Learning More With Less: Sample-Efficient Model-Based RL for Loco-Manipulation

Benjamin Hoffman
ETH Zürich, Switzerland
bhoffman@ethz.ch

Jin Cheng
ETH Zürich, Switzerland
jin.cheng@inf.ethz.ch

Chenhao Li
ETH Zürich, Switzerland
chenhli@ethz.ch

Stelian Coros
ETH Zürich, Switzerland
scoros@inf.ethz.ch

Abstract: By combining the agility of legged locomotion with the capabilities of manipulation, loco-manipulation platforms have the potential to perform complex tasks in real-world applications. To this end, state-of-the-art quadrupeds with manipulators, such as the Boston Dynamics Spot, have emerged to provide a capable and robust platform. However, the complexity of loco-manipulation control, as well as the black-box nature of commercial platforms, pose challenges for deriving accurate dynamics models and robust control policies. To address these challenges, we turn to model-based reinforcement learning (RL). We develop a hand-crafted kinematic model of a quadruped-with-arm platform which – employing recent advances in Bayesian Neural Network (BNN)–based learning – we use as a physical prior to efficiently learn an accurate dynamics model from limited data. We then leverage our learned model to derive control policies for loco-manipulation via RL. We demonstrate the effectiveness of our approach on state-of-the-art hardware using the Boston Dynamics Spot, accurately performing dynamic end-effector trajectory tracking even in low data regimes. Project website and videos: sites.google.com/view/learning-more-with-less.

Keywords: Sample Efficient Model-Based RL, Loco-Manipulation

1 Introduction

Legged robots have demonstrated impressive capabilities in navigating complex terrains, offering agility and adaptability as a result of continuous research efforts [1, 2, 3, 4, 5, 6]. However, while many systems excel at locomotion, their ability to interact with their environment remains limited. The integration of a manipulator onto a legged platform, i.e., legged loco-manipulation, holds the potential to bridge this gap. This combination enables a robot to both navigate challenging terrain and perform advanced manipulation tasks such as opening doors [7], grasping objects [8, 9, 10], or possibly even interacting with objects in a dynamic setting such as catching or throwing a ball. State-of-the-art commercial robots such as the Boston Dynamics Spot quadruped, now equipped with an arm, have emerged to provide a capable and robust platform to perform such tasks [8].

However, their proprietary, black-box nature complicates the development of an accurate dynamics model necessary to derive new control policies [8, 11]. Simplified or hand-crafted modeling approaches alone often fall short in face of unknown internal controller behavior and complex dynamics, while purely model-free learning can demand large amounts of real-world data. Further, realizing robust loco-manipulation introduces new challenges. The coupling between a dynamic, moving base and a mounted manipulator creates complex, high-dimensional dynamics that are difficult to capture and control using classical methods [10].

Workshop on Resource-Rational Robot Learning at CoRL 2025.

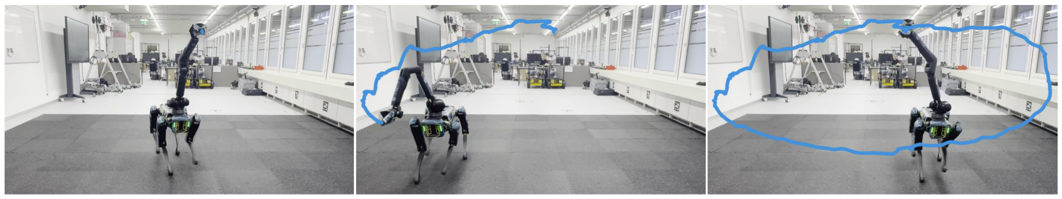


Figure 1: Boston Dynamics Spot in our experiments tracking an ellipsoidal reference trajectory.

To address both the challenge of learning an accurate system model, as well as performing robust loco-manipulation control, we turn to model-based reinforcement learning (RL) [12]. Leveraging recent progress in dynamics learning with Bayesian Neural Networks (BNNs), specifically SIM-FSVGD [13], and by developing a hand-crafted kinematic model of our platform, we efficiently learn a dynamics model of Spot with a mounted arm from real-world data. We use SIM-FSVGD to incorporate our kinematic model as a low-fidelity physical prior during BNN-learning, allowing us to learn an accurate model at low data requirements. Inspired by the recent success of RL-based control for loco-manipulation [14, 15, 10, 16], we then leverage our learned dynamics model to derive control policies via RL that enable Spot to accurately perform loco-manipulation tasks [17].

In summary, our main contributions are: (*i*) we allow for learning an accurate dynamics model for a complex, black-box quadruped-with-arm platform from limited real-world data by developing a hand-crafted kinematic model and employing it as a physical prior for efficient BNN-learning leveraging SIM-FSVGD, (*ii*) we use the learned dynamics model to derive control policies for loco-manipulation via RL, and (*iii*) we demonstrate the effectiveness of our approach on the Boston Dynamics Spot with a manipulator, achieving improved dynamic end-effector trajectory tracking accuracy even at reduced data requirements compared to baseline methods.

2 Related Work

Loco-Manipulation Control Combining legged locomotion and manipulation to achieve dynamic mobile manipulation is an increasingly relevant problem that has been the focus of a considerable amount of research. We can generally make a distinction between platforms that use a robot's body [18, 19] or legs [14, 16, 20] for manipulation, those that apply a hybrid approach [21, 22, 23], and those that use a dedicated arm [15, 10, 8, 9, 7, 24]. Especially this last category allows for combining the advances in legged locomotion with the benefits of a manipulator, enabling advanced tasks such as grasping stationary items [10, 8, 9], opening doors [7], or wiping a whiteboard [10].

However, due to the platform's complexity, loco-manipulation control is inherently a challenging, high-dimensional, and non-smooth control problem [15]. This calls for pursuing a closed-loop approach to allow for more robust and adaptive control instead of previous feed-forward trajectory optimization methods such as in [8]. To this end, RL has emerged as a powerful approach that enables robust legged locomotion in general [6, 25, 26, 27, 28, 29] and shows impressive performance in the setting of loco-manipulation [14, 15, 10, 16, 20, 19]. While these prior works mainly focus on using *model-free* RL, recently *model-based* RL has shown potential for more sample-efficient robot learning across distinct locomotion and manipulation tasks [30, 31]. Motivated by this success, we leverage recent advances in *model-based* RL and dynamics learning, primarily SIM-FSVGD [13], to achieve accurate and robust loco-manipulation control, as well as better generalization to new tasks, even when data is scarce and a simulation environment is not available.

Modeling Robot Dynamics Although state-of-the-art quadrupeds like the Boston Dynamics Spot are very capable platforms, their usage in a research setting yields certain challenges. As a commercial product, knowledge of their built-in controller remains proprietary, and low-level control access is often restricted. However, an accurate system model is necessary to pursue a model-based or learning-based control method. To this end, [8] uses a simple parametrized dynamics model and parameter identification to simulate the platform's behavior. Yet, accurately modeling the dynamics of a complex system such as a quadruped with an attached arm is a challenging, dynamic, and high-DoF problem [10, 17]. In [8], Spot fails to achieve the desired task in certain cases due to the model's inability to fully capture the robot's complex internal behavior. Especially the wrench and disturbances introduced by the arm are difficult to model accurately.

In the face of these challenges, we pursue a more sophisticated approach. By developing a handcrafted kinematic model of our system and employing methods for BNN-learning with physical priors, we incorporate domain knowledge into our learning pipeline to accurately capture the complex dynamics of our system in a sample-efficient manner. To this end, we leverage SIM-FSVGD [13], a BNN-based method to learn accurate dynamics from limited data by incorporating our kinematic model as a low-fidelity physical prior during training. Further, by using *Bayesian* Neural Networks, we tend to avoid the same overfitting behavior as non-bayesian methods [13]. This combined approach allows us to learn an accurate model of our system with improved sim-to-real performance, even in low-data regimes, which we can then leverage to derive robust control policies via RL.

3 Preliminaries

3.1 Learning Robot Dynamics

Learning with NNs We omit the preliminaries on (model-based) RL here and provide them in Appendix A. In a robotics context, we want to use model-based RL to derive control policies π that allow us to perform dynamic tasks, such as loco-manipulation. However, since we often lack a model of our system in practice, we first need to learn an accurate dynamics model of our robot. For this purpose, we consider a time-discretized dynamical system described as

$$\mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{u}_t), \tag{1}$$

where $\mathbf{s}_t \in \mathbb{R}^{n_s}$ is the state of the system at time t, $\mathbf{u}_t \in \mathbb{R}^{n_u}$ is the control input, and $f(\mathbf{s}_t, \mathbf{u}_t)$ is the unknown dynamics of the system. We now aim to learn a model $\hat{f}(\mathbf{s}_t, \mathbf{u}_t)$ that approximates the true dynamics of the system from a dataset of state-action-state transitions $(\mathbf{s}_t, \mathbf{u}_t, \mathbf{s}_{t+1})$. To this end, we can state our problem as learning an unknown dynamics function $\hat{f} : \mathcal{X} \to \mathcal{Y}$ from a dataset $\mathcal{D} = (\mathbf{X}^\mathcal{D}, \mathbf{y}^\mathcal{D})$ of size N, where our training inputs consist of the state-action pairs $\mathbf{X}^\mathcal{D} = \{\mathbf{x}_j\}_{j=1}^N$ and the target outputs $\mathbf{y}^\mathcal{D} = \{\mathbf{y}_j\}_{j=1}^N$ are the measured noisy observations of our dynamics, i.e., $\mathbf{y}_j = f(\mathbf{x}_j) + \epsilon_j$. We assume the noise ϵ to be i.i.d. and Gaussian with variance σ^2. Using a Neural Network (NN) to model $\hat{f}$, we can then formulate our learning problem as fitting a NN model $h_\theta : \mathcal{X} \to \mathcal{Y}$ with network weights θ from $\mathcal{D}$. We can use h_θ to define the conditional predictive distribution of our observations as $p(\mathbf{y}|\mathbf{x}, \theta) = \mathcal{N}(\mathbf{y}|h_\theta(\mathbf{x}), \sigma^2)$.

Learning with BNNs This can be extended to learning a BNN by considering not only a single set of weights but a distribution over θ. BNNs then infer a posterior distribution over the weights $p(\theta|\mathbf{X}^\mathcal{D}, \mathbf{y}^\mathcal{D}) \propto p(\mathbf{y}^\mathcal{D}|\mathbf{X}^\mathcal{D}, \theta)p(\theta)$ given the data-likelihood $p(\mathbf{y}^\mathcal{D}|\mathbf{X}^\mathcal{D}, \theta)$ and a known prior distribution $p(\theta)$. Under the assumption that, given θ, each data point is conditionally independent, the likelihood can be factorized as $p(\mathbf{y}^\mathcal{D}|\mathbf{X}^\mathcal{D}, \theta) = \prod_{j=1}^N p(\mathbf{y}_j|\mathbf{x}_j, \theta)$. Finally, the predictive distribution for a new input $\mathbf{x}^*$ can be defined by marginalizing over the weights θ [13] as $p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{X}^\mathcal{D}, \mathbf{y}^\mathcal{D}) = \int p(\mathbf{y}^*|\mathbf{x}^*, \theta)p(\theta|\mathbf{X}^\mathcal{D}, \mathbf{y}^\mathcal{D})d\theta = \mathbb{E}_\theta[p(\mathbf{y}^*|\mathbf{x}^*, \theta)|\mathbf{X}^\mathcal{D}, \mathbf{y}^\mathcal{D}]$.

3.2 Function Space Inference and Functional Priors

FSVGD Performing posterior inference with BNNs, however, is challenging. Both the high-dimensionality of the weight space, as well as the over-parametrization of the mapping between θ and a likelihood function $p(\mathbf{y}^\mathcal{D}|\mathbf{X}^\mathcal{D}, \theta)$ render inference difficult [13, 32]. The functional Stein Variational Gradient Descent (FSVGD) method [32] addresses these issues by performing BNN inference in the space of regression functions $h : \mathcal{X} \to \mathcal{Y}$, rather than in the weight space of θ. In function space, the posterior is formulated as $p(h|\mathbf{X}^\mathcal{D}, \mathbf{y}^\mathcal{D}) \propto p(\mathbf{y}^\mathcal{D}|\mathbf{X}^\mathcal{D}, h)p(h)$, where $p(h)$ is a stochastic prior distribution over $h : \mathcal{X} \to \mathcal{Y}$ with index space $\mathcal{X}$ and value space $\mathcal{Y}$ [33]. This allows for the functional inference to be restated in a tractable form by using finite measurement sets $\mathbf{X} := [\mathbf{x}_1, ..., \mathbf{x}_k] \in \mathcal{X}^k, k \in \mathbb{N}$ that allow for characterizing a stochastic process by marginals of function values $\rho(\mathbf{h}^{\mathbf{X}}) := \rho(h(\mathbf{x}_1), ..., h(\mathbf{x}_k))$ and subsequently stating the functional posterior as $p(\mathbf{h}^{\mathbf{X}}|\mathbf{X}, \mathbf{X}^{\mathcal{D}}, \mathbf{y}^{\mathcal{D}}) \propto p(\mathbf{y}^{\mathcal{D}}|\mathbf{X}^{\mathcal{D}}, \mathbf{h}^{\mathbf{X}})p(\mathbf{h}^{\mathbf{X}})$, for the measurement sets $\mathbf{X}$ [34]. FSVGD approximates this posterior by maintaining L parameter particles $\theta_1, ..., \theta_L$ and iteratively re-sampling $\mathbf{X}$ as a random subset of $\mathcal{X}$ with $\mathbf{X} \sim \mu$ where μ is an arbitrary distribution supported on $\mathcal{X}$. FSVGD then updates the particles using

$$\theta_l = \theta_l - \gamma J_l u_l. \tag{2}$$

Here $J_l = (\nabla_{\theta_l} \mathbf{h}^{\mathbf{X}}_{\theta_l})^\top$ is the NN Jacobian, $u_l = \frac{1}{L}\sum_{i=1}^{L} \mathbf{K}_{li} \nabla_{\mathbf{h}^{\mathbf{x}}_{\theta_i}} \ln p(\mathbf{h}^{\mathbf{x}}_{\theta_l}|\mathbf{X}, \mathbf{X}^D, \mathbf{y}^D) + \nabla_{\mathbf{h}^{\mathbf{x}}_{\theta_l}} \mathbf{K}_{li}$ is the SVGD update [35] in function space and $\mathbf{K} = [k(\mathbf{h}^{\mathbf{x}}_{\theta_l}, \mathbf{h}^{\mathbf{x}}_{\theta_i})]_{li}$ is the gram matrix, based on a kernel function k, between the measurement points and the function values [13, 32].

SIM-FSVGD The SIM-FSVGD method [13] extends FSVGD by using an *informed* functional prior $p(h)$ for a function $h : \mathcal{X} \to \mathcal{Y}$ that incorporates both a *domain-model process* and a *sim-to-real prior*. SIM-FSVGD factorizes the prior over the output dimensions as $p(h) = \prod_{i=1}^{n_s} p(h_i)$, treating each $h_i : \mathbf{X} \to \mathbb{R}$ as an independent function. The *domain-model process* allows for integrating prior domain knowledge of the system via a low-fidelity simulation model $g(\mathbf{x}, \phi)$, e.g., derived from first-principle physics, where ϕ are the model parameters. As the exact model parameters are unknown, we can randomly sample them from a plausible range as $\phi \sim p(\phi)$ and create distinct simulation models per parameter set, implicitly creating a stochastic process of functions. The *sim-to-real prior* addresses the gap between a simulation model and the actual system dynamics $f(\mathbf{x})$ by adding a sim-to-real gap process as a Gaussian Process (GP) $p(\tilde{h}_i)$ per output dimension $i = 1, ..., n_s$. SIM-FSVGD uses a zero-mean GP with isotropic kernel $k(\mathbf{x}, \mathbf{x}') = \nu^2 \rho(||\mathbf{x} - \mathbf{x}'||/l)$, where the lengthscale l and variance ν^2 are hyperparameters that allow us to incorporate our assumptions about the actual sim-to-real gap.

The combined stochastic process prior $p(h)$ is then defined implicitly via the marginal distributions implied by independently sampling conditional random vectors from each process and adding them: $\mathbf{h}^{\mathbf{x}}_i = [g_i(\mathbf{x}_1, \phi), \ldots, g_i(\mathbf{x}_k, \phi)]^\top + \tilde{\mathbf{h}}^{\mathbf{x}}_i$, where $\phi \sim p(\phi)$ and $\tilde{\mathbf{h}}^{\mathbf{x}}_i \sim \mathcal{N}(\tilde{\mathbf{h}}^{\mathbf{X}}_i|0, \mathbf{K})$ [13]. Lastly, SIM-FSVGD relies on the same update rule as in Equation (2) to update the particles θ_l. The stochastic process prior score $\nabla_{\mathbf{h}^{\mathbf{x}}} \ln p(\mathbf{h}^{\mathbf{X}}) = \sum_i^{n_s} \nabla_{\mathbf{h}^{\mathbf{x}}_i} \ln p(\mathbf{h}^{\mathbf{X}}_i)$ is approximated using a Gaussian approximation of the prior process, i.e., $p(\mathbf{h}^{\mathbf{X}}_i) \sim \mathcal{N}(\mu^{\mathbf{X}}_i, \Sigma^{\mathbf{X}}_i)$. The approximation is constructed by sampling the measurement set $\mathbf{X}$ from a distribution μ supported on $\mathcal{X}$, sampling $m = 1, ..., P$ vectors of function values $\mathbf{h}^{\mathbf{X}}_{i,m} \sim p(\mathbf{h}^{\mathbf{X}}_i)$ and computing their mean $\mu^{\mathbf{X}}_i$ and covariance $\Sigma^{\mathbf{X}}_i$.

4 Learning Control Policies for Loco-Manipulation

Our work focuses on learning an accurate dynamics model of our robot from limited data to subsequently derive control policies for loco-manipulation tasks via RL. To this end, we develop a hand-crafted kinematic model $\hat{f}_{kin}$ of our quadruped-with-arm platform. We then use $\hat{f}_{kin}$ as a physical prior and leverage SIM-FSVGD to efficiently learn a model $\hat{f}$ from data that approximates our platform's true dynamics f. Subsequently, we develop a reward structure r and use our learned model $\hat{f}$ to derive a control policy π for end-effector trajectory tracking using Soft-Actor-Critic (SAC) [36]. In this section, we present our control approach, develop our kinematic model $\hat{f}_{kin}$, show how we use it and SIM-FSVGD to learn $\hat{f}$, and finally discuss our policy learning process.

4.1 Robot State and Control Input

We define our base state as the position, orientation and velocity of the robot's base on a 2D plane in world frame W, i.e., $\mathbf{p}^{\text{base}} = [x^{\text{base}}, y^{\text{base}}, \theta^{\text{base}}]$ and $\mathbf{v}^{\text{base}} = [v_x^{\text{base}}, v_y^{\text{base}}, \omega^{\text{base}}]$. Here θ^{base} is the yaw angle of the robot's base, and ω^{base} is the corresponding angular velocity. For the end-effector, we represent the state using the 3D position and velocity in world frame W, i.e., $\mathbf{p}^{\text{ee}} = [x^{\text{ee}}, y^{\text{ee}}, z^{\text{ee}}]$ and $\mathbf{v}^{\text{ee}} = [v_x^{\text{ee}}, v_y^{\text{ee}}, v_z^{\text{ee}}]$. Stacking the individual components, our state vector becomes $\mathbf{s} = \left[\mathbf{p}^{\text{base}}, \mathbf{v}^{\text{base}}, \mathbf{p}^{\text{ee}}, \mathbf{v}^{\text{ee}}\right] \in \mathbb{R}^{12}$. We provide an overview of the respective frames in Appendix B.

To control our robot, we use velocity commands. For the base of the robot, we command the v_x^{base}, v_y^{base} and ω^{base} velocities on a 2D plane, yielding the control input $\mathbf{u}^{\text{base}} = [u_{vx}^{\text{base}}, u_{vy}^{\text{base}}, u_{\omega}^{\text{base}}]$. We express the velocities in the body frame B and apply them at the robot's center of mass. For the end-effector, we command the v_x^{ee}, v_y^{ee} and v_z^{ee} velocities in 3D, i.e., $\mathbf{u}^{\text{ee}} = [u_{vx}^{\text{ee}}, u_{vy}^{\text{ee}}, u_{vz}^{\text{ee}}]$. Again, the velocities are expressed in the body frame B and are applied at the center of the end-effector frame H, which sits at the center of the gripper. Our control input is then given as $\mathbf{u} = \left[\mathbf{u}^{\text{base}}, \mathbf{u}^{\text{ee}}\right] \in \mathbb{R}^6$.

4.2 Developing a Dynamics Model

We model the dynamics of our robot as the time discretized system $\mathbf{s}_{t+1} = \hat{f}(\mathbf{s}_t, \mathbf{u}_t)$, where $\mathbf{s}_t \in \mathbb{R}^{n_s}$ is the state at time t with $n_s = 12$, $\mathbf{u}_t \in \mathbb{R}^{n_u}$ is the control input with $n_u = 6$, and $\hat{f}(\mathbf{s}_t, \mathbf{u}_t)$ are the approximated dynamics. We now develop our dynamics model in two steps. First, we create a hand-crafted kinematic model $\hat{f}_{kin}$ of our robot derived from first principles using domain knowledge. In a second step, we use our model as a physical prior, i.e., to create the *domain-model process* within SIM-FSVGD, and efficiently learn a dynamics model $\hat{f}$ from real-world data.

Kinematic Model We derive our kinematic model from first principles and use the Forward Euler Method with time step Δt to propagate our state. To better capture the complex dynamics of our platform, we enhance our equations with the parameters $\alpha \in \mathbb{R}^{6\times1}$, $\beta \in \mathbb{R}^{12\times1}$, and $\gamma \in \mathbb{R}^{6\times1}$. As our actions $\mathbf{u}_t$ are given in the body frame B and our state $\mathbf{s}_t$ is in the world frame W, we first convert our inputs to $\mathbf{u}_t^W$ using the base's current yaw angle θ_t^{base}, i.e.,

$$\mathbf{u}_t^W = \begin{bmatrix} R(\theta_t^{\text{base}})\,\mathbf{u}_t^{\text{base},B} \\ R(\theta_t^{\text{base}})\,\mathbf{u}_t^{\text{ee},B} \end{bmatrix}, \quad \text{where} \quad R(\theta) := \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{3}$$

We can then derive the update equations for the base velocity $\mathbf{v}^{\text{base}}$ and position $\mathbf{p}^{\text{base}}$, using $\mathbf{A}_{\text{base}} := \operatorname{diag}(\boldsymbol{\alpha}_{1:3})$ and $\boldsymbol{\Gamma}_{\text{base}} := \operatorname{diag}(\boldsymbol{\gamma}_{1:3})$, as

$$\begin{aligned} \mathbf{v}_{t+1}^{\text{base}} &= \mathbf{A}_{\text{base}}\,\mathbf{v}_t^{\text{base}} + (\mathbf{I}_3 - \mathbf{A}_{\text{base}})\mathbf{u}_t^{\text{base},W} + \boldsymbol{\beta}_{1:3}, \\ \mathbf{p}_{t+1}^{\text{base}} &= \mathbf{p}_t^{\text{base}} + \Delta t\,\boldsymbol{\Gamma}_{\text{base}}\,\mathbf{v}_{t+1}^{\text{base}} + \boldsymbol{\beta}_{4:6}. \end{aligned} \tag{4}$$

For the end-effector updates, we need to consider the base's movements in addition to the end-effector velocity commands. Incorporating the horizontal linear velocities of the base $v_{x,t}^{\text{base}}$ and $v_{y,t}^{\text{base}}$ follows simply via addition. However, to consider the base's angular velocity ω_t^{base} in the end-effector's movement, we first need to calculate the induced velocity on the end-effector as $\mathbf{v}^{\text{ind}} = \left[-\omega_{t+1}^{\text{base}} \cdot d \cdot \sin(\phi), \omega_{t+1}^{\text{base}} \cdot d \cdot \cos(\phi), 0\right]$. Here $d = \sqrt{\left(x_t^{\text{ee}} - x_t^{\text{base}}\right)^2 + \left(y_t^{\text{ee}} - y_t^{\text{base}}\right)^2}$ is the distance between the base's rotational axis and the end-effector, and $\phi = \operatorname{arctan2}\left(y_t^{\text{ee}} - y_t^{\text{base}}, x_t^{\text{ee}} - x_t^{\text{base}}\right)$ is the angle of the induced velocity vector in the global frame W. We can then update the end-effector velocity $\mathbf{v}^{\text{ee}}$ and position $\mathbf{p}^{\text{ee}}$, again using $\mathbf{A}_{\text{ee}} := \operatorname{diag}(\boldsymbol{\alpha}_{4:6})$, $\boldsymbol{\Gamma}_{\text{ee}} := \operatorname{diag}(\boldsymbol{\gamma}_{4:6})$ and $\mathbf{D}_{ee} := \operatorname{diag}(1, 1, 0)$, as

$$\begin{aligned} \mathbf{v}_{t+1}^{\text{ee}} &= \mathbf{A}_{\text{ee}}\,\mathbf{v}_t^{\text{ee}} + (\mathbf{I}_3 - \mathbf{A}_{\text{ee}})\,\mathbf{u}_t^{\text{ee},W} + \boldsymbol{\beta}_{7:9} + \mathbf{D}_{ee}\,\mathbf{v}_{t+1}^{\text{base}} + \mathbf{v}^{\text{ind}}, \\ \mathbf{p}_{t+1}^{\text{ee}} &= \mathbf{p}_t^{\text{ee}} + \Delta t\,\boldsymbol{\Gamma}_{\text{ee}}\,\mathbf{v}_{t+1}^{\text{ee}} + \boldsymbol{\beta}_{10:12}. \end{aligned} \tag{5}$$

BNN Model To learn a BNN model of our robot from data, we leverage SIM-FSVGD [13], as detailed in section 3. SIM-FSVGD allows us to incorporate our prior knowledge by using our kinematic model $\hat{f}_{kin}$ to create the *domain-model process*, where our system parameters ϕ are the set of parameters used in our update equations, i.e., $\phi = [\alpha, \beta, \gamma] \in \mathbb{R}^{24\times1}$. Note that, as SIM-FSVGD randomly samples parameter sets as $\phi \sim p(\phi)$ to implicitly create a stochastic process of functions, we do *not* fit the parameters of our kinematic model from data beforehand. However, we use real-world data to heuristically estimate a plausible range for our parameters. We then use a similar *sim-to-real prior* as [13] and learn our dynamics model $\hat{f}$ from real-world data.

4.3 Policy Learning

Having learned a model of our system, we now turn to deriving loco-manipulation control policies via RL. We employ SAC [36] and condition our policy on end-effector goal positions $\mathbf{g}^{\text{ee}} = [x_g^W, y_g^W, z_g^W]$. To this end, we uniformly sample an initial state $\mathbf{s}_0$ and end-effector goal position $\mathbf{g}^{\text{ee}}$ at each episode and construct a goal conditioned state vector $\mathbf{s}_{\text{cond}} = [\mathbf{s}, \mathbf{g}^{\text{ee}}] \in \mathbb{R}^{15\times 1}$, while our action space remains our control input $\mathbf{u} \in \mathbb{R}^{6\times 1}$. To guide our policy learning, we design a reward structure $r(\mathbf{s}_{\text{cond}}, \mathbf{u})$ that encourages the end-effector to smoothly move towards the goal while keeping the end-effector within a physically valid range, consisting of a state-goal distance reward r_{state}, an end-effector to base distance reward $r_{\text{ee-base}}$ and a regularizing action reward r_{action}.

r_{state} To drive the end-effector towards a goal position, we assign a full reward when the distance $d_{\text{ee-goal}} = \|\mathbf{p}^{\text{ee}} - \mathbf{g}^{\text{ee}}\|$ between the end-effector position $\mathbf{p}^{\text{ee}}$ and the goal position $\mathbf{g}^{\text{ee}}$ lies within $(0, b)$. Outside these bounds, we smoothly decrease the reward using a long-tailed sigmoid function $\sigma_{m,a}(x)$ [1], with a defined value a at margin m, creating a smooth reward with infinite-support and range $[0, 1]$: $r_{\text{state}}(\mathbf{s}_{\text{cond}}) = \mathbb{1}_{\{d_{\text{ee-goal}} \le b\}} + \mathbb{1}_{\{d_{\text{ee-goal}} > b\}}\, \sigma_{m,a}(d_{\text{ee-goal}} - b)$.

$r_{ee\text{-}base}$ The second component encourages the distance $d_{\text{ee-base}} = \|\mathbf{p}^{\text{ee}} - \mathbf{p}^{\text{base}}\|$ between the end-effector position $\mathbf{p}^{\text{ee}}$ and the base position $\mathbf{p}^{\text{base}}$ to stay within a physically valid range given by the arm's length l_{arm}. To achieve this, we use the same approach as above with the bounds $(0, l_{\text{arm}})$, resulting in: $r_{\text{ee-base}}\big(\mathbf{s}_{\text{cond}}\big) = \mathbb{1}_{\{d_{\text{ee-base}} \le l_{\text{arm}}\}} + \mathbb{1}_{\{d_{\text{ee-base}} > l_{\text{arm}}\}}\, \sigma_{m,a}\big(d_{\text{ee-base}} - l_{\text{arm}}\big)$.

r_{action} Lastly, we include an action cost term that penalizes inefficient policies. However, instead of weighting each control input equally, we encourage the use of end-effector movements over body movements by assigning a higher weight λ_{base} to the base actions than the end-effector actions weight λ_{ee}: $r_{\text{action}}(\mathbf{u}) = -\big(\lambda_{\text{base}}\|\mathbf{u}^{\text{base}}\|^2 + \lambda_{\text{ee}}\|\mathbf{u}^{\text{ee}}\|^2\big)$.

We calculate our final reward as a weighted sum of the three components, i.e., $r(\mathbf{s}_{\text{cond}}, \mathbf{u}) = w_1 r_{\text{state}}(\mathbf{s}_{\text{cond}}) + w_2 r_{\text{ee-base}}(\mathbf{s}_{\text{cond}}) + w_3 r_{\text{action}}(\mathbf{u})$, where w_1, w_2, and w_3 are tuned heuristically.

5 Experimental Results

In this section, we present our experiments and results, evaluating the effectiveness of our model-based RL approach at learning dynamic loco-manipulation control policies for a complex quadruped-with-arm platform in a data-efficient manner. To this end, we compare the performance of the dynamics model learned using SIM-FSVGD with our kinematic model as a physical prior, which we simply label SIM-FSVGD, to two baseline models, SIM-MODEL and FSVGD, across different training set sizes. We evaluate both the sim-to-real transfer performance of the models, as well as the real-world loco-manipulation performance of the control policies derived from them.

In the following, we describe our baseline models, our experiment platform (the Boston Dynamics Spot), as well as our data collection and data processing steps. Subsequently, we introduce our three experiments: *Model Validation* and our hardware experiments *Ellipse Tracking* and *Helix Tracking*. Finally, we present the results of our evaluation.

5.1 Baseline Models

We consider two baseline models, SIM-MODEL and FSVGD [32]. SIM-MODEL is our hand-crafted kinematic model with the parameters α, β, and γ fitted from real-world data using the optimizer *Adam* [38]. Notably, we use an *unfitted* version of the same kinematic model as a low-fidelity physical prior in the SIM-FSVGD approach. FSVGD is a BNN-based method widely applied in deep learning and, contrary to SIM-FSVGD, FSVGD does *not* use an informed prior.

[1] We borrow the definition from [37], i.e., $\sigma_{m,a}(x) = \left(\left(xm^{-1}\sqrt{a^{-1}-1}\right)^2 + 1\right)^{-1}$

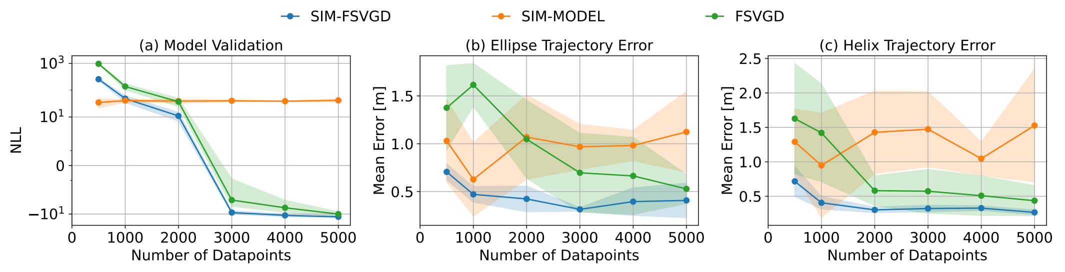


Figure 2: (a) We compare the dynamics models' sim-to-real transfer performance across increasing training set sizes by evaluating their NLL scores. SIM-FSVGD consistently outperforms the FSVGD baseline (especially in low data regimes) and the SIM-MODEL baseline from $N \geq 2000$ onwards. (b) We compare the mean error our policies achieve on the ellipsoidal goal trajectory across different training set sizes. The policies learned using the SIM-FSVGD model outperform those learned using our baseline models across all set sizes and especially at smaller training set sizes ($N < 3000$). (c) Similarly, for the helix trajectory, the policies learned using the SIM-FSVGD model outperform those learned using FSVGD and our kinematic SIM-MODEL across all training set sizes.

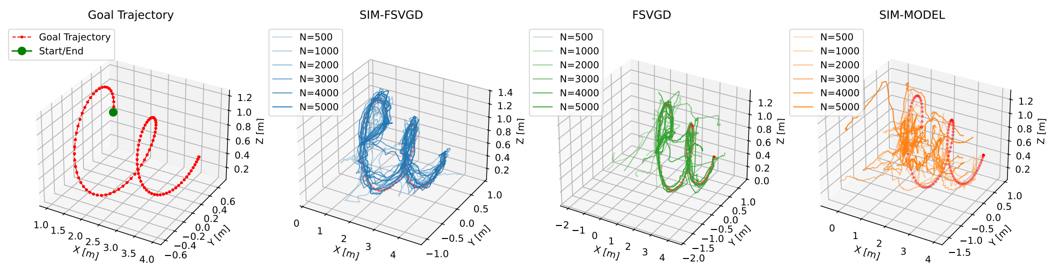


Figure 3: We plot the realized helix trajectories for all sample sizes and seeds. The policies learned using the SIM-FSVGD model follow the reference trajectory more closely than both baselines. Note that trajectories on course for collision were stopped early and we then plot only a truncated version.

5.2 Experiment Setup

Boston Dynamics Spot Quadruped For our experiments, we use the Boston Dynamics Spot, a state-of-the-art quadruped robot equipped with an arm for manipulation. We pass our control input and collect state measurements over Wi-Fi from a PC via Spot's high-level Python SDK. Spot relies on an unkown state estimator that fuses data from onboard sensors and cameras, and an unkown onboard controller that executes our commands. Appendix B shows Spot and its reference frames.

Data Collection and Processing We collect our training data by interacting with Spot, manually controlling its base $\mathbf{v}^{\text{base}}$ velocities using an Xbox controller and its end-effector $\mathbf{v}^{\text{ee}}$ velocities using a 3D Space Mouse. We send commands and collect data at $15\,\text{Hz}$. We collect transitions consisting of the current state $\mathbf{s}_t$, the commanded action $\mathbf{u}_t$, and the next state $\mathbf{s}_{t+1}$. However, instead of using our dynamics model to predict the next system state directly, we predict the change in the state. To this end, we adapt our dataset by creating a new set where our input remains the state action pair $\mathbf{x}_t = [\mathbf{s}_t, \mathbf{u}_t]$ but our target output becomes the state difference $\mathbf{y}_t = [\mathbf{s}_{t+1} - \mathbf{s}_t]$. Additionally, we encode the base's yaw angle θ^{base} as $(\sin(\theta^{base}), \cos(\theta^{base}))$ to avoid discontinuities and provide a representation more suitable for NNs [39]. Further, our control setup has a delay (ca. two timesteps, i.e., $133\,\text{ms}$) between the command and execution of an action. To compensate for this, we append the previous two actions $[\mathbf{u}_{t-2}, \mathbf{u}_{t-1}]$ to the state $\mathbf{s}_t$. Our model input becomes $\mathbf{x}_t = [\mathbf{s}_t, \mathbf{u}_{t-2}, \mathbf{u}_{t-1}, \mathbf{u}_t] \in \mathbb{R}^{31\times1}$ while our target output is $\mathbf{y}_t = [\mathbf{s}_{t+1} - \mathbf{s}_t] \in \mathbb{R}^{13\times1}$. We then create a training set for supervised learning by sampling i.i.d. from the collected transitions.

5.3 Results

Model Validation Spot's unknown internal controller behavior, along with the wrench and disturbances introduced by its arm, give rise to dynamics that are difficult to capture and model accurately. To evaluate how well our dynamics learning approach bridges this sim-to-real gap, we train the SIM-FSVGD and FSVGD models on a dataset sampled i.i.d. from our collected transitions and use the same dataset to fit the SIM-MODEL parameters. We evaluate their sim-to-real performance using the negative log-likelihood (NLL) scores they achieve on a real-world test set and compare our SIM-FSVGD model's performance to the baseline models across increasing training set sizes. We repeat our experiment with three random seeds and average the results.

We observe that SIM-FSVGD outperforms FSVGD across all training set sizes (Figure 2). Especially in low data regimes, leveraging our physical prior via SIM-FSVGD helps us achieve NLL scores significantly lower than FSVGD. At $N = 1000$, SIM-FSVGD performs similarly to our hand-crafted kinematic SIM-MODEL and surpasses it beyond $N \geq 2000$. While the BNN-based models improve with growing training set sizes, our kinematic SIM-MODEL's NLL scores remain consistent throughout, which is expected as we fit 24 parameters from an abundant amount of data.

Shape Tracking We now leverage our learned dynamics to derive loco-manipulation control policies via RL. To evaluate the effectiveness of our approach on hardware, we use the learned policies to dynamically track two shapes with Spot's end-effector: an ellipse and a helix. We show Spot in action in Figure 1. We compare the performance of the policies learned using our SIM-FSVGD dynamics model to the baselines across increasing training set sizes, repeating the experiment with three random seeds. We evaluate the performance via the mean error of the realized trajectories versus the reference trajectory: $\frac{1}{T}\sum_{t=1}^{T}||\mathbf{p}_t^{\mathrm{ee}} - \mathbf{g}_t^{\mathrm{ee}}||_2$, where $\mathbf{g}_t^{\mathrm{ee}}$ is the end-effector goal at time t.

The policies learned using our SIM-FSVGD model outperform both baselines across all training set sizes (Figure 2). Especially in low-data regimes ($N \leq 2000$), leveraging our physical prior via SIM-FSVGD allows us to still learn an accurate dynamics model, enabling us to derive control policies that achieve significantly lower errors than both the FSVGD and the SIM-MODEL baselines (147% and 153% higher at $N = 2000$ for the ellipse, respectively). While the performance of the policies learned using the FSVGD model improves with increasing training set sizes (as the model accuracy itself improves), even at $N = 5000$, their error is still larger than that of the policies learned using the SIM-FSVGD model at $N = 1000$. The plotted trajectories underline our results (Figure 3 and Appendix C); the policies learned using the SIM-FSVGD model follow the reference trajectory more closely than both baseline methods. These hardware results demonstrate the effectiveness of our approach at efficiently learning loco-manipulation control policies for a complex platform.

6 Conclusion

In this work, we address the problem of learning policies for loco-manipulation control on a quadruped platform with a manipulator. We develop a hand-crafted kinematic model which, by employing advances in dynamics learning with BNNs (i.e., SIM-FSVGD [13]), we leverage as a physical prior to efficiently learn a dynamics model of our system from limited data. In our hardware experiments, we use our learned dynamics model to derive loco-manipulation policies via RL, achieving improved dynamic end-effector trajectory tracking accuracy even at reduced data requirements compared to baseline methods. Our results demonstrate the effectiveness of our approach on a complex, commercial loco-manipulation system with a proprietary, black-box nature, such as Spot.

Limitations Our approach shows certain shortcomings that could be addressed in future work. Although our tracked trajectories cover 3D space, they do not fully exploit the dynamic capabilities of the platform. Future work could investigate trajectories that require even faster motion of the base, such as catching a ball, or longer trajectories. Further, our current state and action space do not yet include the end-effector's orientation. Exploring how to incorporate these additional degrees of freedom into our model and control policies could be beneficial for performing more complex tasks.

References

[1] Z. Zhuang, Z. Fu, J. Wang, C. Atkeson, S. Schwertfeger, C. Finn, and H. Zhao. Robot Parkour Learning. In *Conference on Robot Learning (CoRL)*, 2023.

[2] F. Jenelten, J. He, F. Farshidian, and M. Hutter. DTC: Deep Tracking Control. *Science Robotics*, 2024.

[3] C. Li, M. Vlastelica, S. Blaes, J. Frey, F. Grimminger, and G. Martius. Learning agile skills via adversarial imitation of rough partial demonstrations. In *Proceedings of The 6th Conference on Robot Learning (CORL)*, 2023.

[4] X. Cheng, K. Shi, A. Agarwal, and D. Pathak. Extreme Parkour with Legged Robots. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

[5] D. Hoeller, N. Rudin, D. Sako, and M. Hutter. ANYmal parkour: Learning agile navigation for quadrupedal robots. *Science Robotics*, 2024.

[6] S. Choi, G. Ji, J. Park, H. Kim, J. Mun, J. H. Lee, and J. Hwangbo. Learning quadrupedal locomotion on deformable terrain. *Science Robotics*, 2023.

[7] C. D. Bellicoso, K. Kramer, M. Stauble, D. Sako, F. Jenelten, M. Bjelonic, and M. Hutter. ALMA - Articulated Locomotion and Manipulation for a Torque-Controllable Robot. In *2019 International Conference on Robotics and Automation (ICRA)*, 2019.

[8] S. Zimmermann, R. Poranne, and S. Coros. Go Fetch! - Dynamic Grasps using Boston Dynamics Spot with External Robotic Arm. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[9] M. Liu, Z. Chen, X. Cheng, Y. Ji, R.-Z. Qiu, R. Yang, and X. Wang. Visual Whole-Body Control for Legged Loco-Manipulation. *arXiv*, 2024.

[10] Z. Fu, X. Cheng, and D. Pathak. Deep whole-body control: Learning a unified policy for manipulation and locomotion. In *Conference on Robot Learning (CoRL)*, 2022.

[11] C. Li, E. Stanger-Jones, S. Heim, and S. Kim. Fld: Fourier latent dynamics for structured motion representation and learning. *arXiv preprint arXiv:2402.13820*, 2024.

[12] T. M. Moerland, J. Broekens, A. Plaat, and C. M. Jonker. Model-based reinforcement learning: A survey. *Foundations and Trends in Machine Learning*, 2023.

[13] J. Rothfuss, B. Sukhija, L. Treven, F. Dörfler, S. Coros, and A. Krause. Bridging the Sim-to-Real Gap with Bayesian Inference. *Proc. of the IEEE Int. Conf. on Intelligent Robots and Systems (IROS)*, 2024.

[14] X. Huang, Z. Li, Y. Xiang, Y. Ni, Y. Chi, Y. Li, L. Yang, X. B. Peng, and K. Sreenath. Creating a Dynamic Quadrupedal Robotic Goalkeeper with Reinforcement Learning. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023.

[15] Y. Ma, F. Farshidian, T. Miki, J. Lee, and M. Hutter. Combining Learning-Based Locomotion Policy With Model-Based Manipulation for Legged Mobile Manipulators. *IEEE Robotics and Automation Letters*, 2022.

[16] Y. Ji, G. B. Margolis, and P. Agrawal. DribbleBot: Dynamic Legged Manipulation in the Wild. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

[17] C. Li, A. Krause, and M. Hutter. Robotic world model: A neural network simulator for robust policy optimization in robotics. *arXiv e-prints*, pages arXiv–2501, 2025.

[18] M. Sombolestan and Q. Nguyen. Hierarchical Adaptive Loco-manipulation Control for Quadruped Robots. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

[19] S. Jeon, M. Jung, S. Choi, B. Kim, and J. Hwangbo. Learning Whole-Body Manipulation for Quadrupedal Robot. *IEEE Robotics and Automation Letters*, 2024.

[20] Y. Ji, Z. Li, Y. Sun, X. B. Peng, S. Levine, G. Berseth, and K. Sreenath. Hierarchical Reinforcement Learning for Precise Soccer Shooting Skills using a Quadrupedal Robot. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022.

[21] C. Lin, X. Liu, Y. Yang, Y. Niu, W. Yu, T. Zhang, J. Tan, B. Boots, and D. Zhao. LocoMan: Advancing Versatile Quadrupedal Dexterity with Lightweight Loco-Manipulators. *arXiv*, 2024.

[22] J. Whitman, S. Su, S. Coros, A. Ansari, and H. Choset. Generating gaits for simultaneous locomotion and manipulation. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[23] B. Forrai*, T. Miki*, D. Gehrig*, M. Hutter, and D. Scaramuzza. Event-based agile object catching with a quadrupedal robot. In *IEEE Int. Conf. Robot. Autom. (ICRA)*, 2023.

[24] H. Ferrolho, V. Ivan, W. Merkt, I. Havoutis, and S. Vijayakumar. RoLoMa: robust loco-manipulation for quadruped robots with arms. *Autonomous Robots*, 2023.

[25] A. Agarwal, A. Kumar, J. Malik, and D. Pathak. Legged locomotion in challenging terrains using egocentric vision. In *6th Annual Conference on Robot Learning (CORL)*, 2022.

[26] G. B. Margolis and P. Agrawal. Walk these ways: Tuning robot control for generalization with multiplicity of behavior. *Conference on Robot Learning*, 2022.

[27] T. Miki, J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter. Learning robust perceptive locomotion for quadrupedal robots in the wild. *Science Robotics*, 2022.

[28] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter. Learning quadrupedal locomotion over challenging terrain. *Science Robotics*, 2020.

[29] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 2019.

[30] P. Wu, A. Escontrela, D. Hafner, P. Abbeel, and K. Goldberg. Daydreamer: World models for physical robot learning. In *Conference on Robot Learning (CoRL)*, 2023.

[31] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations (ICLR)*, 2020.

[32] Z. Wang, T. Ren, J. Zhu, and B. Zhang. Function Space Particle Optimization for Bayesian Neural Networks. In *International Conference on Learning Representations (ICLR)*, 2019.

[33] S. Sun, G. Zhang, J. Shi, and R. Grosse. Functional Variational Bayesian Neural Networks. In *International Conference on Learning Representations (ICLR)*, 2019.

[34] B. Øksendal. *Stochastic Differential Equations*. Springer Berlin Heidelberg, 2003.

[35] Q. Liu and D. Wang. Stein variational gradient descent: A general purpose bayesian inference algorithm. In *Advances in Neural Information Processing Systems*, 2016.

[36] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

[37] Y. Tassa, S. Tunyasuvunakool, A. Muldal, Y. Doron, P. Trochim, S. Liu, S. Bohez, J. Merel, T. Erez, T. Lillicrap, and N. Heess. dm_control: Software and tasks for continuous control. *Software Impacts*, 2020.

[38] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

[39] A. R. Geist, J. Frey, M. Zhobro, A. Levina, and G. Martius. Learning with 3D rotations, a hitchhiker's guide to SO(3). In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

A Preliminaries on Model-Based Reinforcement Learning

In this section, we provide a brief introduction to model-based RL in the context of robot control. In an RL setting, our problem of learning control policies can be formulated as a Markov Decision Process (MDP) defined by the tuple $(\mathcal{S}, \mathcal{A}, f, r, \gamma, \mathbf{s}_0)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ the action space, $f : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the transition model or dynamics of the system, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ the reward function, $\gamma \in (0, 1)$ the discount factor, and $\mathbf{s}_0$ the initial state distribution. The goal of an RL approach is to then find an optimal policy π^* that maximizes an agent's performance in this setting. We can define the performance of a policy π over a fixed horizon H subject to the dynamics $\mathbf{s}_{t+1} \sim f(\mathbf{s}_t, \mathbf{u}_t)$ as the expected sum of discounted rewards over the horizon as

$$J(\pi, f) = \mathbb{E}_{u_t \sim \pi}\left[\sum_{t=0}^{H} \gamma^t r(\mathbf{s}_t, \mathbf{u}_t) \mid \mathbf{s}_0\right]. \tag{6}$$

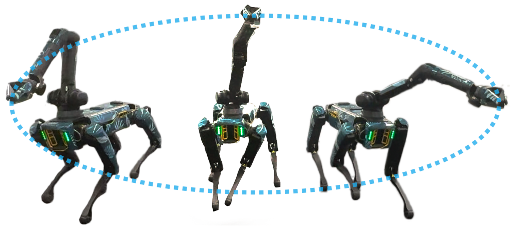
Our desired optimal policy π^* is then the result of the optimization problem

$$\pi^* = \arg\max_{\pi} J(\pi, f). \tag{7}$$

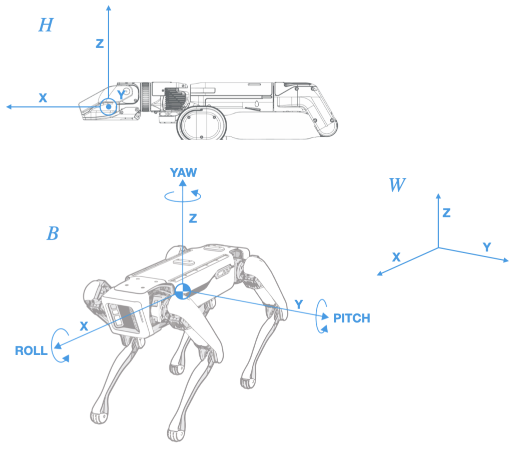
In practice, however, we often lack a model of the system dynamics f. Consequently, in *model-based* RL, we first learn an approximate model $\hat{f}$ of the dynamics from real-world data and then leverage this learned model to derive control policies that maximize our objective using an RL scheme.

B Experiment Platform and Setup

In Figure 4, we show Boston Dynamics Spot following an ellipsoidal reference trajectory in our experiments, as well as an overview of the platform's geometry and its reference frames.



(a)



(b)

Figure 4: (a) Spot following an ellipsoidal reference trajectory, and (b) the platform with the defined reference frames: the end-effector frame H, the body frame B, and the world frame W.

C Shape Tracking Experiments

In Figure 5, we provide supplementary plots that show the reference and realized trajectories for the ellipse shape tracking experiments for all polices, sample sizes and seeds.

D Model and Policy Learning Hyperparameters

We provide the hyperparameters we used during model and policy learning in Table 1.

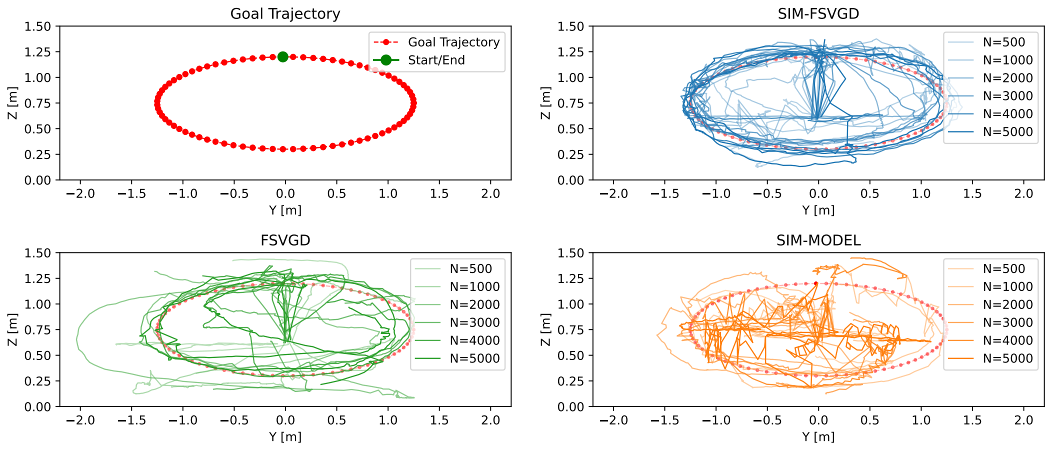


Figure 5: We plot the realized ellipse trajectories for all sample sizes and seeds. The policies learned using the SIM-FSVGD model follow the reference trajectory more closely than both baselines. Note that trajectories on course for collision were stopped early, in which case we plot only a truncated version. Also, the initial position of the end-effector and the first goal are not equivalent, resulting in the lines connecting the center to the first goal.

General BNN Hyperparameters	
Particles	5
Batch size	64
Epochs	100
Max. training steps	200’000
Learning rate	1e-3
Weight decay	1e-3
Hidden layer sizes	64, 64, 64
Hidden activation function	LeakyReLU
Learn likelihood std	Yes
Likelihood exponent	1.0
Predict state difference	Yes

FSVGD Hyperparameters	
Bandwidth SVGD	5.0
Lengthscale GP prior	0.2
Outputscale GP prior	1.0
Measurement points	16

SIM-FSVGD Hyperparameters	
Bandwidth SVGD	5.0
Lengthscale physical prior	1.0
Outputscale physical prior	0.2
Measurement points	64
Function samples	256
Score estimator	GP

SIM-MODEL Hyperparameters	
Optimizer	Adam [38]
Training steps	10’000
Learning rate	1e-3
Weight decay	1e-3

SAC Hyperparameters	
Environment steps	2’500’000
Episode length	120
Action repeat	1
Environment steps between updates	16
Environments	64
Evaluation environments	128
Learning rate α	1e-4
Learning rate policy	1e-4
Learning rate q	1e-4
Weight decay α	0.0
Weight decay policy	0.0
Weight decay q	0.0
Max. gradient norm	100
Discounting	0.99
Batch size	64
Evaluations	20
Reward scaling	1.0
τ	0.005
Min. replay size	2048
Max. replay size	50’000
Gradient updates per step	1024
Policy hidden layer	64, 64
Policy activation function	Swish
Critic hidden layer	64, 64
Critic activation function	Swish

Reward Hyperparameters	
r_{state} bound b	0.15
r_{state} margin m	1.5
r_{state} value a at margin	0.1
$r_{\text{ee-base}}$ bound l_{arm}	1.3
$r_{\text{ee-base}}$ margin m	1.3
$r_{\text{ee-base}}$ value a at margin	0.1
r_{action} base action weight λ_{base}	2.0
r_{action} end-effector action weight λ_{ee}	0.5
r_{state} weight w_1	1.5
$r_{\text{ee-base}}$ weight w_2	0.01
r_{action} weight w_3	0.1

Table 1: Hyperparameters for model and policy learning used in our experiments.