\mathbf{x}_t} \tag{13}$$

The following definition makes the distinction more concrete.

Definition 1. *Consider two trajectories $\mathcal{T}^{(1)}(t)$ and $\mathcal{T}^{(2)}(t)$ as two temporal signals. Assume $S_{t_\circ}$ is the shift-in-time operator defined as*

$$S_{t_\circ}\mathcal{T}(t) = \mathcal{T}(t + t_\circ) \tag{14}$$

for an arbitrary function of time $\mathcal{T}(t)$. We say $\mathcal{T}^{(2)}(t)$ is temporally noisy version of $\mathcal{T}^{(1)}(t)$ if

$$\exists t_\circ \in \mathbb{R} \; s.t. \; \|\mathcal{T}^{(2)} - S_{t_\circ}\mathcal{T}^{(1)}\|_1 \leq \epsilon \tag{15}$$

where ϵ is a hyper-parameter threshold that reflects our prior confidence about the accuracy of the motors, joints, physical and electrical elements (in the general construction process) of the robot. On the other hand, $\mathcal{T}^{(2)}$ is called a spatially noisy version of $\mathcal{T}^{(1)}$ if

$$\nexists t_\circ \in \mathbb{R} \; s.t. \; \|\mathcal{T}^{(2)} - S_{t_\circ}\mathcal{T}^{(1)}\|_1 \leq \epsilon \tag{16}$$

The temporal noise is coped with correlation-based delay estimation where the time-shift is estimated by computing the correlation between shifted trajectories and the spatial noise is dealt with voxelization which refers to discretizing the space into voxels (regular 3D boxes) where all states within a voxel are projected on the center of that voxel. We found the decomposition of noise sources into these two classes critical for dealing with them. The detailed description of each method is postponed to Appendix A.5.1 and Appendix A.5.2.

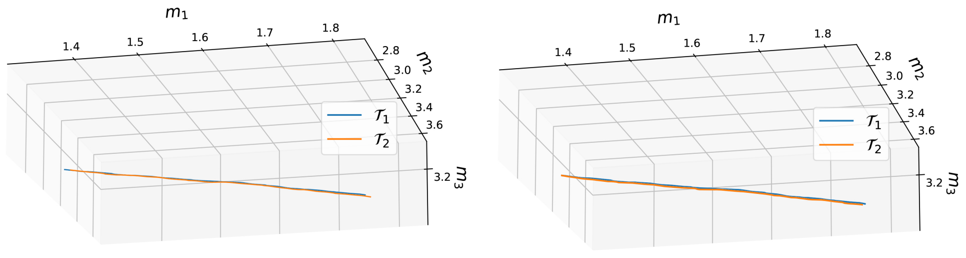
A.5.1 Solution to temporal noise

Fortunately, this type of noise is not state-dependent by definition. If we find out how much a trajectory is shifted in time with respect to another trajectory, we can simply shift the trajectory for those many time steps and compensate for the delay. Hence, the problem becomes detecting the lagged trajectories with respect to a reference trajectory and also estimating the amount of the required time shift to compensate for the delay. We can either use physical landmarks in the trajectories to align them or use the correlation between them as a measure of alignment. The latter gave better results. Hence, we postpone the description of the former.

Correlation-based delay estimation. In this method, we use the correlation between zero-meaned trajectories $\mathcal{T}^{(i)}$ and $\mathcal{T}^{(j)}$ to check if one is the lagged version of the other one. The delay τ is found by

$$\tau^* = \arg\max_{\tau} \sum_{t=0}^{T-\tau} \langle S_\tau \mathbf{x}_t^{(i)}, \mathbf{x}_t^{(j)} \rangle \tag{17}$$

where S_τ is a shift-operator by $\tau \in \mathbb{Z}$ time steps. In practice, we take one trajectory of $\{\mathcal{T}^{(1)}, \mathcal{T}^{(2)}, \ldots, \mathcal{T}^{(M)}\}$, e.g. $\mathcal{T}^{(r)}$ as the reference and synchronize other trajectories with respect to it using Equation (17). The trajectories must be initially normalized to avoid trivial solutions where every trajectory is pushed towards the larger parts of the reference trajectory. For illustrative purposes, the plots of Figure 9 show a sample of the lagged trajectory from the finger platform and its correction by the above method.



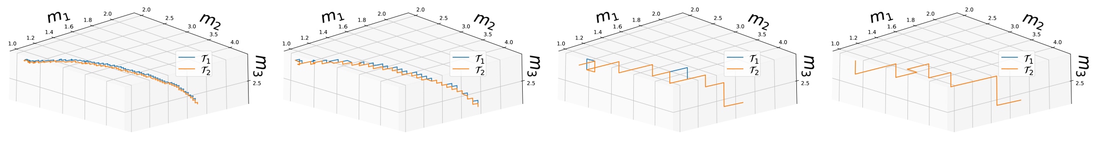
(a) Trajectories affected by temporal noise (b) Temporally aligned trajectories

Figure 9: The effect of temporal noise in delaying one trajectory versus the other one and its correction. The trajectories are produced by the linear open-loop controller similar to those used in Appendix A.7.1.

Detecting zero crossing. In this method, we take advantage of special landmarks in the trajectories. The landmarks are typically caused by the physical constraints of the system. For example, when a robot's leg touches the ground, the velocity of the leg becomes zero. Likewise, when a joint reaches its physical limit, the velocity of the connected arm to the joint becomes zero or changes signs. In both cases, a zero-crossing occurs that can be used as a landmark to synchronize lagged trajectories with a reference trajectory. Even though this method will eliminate the temporal noise, it requires the presence of such landmarks along the trajectories. Notice that from a mathematical point of view, there is nothing special about *zero*. We can pick any value of states along a reference trajectory and synchronize all other trajectories with respect to it. However, in practice, physical landmarks are easier to detect and have less ambiguity that consequently giving a more accurate synchronization.

A.5.2 Solution to spatial noise

The spatial noise can be a stochastic function of the actuator, environmental change, and electronic drivers. In a perfect model of the transition dynamics $\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t)$, applying the same control sequence $\{\mathbf{u}_0, \mathbf{u}_1, \dots, \mathbf{u}_{T-1}\}$ always results in the same sequence of states $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T\}$ when it starts from the same initial state $\mathbf{x}_0$. This assumption is often violated in physical systems as different runs of the same policy may result in different trajectories, as can be seen in Figure 4 in the Appendix. The noise in the dynamics can be any function of states, input, and time. Therefore, it is difficult to model this noise since it requires a prohibitively large number of random experiments. The good news is that if the physical system is built properly, the effect of this noise is expectedly low. Based on our observations from the finger platform, we can assume the following.



(a) Small voxels ($\gamma = 0.01$) (b) Medium voxels ($\gamma = 0.04$) (c) Large voxels ($\gamma = 0.16$) (d) Very large voxels ($\gamma = 0.2$)

Figure 10: The effect of voxels on supressing spatial noise of the physical system. The trajectories are produced by linear open-loop controllers as those in Appendix A.7.1 for the purpose of illustrating the effect of voxelization.

Assumption 2. *Limit on the physical noise: Let's the control sequence* $\mathbf{U} = \{\mathbf{u}_0, \mathbf{u}_1, \dots, \mathbf{u}_{T-1}\}$ *be applied to the system* M *times resulting in multiple sequence of states* $\mathcal{T}^{(1)}, \mathcal{T}^{(2)}, \dots, \mathcal{T}^{(M)}$. *There exists a relatively small* ζ *such that*

$$\|\mathcal{T}^{(i)} - \mathcal{T}^{(j)}\|_\infty \leq \zeta \quad \text{for every} \quad i, j \in \{1, 2, \dots, m\}. \tag{18}$$

The word *relatively* here means that the change of the trajectory due to the inherent physical noise of the system must be small compared to the change of the trajectories when the parameters of the policy are perturbed.

To reduce the sensitivity of the estimated gradient to this unwanted spatial noise, we divide the state space of the physical system into regularly located adjacent cells called *voxels*. Each voxel $vox(\mathbf{c})$ is represented by its center $\mathbf{c}$ and is defined as

$$vox(\mathbf{c}) = \{\mathbf{x} \in \mathcal{X} \mid \|\mathbf{x} - \mathbf{c}\|_\infty \leq \gamma\} \tag{19}$$

where γ is the parameter of the voxelization. The concept of the voxel is roughly used as a *super-state*. Every state that ends up within $vox(\mathbf{c})$ gives rise to the same superstate. After recording the trajectories from the robot, every state is mapped to the center of the voxel it belongs to as

$$\mathbf{c} \leftarrow \mathbf{x} \text{ for } \mathbf{x} \in vox(\mathbf{c}) \tag{20}$$

For simplicity, we denote the center $\mathbf{c}$ of the voxel that $\mathbf{x}$ belongs to with $\mathbf{c}_\gamma(\mathbf{x})$. After voxelization, we work with $\mathbf{c}_\gamma(\mathbf{x})$ instead of $\mathbf{x}$. For example, all the gradients of equation 6 are computed as $\nabla_{\boldsymbol{\theta}}\mathbf{c}$ rather than $\nabla_{\boldsymbol{\theta}}\mathbf{x}$. To illustrate the positive effect of voxelization of the state space, it can be seen in Figure 10 that increasing the voxel size improves the overlapping between two trajectories that deviate from each other due to the inherent spatial noise of the system not because of perturbing the parameters of the policy, but because of the inherent imperfection of the mechanical and electrical components of the system. This benefit comes with a cost which is the error introduced by voxelization. Fortunately, this error is bounded due to the following lemma.

Lemma 3. *The additional error caused by voxelization* $\mathbf{c}_\gamma(.)$ *is bounded proportional to the size of each voxel* γ.

Proof. Assume that we have two noisy (spatial noise) trajectories $\mathbf{y}_t^{(1)} = \mathbf{x}_t^{(1)} + \epsilon_t^{(1)}, \mathbf{y}_t^{(2)} = \mathbf{x}_t^{(2)} + \epsilon_t^{(2)}$, for $t \in \{1, 2, \dots, T\}$ where $\mathbf{x}_t^{(1)}, \mathbf{x}_t^{(2)}$ are primary noiseless trajectories having significant difference that we want to calculate derivatives for and spatial errors $\epsilon_t^{(1)}, \epsilon_t^{(2)}$ comes from a distribution. The error induced in the standard case is $\mathbb{E}_{\epsilon_t^{(1)},\epsilon_t^{(2)}}[||(\mathbf{y}_t^{(2)} - \mathbf{y}_t^{(1)}) - (\mathbf{x}_t^{(2)} - \mathbf{x}_t^{(1)})||_\infty] = \mathbb{E}_{\epsilon_t^{(1)},\epsilon_t^{(2)}}[||(\mathbf{y}_t^{(2)} - \mathbf{x}_t^{(2)}) - (\mathbf{y}_t^{(1)} - \mathbf{x}_t^{(1)})||_\infty] = \mathbb{E}_{\epsilon_t^{(1)},\epsilon_t^{(2)}}[||\epsilon_t^{(2)} - \epsilon_t^{(1)}||_\infty]$, while the error in the voxelized world is $\mathbb{E}_{\epsilon_t^{(1)},\epsilon_t^{(2)}}[||(\mathbf{c}_\gamma(\mathbf{y}_t^{(2)}) - \mathbf{c}_\gamma(\mathbf{y}_t^{(1)})) - (\mathbf{x}_t^{(2)} - \mathbf{x}_t^{(1)})||_\infty] = \mathbb{E}_{\epsilon_t^{(1)},\epsilon_t^{(2)}}[||(\mathbf{c}_\gamma(\mathbf{y}_t^{(2)}) - \mathbf{y}_t^{(2)}) - (\mathbf{c}_\gamma(\mathbf{y}_t^{(1)}) - \mathbf{y}_t^{(1)}) + (\mathbf{y}_t^{(2)} - \mathbf{y}_t^{(1)}) - (\mathbf{x}_t^{(2)} - \mathbf{x}_t^{(1)})||_\infty]$. Using triangle inequality and knowing that $||\mathbf{c}_\gamma(\mathbf{y}_t^{(i)}) - \mathbf{y}_t^{(i)}||_\infty \leq \gamma$, we have

$$\mathbb{E}_{\epsilon_t^{(1)},\epsilon_t^{(2)}}[||(\mathbf{c}_\gamma(\mathbf{y}_t^{(2)}) - \mathbf{c}_\gamma(\mathbf{y}_t^{(1)})) - (\mathbf{x}_t^{(2)} - \mathbf{x}_t^{(1)})||_\infty] \leq 2\gamma + \mathbb{E}_{\epsilon_t^{(1)},\epsilon_t^{(2)}}[||\epsilon_t^{(2)} - \epsilon_t^{(1)}||_\infty].$$

Hence, the lemma is proved. ☐

The voxels become boxes in 3D as in Figure 11. The gradient is estimated as the distance between two points in 3D coordinates. Hence the source of voxelization error is approximating the distance between two points in 3D with the distance between the centers of the corresponding boxes to which those points belong. This error is written next to the boxes in Figure 11. The maximum additional error due to voxelization is proportional to the voxel sizes γ and its proportion is upper-bounded by 2.

A.6 PERTURBATION METHODS

Gaussian Perturbation— Likely values of $\boldsymbol{\theta}$ create nominal policies encoded by $\{\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, \dots, \boldsymbol{\theta}^{(m)}\}$. We put Gaussian distributions centered at each of the nominal values resulting in a mixture of Gaussians. To reduce the hyper-parameters, we assume the variances of the Gaussians are themselves sampled from an exponential distribution making sure they all take positive values (See Figure 12 left). Here, we manually choose a reasonable value for the rate parameter of the exponential distribution. Making inference on the hyper-parameters of the sampling distributions can be a topic for future research, especially in active learning for a more

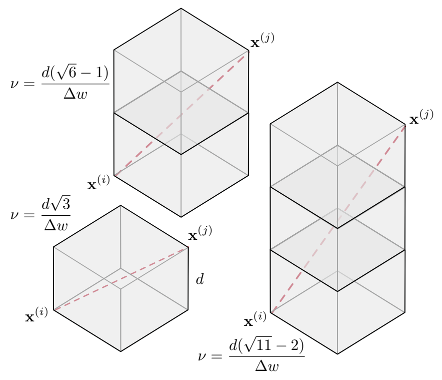

$\nu = \frac{d(\sqrt{6}-1)}{\Delta w}$
$\mathbf{x}^{(j)}$
$\mathbf{x}^{(i)}$
$\nu = \frac{d\sqrt{3}}{\Delta w}$
d
$\nu = \frac{d(\sqrt{11}-2)}{\Delta w}$

Figure 11: The maximum potential error in the estimated gradients when the space is voxelized. As can be seen, the error vanishes when the corresponding voxels to $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$ are far from each other.

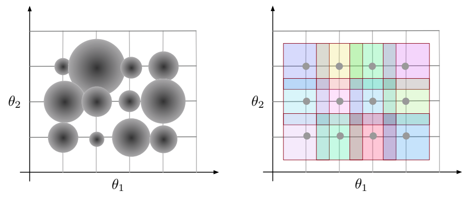


Figure 12: Gaussian (left) and uniform (right) perturbation examples.

clever less costly sampling strategy.

Uniform Perturbation— In this setting, the state space of the changeable parameters of the policy is discretized and a uniform distribution is assumed around each value of this grid with some overlapping with the neighboring cells (See Figure 12 right).

A.7 EXPERIMENTAL RESULTS

In this section, the experiments are designed to show each challenge separately and the efficacy of our proposed solution to it. These experiments subsume *linear open-loop controller*, *Nonlinear open-loop controller*, and *Feedback controller*.

A.7.1 LINEAR OPEN-LOOP CONTROLLER

As a simple yet general policy, in this section, we consider an open-loop controller , which is a linear function of time. The policy $\mathbf{u}_t = [u_{1t}, u_{2t}, u_{3t}]$ constitutes the applied torques to the three motors $\{m_1, m_2, m_3\}$ of the system and is assigned as

$$u_{it} = w_i t + b_i \quad \text{for} \quad i = 1, 2, 3 \tag{21}$$

Notice that the torque consists of two terms. The first term $w_i t$ grows with time and the second term remains constant. The controller has 6 parameters in total denoted by $\boldsymbol{\theta}$. The task is to predict $\nabla_{\boldsymbol{\theta}} \mathbf{x}_t$ for every t along the trajectory. In the training phase, the training data is obtained via perturbation as described in Section 2.

Figure 15 shows examples of nominal trajectories + trajectories produced by the perturbed controller and the computed derivatives. The arrows are plotted as they originate from the perturbed trajectories only for easier distinction. Each arrow corresponds to the change of the states at a certain time step on the source trajectory as a result of perturbing the policy. Each figure corresponds to a pair of nominal values of $\{w, b\}$ for the linear open-loop controller.

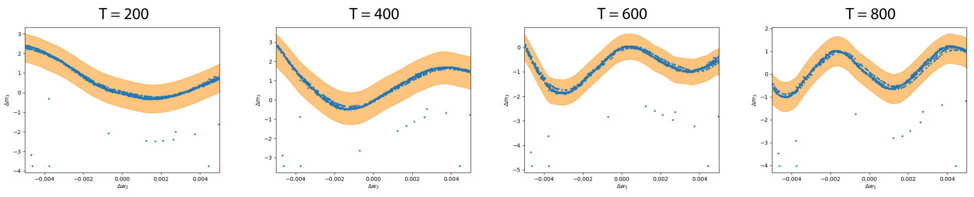


Figure 13: The time evolution of the GP approximated $\hat{g}_t$ for a nonlinear sinusoidal open-loop controller at some exemplary time instances.

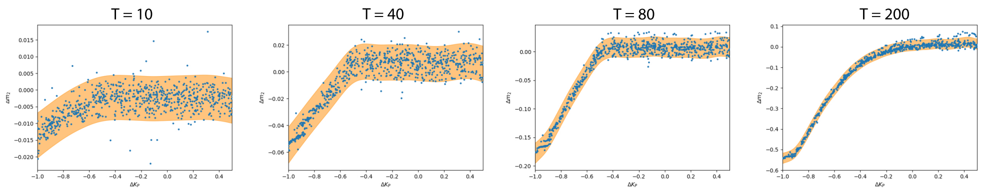


Figure 14: The time evolution of the GP approximated $\hat{g}_t$ for a PD feedback controller at some exemplary time instances

Table 1: The aggregate performance of our method to predict physical derivatives in unseen directions of perturbations to the parameters. $\langle \cdot \rangle$ shows the time average. The first column is the normalized time-averaged MSE. The second column is the time-averaged GP score (closer to 1 is better. See Appendix A.8.5 for definition). The third column is the time-averaged misalignment between derivatives. Every experiment is repeated for 10 voxel sizes and the values are chosen for the best voxel size. N: Gaussian sampling, U: uniform sampling.

Task	$\langle$MSE$\rangle$	$\langle$Score$\rangle$	$\langle\cos\alpha\rangle$
PD controller (N)	0.0087	0.8991	0.9492
PD controller (U)	0.0018	0.9841	0.9723
Sine 2 joints (U)	0.0368	0.7992	0.9513
Sine 2 joints (N)	0.0796	0.6696	0.9553

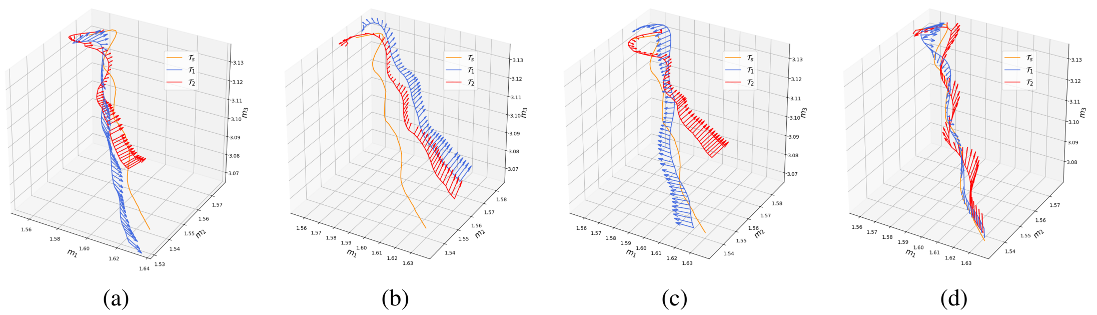


Figure 15: Physical gradients computed for various time steps along a source trajectory using two perturbed trajectories of linear open-loop. The orange trajectory is the source trajectory $\mathcal{T}_s$ and others are the perturbed ones $\mathcal{T}_1, \mathcal{T}_2$. The quivers on the perturbed trajectories represent calculated finite-difference physical derivatives which point for a state on the perturbed trajectory towards the state that occurs at the same time on the source trajectory. See further examples in Figure 29.

A.7.2 NONLINEAR OPEN-LOOP CONTROLLER

Physical derivatives can naturally be computed for either linear or nonlinear controllers, which makes it different from taking the gradient of models through time. In model-based methods, if the model's transition dynamics is not differentiable, taking the derivative is theoretically challenging. However, our method takes advantage of the real physics of the system to compute the gradients regardless of whether the approximating model is differentiable or not. To elaborate more on this, we test our method for a simple but nonlinear policy, i.e., $u_t = \mathcal{A}\sin(\omega t)$. The sinusoidal torque is applied to either one or two motors of the system to investigate the performance of our method. We tested Gaussian and uniform perturbation for $\boldsymbol{\theta} = \{\mathcal{A}, \omega\}$ as parameters of this controller. The GP interpolation for the partial derivatives at some time instances along the trajectory can be seen in Figure 13 and more extensively in Figures 16 to 18. One might be interested in the direction of the predicted derivative instead of its exact size. To this end, we take several test perturbations for every time step and use $\cos(\alpha)$ as a measure of alignment between the predicted and ground-truth derivative vectors. The time evolution of the histogram of this measure along the trajectory shows a better alignment as time proceeds. This effect can be seen in Figures 27 and 28. This confirms our observation of initial transient noise in the system that dies out gradually by the progression of time. The overall performance of our method in predicting physical derivatives in unseen directions for two different perturbation methods is shown in Table 1.

A.7.3 Feedback controller

Often in practice, the policy incorporates some function of the states of the system. Some well-known examples which have been extensively used in control applications are P, PD, PI and PID controllers. Here, we consider two members of this family, i.e., P and PD controllers. The policy becomes $\mathbf{u} = K_p\mathbf{e}$ for P controllers and $\mathbf{u} = K_p\mathbf{e} + K_d\dot{\mathbf{e}}$ for PD controllers. The error $\mathbf{e}$ shows the difference between the current state $\mathbf{x}$ and the desired state $\mathbf{x}^*$. The parameters of the controller $\{K_p, K_d\}$ are scalar values that are multiplied by the error vector elementwise. This implies that the controller parameters are the same for three motors, leaving the whole platform's controller with two parameters that weigh the value and the rate of the error. We applied the uniform and Gaussian perturbation for the set of parameters $\boldsymbol{\theta} = \{K_p, K_d\}$ with different scenarios. Figure 8 is an illustration of the resultant trajectories for different levels of noise in the policy. The GP interpolation for the physical derivatives at some time instances along the trajectory can be seen in Figure 14 and more extensively in Figures 19 to 24. The time evolution of the histogram of misalignment between predicted and ground-truth directional derivatives (see Figures 25 and 28) once again confirms the existence of the initial transient noise, as was also observed in the Appendix A.7.2. Similar to the sinusoidal experiment, the overall performance of our method is presented in Table 1.

A.8 Experimental details

Starting position in all the experiments is $(\frac{\pi}{2}, \frac{\pi}{2}, \pi)$. Task's overall details are as following:

Task	number of trajectories	timesteps
Linear (N)	640	1500
PD controller(N)	640	1500
PD controller(U)	1000	1500
Sine 1 joint(N)	640	5000
Sine 1 joint(U)	1000	5000
Sine 2 joints(U)	640	5000
Sine 2 joints(N)	1000	5000

In normal sampling cases, we ran 10 simulations for each set of λ parameters which indicates noise level.

A.8.1 Linear

$$u_{it} = w_i t + b_i \quad \text{for} \quad i = 1, 2, 3 \tag{22}$$

Gaussian Sampling

$$w_i = W_i + \epsilon_{w,i} \quad \text{for} \quad i = 1, 2, 3$$
$$\epsilon_{w,i} \sim N(0, e_w \times \|W_i\|_2)$$
$$e_w \sim exp(\lambda_w) \quad \text{for} \quad \lambda_w = 1, 5, 10, 50, 100, 500, 1000, 5000$$

$$b_i = B_i + \epsilon_{b,i} \quad \text{for} \quad i = 1, 2, 3$$
$$\epsilon_{b,i} \sim N(0, e_b \times \|B_i\|_2)$$
$$e_b \sim exp(\lambda_b) \quad \text{for} \quad \lambda_b = 1, 5, 10, 50, 100, 500, 1000, 5000$$

$$W = [0.00001, 0.0001, -0.00001], B = [-0.28, -0.15, -0.08]$$

A.8.2 PD Controller

Final destination is $(\frac{\pi}{10}, 3\frac{\pi}{4}, 7\frac{\pi}{12})$

Gaussian Sampling

$$kp = KP + \epsilon$$
$$\epsilon_{kp} \sim N(0, e_{kp} \times \|KP\|_2)$$
$$e_{kp} \sim exp(\lambda_{kp}) \quad \text{for} \quad \lambda_{kp} = 1, 5, 10, 50, 100, 500, 1000, 5000$$

$$kd = KD + \epsilon$$
$$\epsilon_{kd} \sim N(0, e_{kd} \times \|KD\|_2)$$
$$e_{kd} \sim exp(\lambda_{kd}) \quad \text{for} \quad \lambda_{kd} = 1, 5, 10, 50, 100, 500, 1000, 5000$$

Uniform Sampling

$$kp \sim U(-0.5, 1.5), KP = 1$$

$$kd = KD = 0.01$$

A.8.3 Sine 1 joint

Gaussian Sampling

$$w = W + \epsilon$$
$$\epsilon_w \sim N(0, e_w \times \|W\|_2)$$
$$e_w \sim exp(\lambda_w) \quad \text{for} \quad \lambda_w = 1, 5, 10, 50, 100, 500, 1000, 5000$$

$$a = A + \epsilon$$
$$\epsilon_a \sim N(0, e_a \times \|A\|_2)$$
$$e_a \sim exp(\lambda_a) \quad \text{for} \quad \lambda_a = 1, 5, 10, 50, 100, 500, 1000, 5000$$

$$W = 0.01, B = 0.5$$

Uniform Sampling

$$w \sim U(0.005, 0.015), a = A = 0.5$$

A.8.4 Sine 2 joints

Gaussian Sampling

$$w_i = W_i + \epsilon \quad \text{for} \quad i = 1, 2$$
$$\epsilon_{w,i} \sim N(0, e_w \times \|W\|_2)$$
$$e_w \sim exp(\lambda_w) \quad \text{for} \quad \lambda_w = 1, 5, 10, 50, 100, 500, 1000, 5000$$

$$a_i = A_i + \epsilon \quad \text{for} \quad i = 1, 2$$
$$\epsilon_{a,i} \sim N(0, e_a \times \|A\|_2)$$
$$e_a \sim exp(\lambda_a) \quad \text{for} \quad \lambda_a = 1, 5, 10, 50, 100, 500, 1000, 5000$$

$$W = [0.01, 0.01], A = [-0.4, 0.5]$$

Uniform Sampling

$$w_i \sim U(0.005, 0.015) \quad \text{for} \quad i = 1, 2, a = A = 0.5$$

A.8.5 GP Score:

Definition of the GP score: The score is defined as $(1 - u/v)$, where u is the residual sum of squares $\Sigma(y_{\text{true}} - y_{\text{pred}})^2$ and v is the total sum of squares $\Sigma(y_{\text{true}} - \text{mean}(y_{\text{true}}))^2$. The best possible score is 1.0.

A.8.6 Zero-shot planning task:

For the task of Section 3: Number of training trajectories: 100 each with 1500 time steps

Kd = 0.01

Kp = Uniformly sampled from $[0.2, 0.6]$

Initial point: $X_\circ = [\pi/2, \pi/2, \pi])$

desired position = $[\pi/10, 3 * \pi/4, 7 * \pi/12]$

A.9 More results

In this section, the results of the extra experiments that were eliminated from the main text due to the space limit are presented.

The following figures show GP models trained by a set of directional derivatives collected during the perturbation phase. The results are provided for the experiments of Appendices A.7.2 and A.7.3.

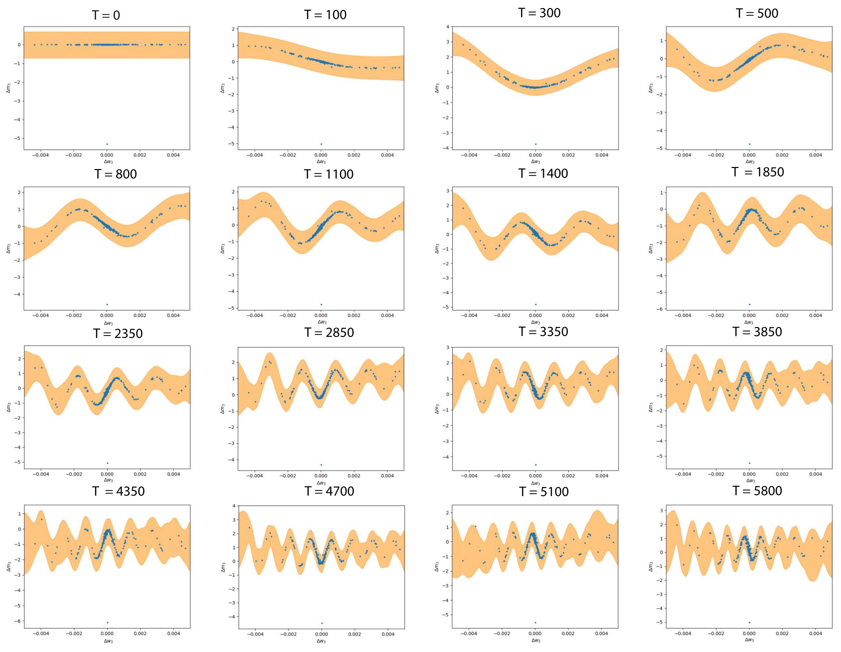


Figure 16: The time evolution of the learned GP models from directional derivatives for $\partial x_3/\partial k_p$ by Gaussian sampling (Sine 1 joint).

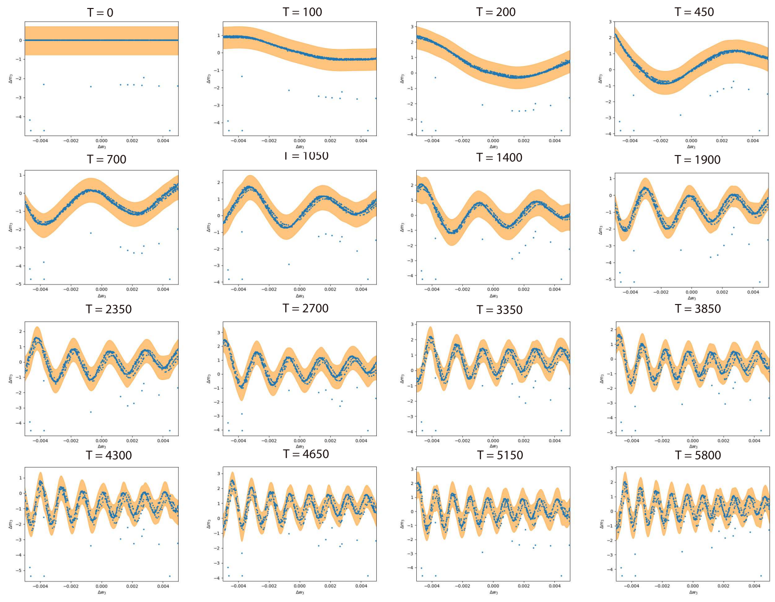


Figure 17: The time evolution of the learned GP models from directional derivatives for $\partial x_3/\partial k_p$ by uniform sampling (Sine 1 joint).

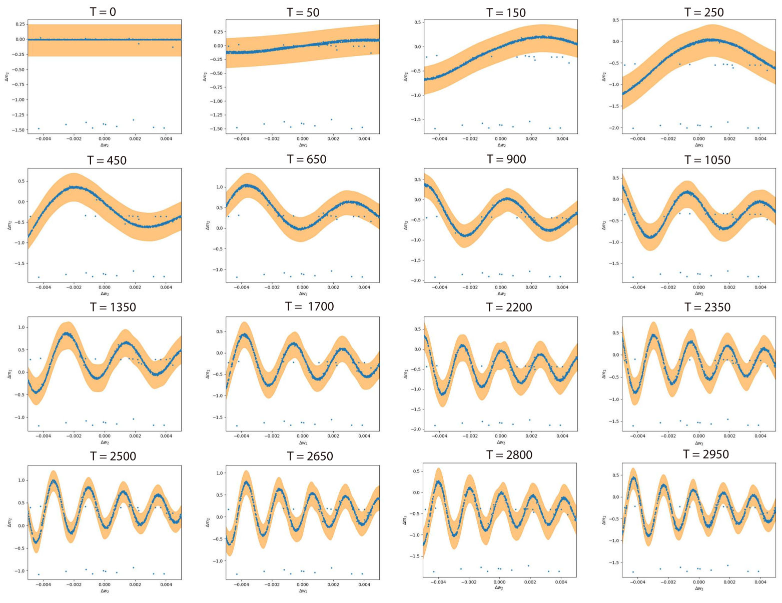


Figure 18: The time evolution of the learned GP models from directional derivatives for $\partial x_2/\partial k_p$ by uniform sampling (Sine 2 joints).

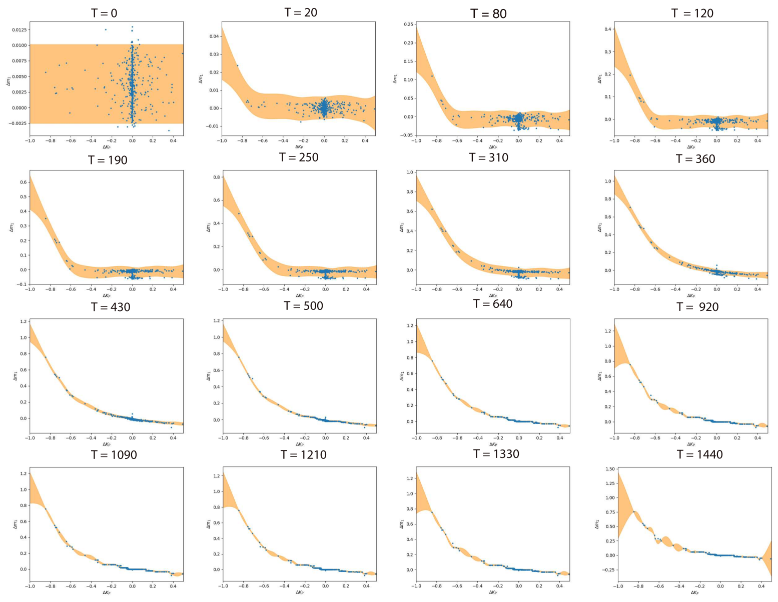


Figure 19: The time evolution of the learned GP models from directional derivatives for $\partial x_1/\partial k_p$ by Gaussian sampling (PD controller).

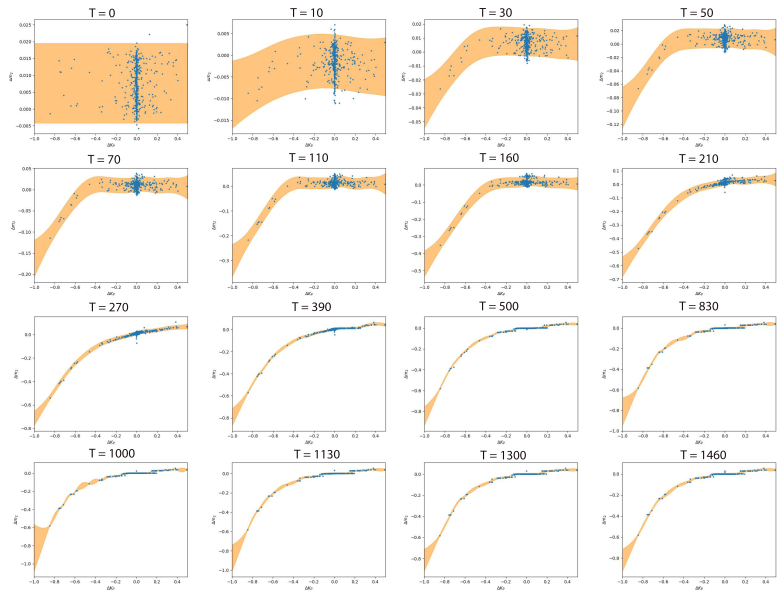


Figure 20: The time evoltuion of the learned GP models from directional derivatives for $\partial x_2/\partial k_p$ by Gaussian sampling (PD controller).

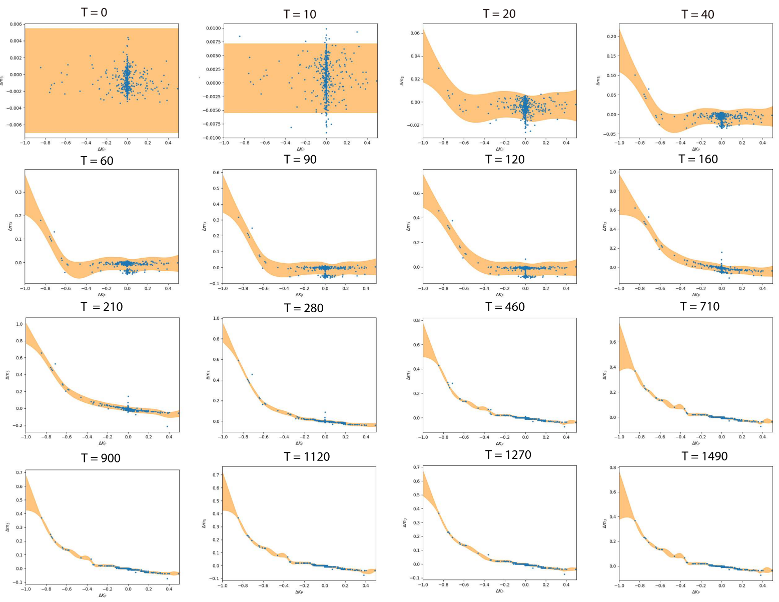


Figure 21: The time evolution of the learned GP models from directional derivatives for $\partial x_3/\partial k_p$ by Gaussian sampling (PD controller).

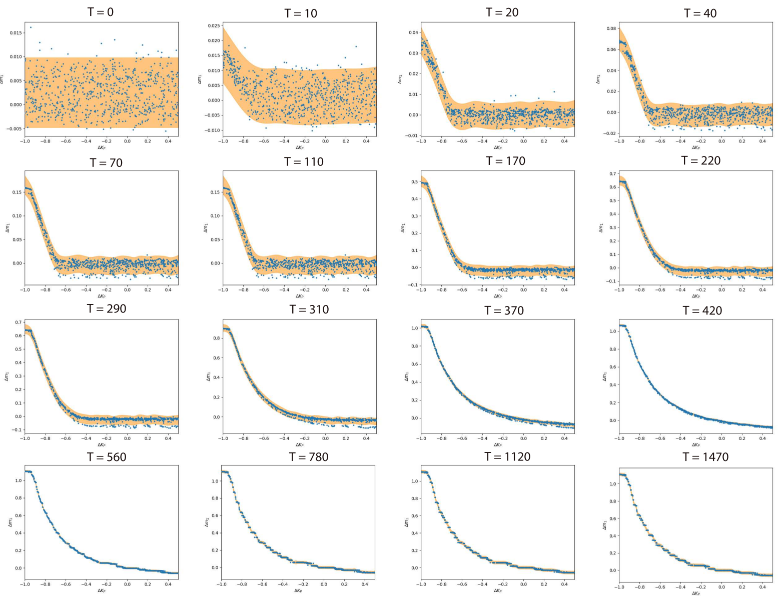


Figure 22: The time evolution of the learned GP models from directional derivatives for $\partial x_1/\partial k_p$ by uniform sampling (PD controller).

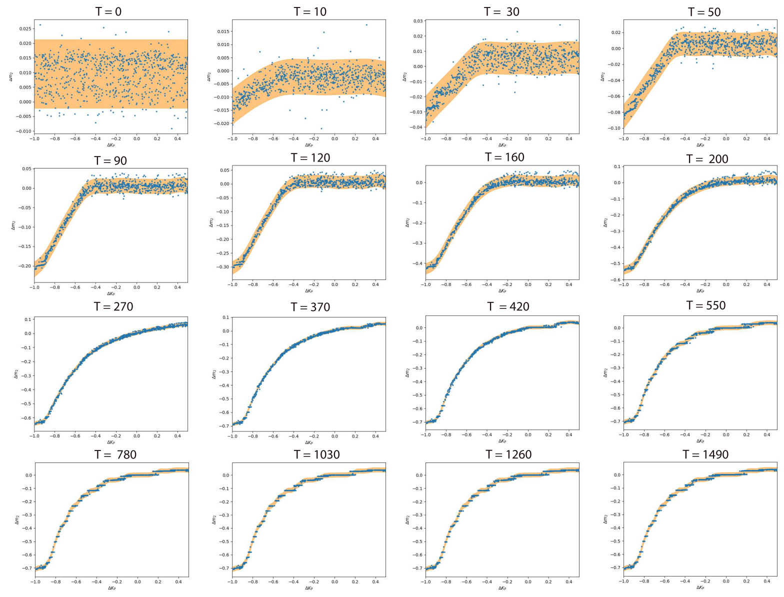


Figure 23: The time evolution of the learned GP models from directional derivatives for $\partial x_2/\partial k_p$ by uniform sampling (PD controller).

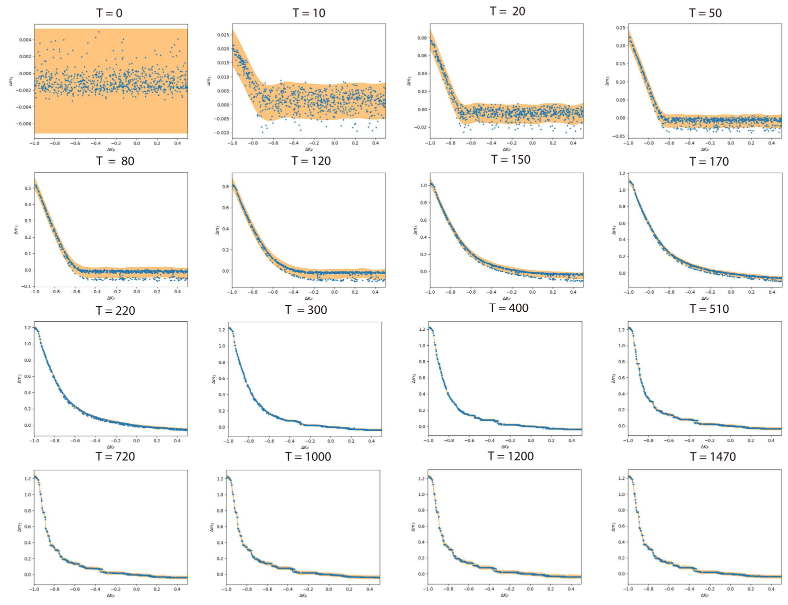


Figure 24: The time evolution of the learned GP models from directional derivatives for $\partial x_3/\partial k_p$ by uniform sampling (PD controller).

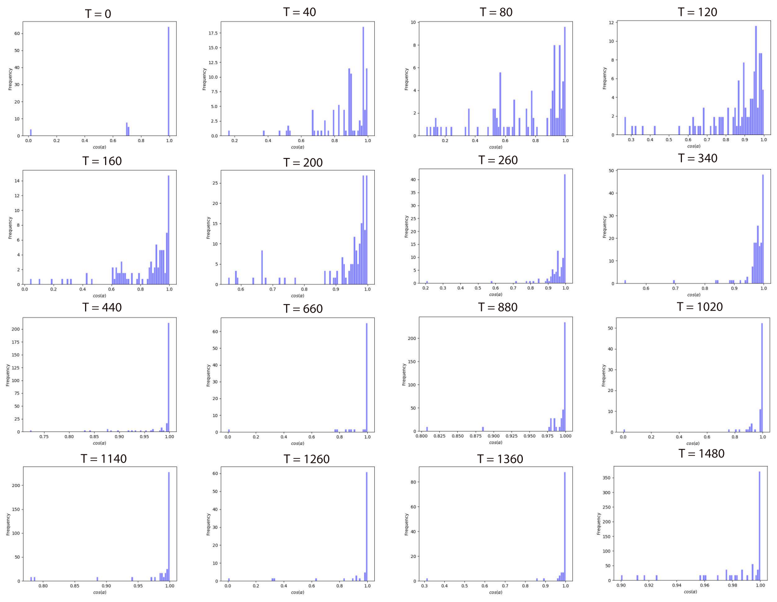


Figure 25: The time evolution of the histogram of $\cos(\alpha)$ where α is the angle between the true and predicted directional derivative. The perturbations in the training phase are generated by Gaussian sampling. As shown, angles of the derivatives are predicted mostly correctly (PD controller).

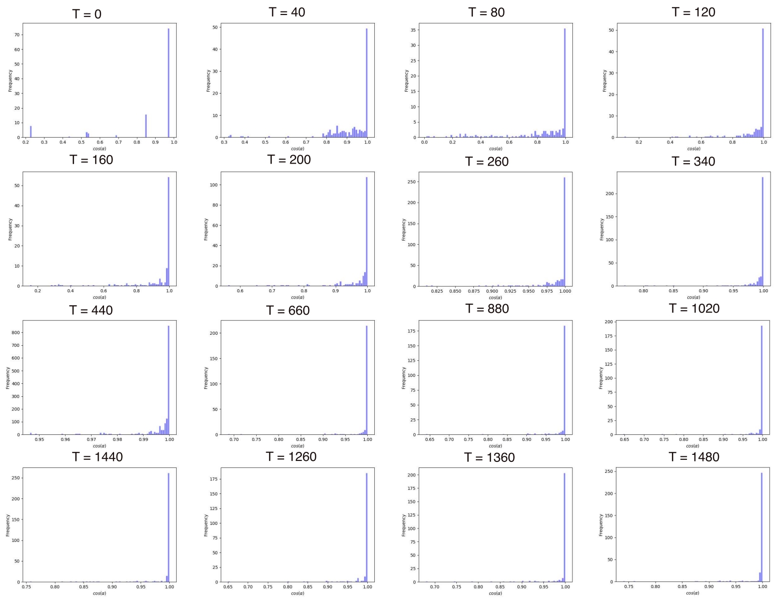


Figure 26: The time evolution of the histogram of $\cos(\alpha)$ where α is the angle between the true and predicted directional derivative. The perturbations in the training phase are generated by uniform sampling. As shown, angles of the derivatives are predicted mostly correctly (PD controller).

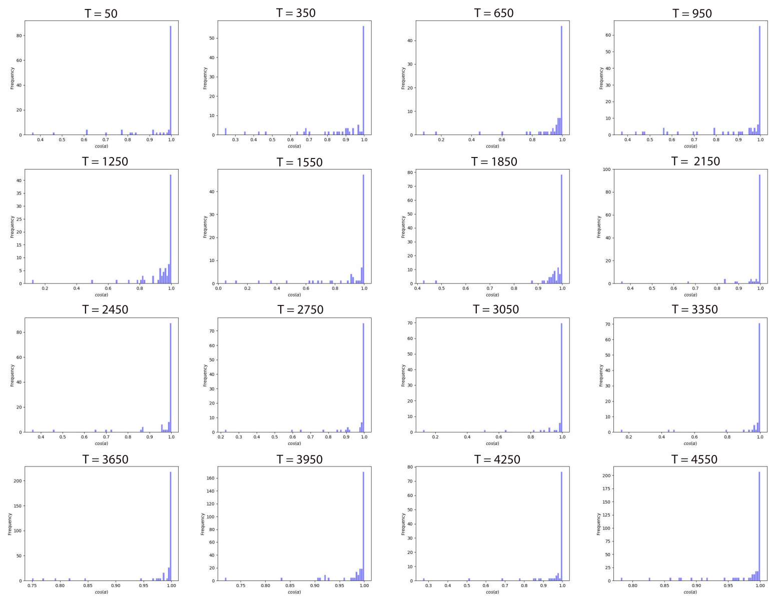


Figure 27: The time evolution of the histogram of $\cos(\alpha)$ where α is the angle between the true and predicted directional derivative. The perturbations in the training phase are generated by Gaussian sampling. As shown, angles of the derivatives are predicted mostly correctly (Sine 2 joints).

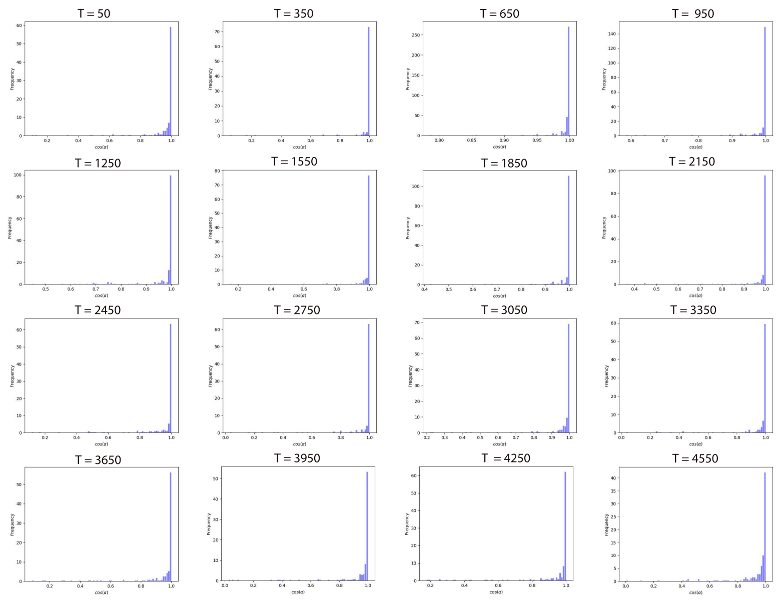
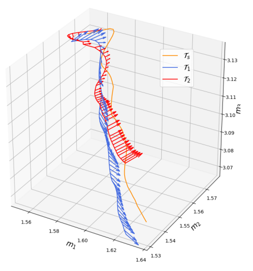
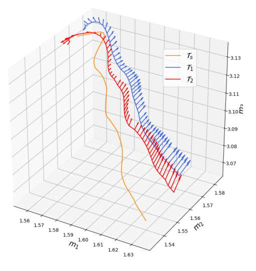
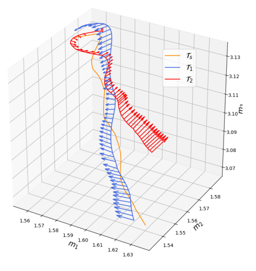
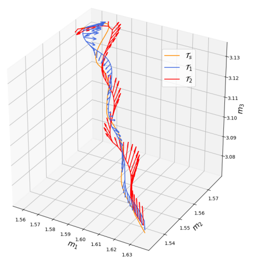


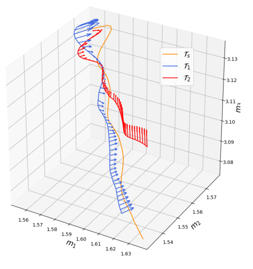
Figure 28: The time evolution of the histogram of $\cos(\alpha)$ where α is the angle between the true and predicted directional derivative. The perturbations in the training phase are generated by uniform sampling. As shown, angles of the derivatives are predicted mostly correctly (Sine 2 joints).

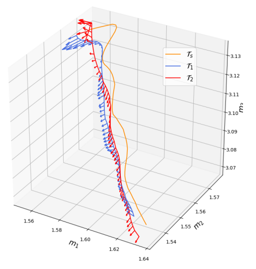

(a)

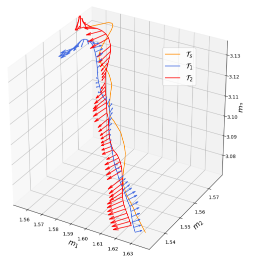

(b)

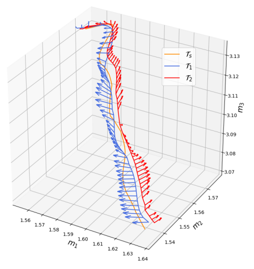

(c)

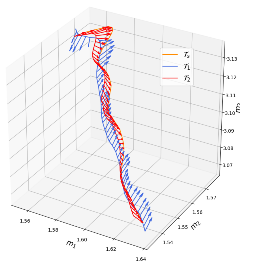

(d)

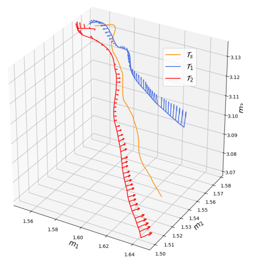

(e)

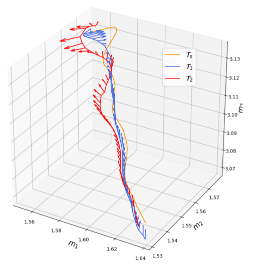

(f)


(g)


(h)


(i)


(j)


(k)

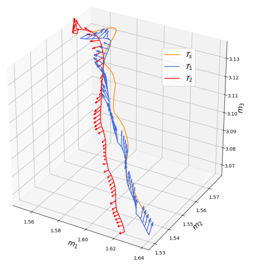

(l)

Figure 29: Examples of trajectories produced by the perturbed controller and the computed derivatives along the trajectory. The arrows are plotted as they originate from the perturbed trajectories only for easier distinction. Each arrow corresponds to the change of the states at a certain time step on the source trajectory as a result of perturbing the policy. Each figure corresponds to a pair of nominal values of $\{w, b\}$ for the linear open-loop controller of Appendix A.7.1 and the perturbed trajectories are produced by Gaussian sampling (The orange trajectory is the source trajectory $\mathcal{T}_s$ and others are the perturbed ones $\mathcal{T}_1, \mathcal{T}_2$. The quivers on the perturbed trajectories represent calculated physical derivatives).