

VAMPS: Visual and Motor Policies from Sampling-Based Planning

Mohamed Yassine Kabouri^{1,2}, Pietro Noah Crestaz^{1,3}, Quang-Nam Nguyen², Qilong Cheng²,
Ludovic Righetti^{2,4}, and Nicolas Mansard^{1,4}

Abstract—Learning robot policies directly on physical systems remains difficult because data collection is costly and policy exploration can be unsafe. We introduce Visual and Motor Policies from Sampling-Based Planning (VAMPS), a framework that uses Model Predictive Path Integral (MPPI) control to train reusable policies without human demonstrations. VAMPS supports two training modes. For one-step proprioceptive policies, it operates iteratively in simulation: the policy warm-starts MPPI, and the refined trajectories provide new supervision as the policy changes. A learned terminal value improves short-horizon planning, while an Implicit Q-Learning (IQL) critic guides the policy update. Iterative refinement outperforms training once on frozen MPPI data, and we transfer the learned locomotion policy to a Unitree Go2. For visuomotor policies, VAMPS operates directly from real-robot data. MPPI uses task-specific state estimates to plan and execute trajectories while recording RGB and sensor observations on a Flexiv Rizon 10S. Action Chunking with Transformers predicts action chunks, reducing the effective prediction horizon, and is trained offline on this fixed dataset. We demonstrate visuomotor pick-and-place and force-aware whiteboard erasing. In the latter task, the policy additionally observes the measured 6-D wrench and desired normal force. These results show that VAMPS can learn policies either in simulation followed by hardware transfer or directly from autonomously collected real-robot data.

I. INTRODUCTION

Learning directly on physical robots, through interaction and sensor feedback, remains a major challenge, despite being the most direct route to acquiring effective sensorimotor behaviors. This is especially compelling for contact-rich robotics, including pick-and-place manipulation, force-regulated interaction, and quadruped locomotion. Current learning methods do not yet solve this problem at scale: they typically rely on human demonstrations or move exploration to simulation before transferring the resulting policy to hardware. This work takes a step toward methods that can autonomously and efficiently collect sensorimotor experience on robots and turn it into better policies.

This objective has been pursued through three main routes. **Imitation learning** methods such as Action Chunking with Transformers (ACT) [1] and Diffusion Policy [2] clone human teleoperation and learn rich behaviors. Their bottleneck is data: demonstrations are expensive, performance is bounded by the operator, and cloning a fixed dataset leads to compounding errors [3]. DAGger [3] corrects those states by querying an expert policy on states visited by the learned policy. **Reinforcement learning** [4] needs no demonstration, but it has to explore, and exploration is a bottleneck on contact-rich tasks with short episodes [5]. **Trajectory Optimization**

(TO) [6], [7], in contrast, optimizes individual rollouts for a given cost. While gradient-based TO relies on derivatives that are challenging to compute in contact-rich tasks, sampling-based planners [8] handle non-smooth contact dynamics and run in parallel on GPUs [9]. Yet their search space scales exponentially with the planning horizon [10]. They are commonly used in receding horizon, which requires online replanning and does not directly produce a reusable policy.

The complementarity between trajectory optimization and policy learning has long been recognized, notably by Guided Policy Search (GPS) [11] and subsequent optimizer-policy coupling methods [12], [13]. This perspective combines the precision of model-based optimization with the fast execution of a learned controller, while avoiding direct exploration by the learned policy. We build on this principle and use MPPI as a behavior generator from a model and a task cost.

In this work, we introduce Visual and Motor Policies from Sampling-Based Planning (VAMPS), a framework that distills trajectories generated by MPPI into reusable robot policies without human demonstrations or simulator gradients. MPPI plans with a model, a task cost and task-specific state estimates, while the learned policy maps the observations available at deployment directly to actions. VAMPS supports two training modes that share this planner-to-policy principle.

VAMPS-Iterative trains a one-step proprioceptive policy in simulation. Small prediction errors can move such a policy away from the states in a fixed training dataset and compound over time [3]. The policy therefore repeatedly warm-starts MPPI, which refines its behavior and provides new supervision as the policy evolves. This loop does not require simulator gradients. A learned terminal value improves short-horizon planning, while an IQL [14] critic guides the policy update through Advantage-Weighted Regression (AWR) [15]. We train the policy in simulation and transfer it to a real robot.

VAMPS-Batch uses MPPI to autonomously collect a real-robot dataset and train a visuomotor policy from it. MPPI uses task-specific state estimates to plan and execute trajectories on the robot while RGB images, robot sensors and actions are recorded. ACT [1] predicts action chunks, reducing the effective prediction horizon. For the tasks considered here, we collect the MPPI episodes once, freeze the dataset and train ACT offline. At deployment, ACT maps images and robot sensors directly to action chunks and does not use the task-specific state estimates.

Our contributions are threefold. (1) We introduce **Visual and Motor Policies from Sampling-Based Planning (VAMPS)**, a framework that uses MPPI to train reusable

¹LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France.

²Machines in Motion Laboratory, New York University, New York, USA.

³Industrial Engineering Department, University of Trento, Trento, Italy.

⁴Artificial and Natural Intelligence Toulouse Institute (ANITI), Toulouse.

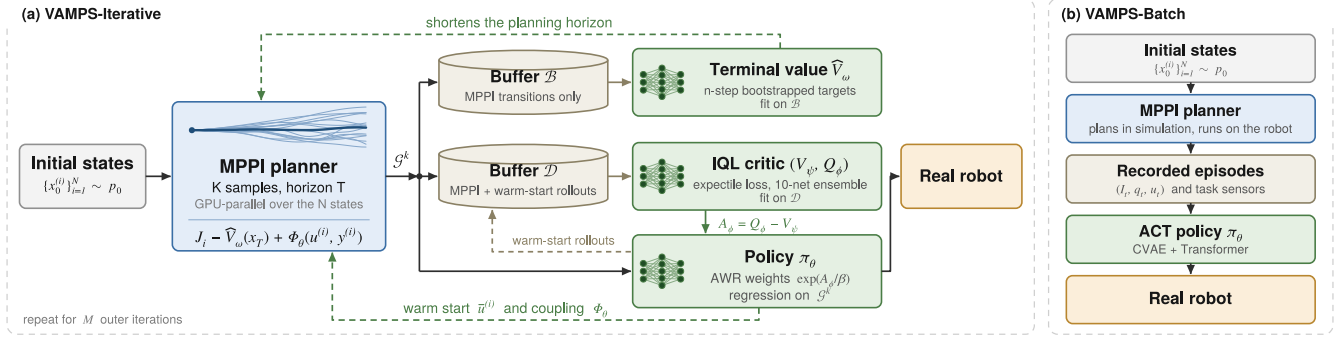


Fig. 1: Overview of the two VAMPS modes. **(a) VAMPS-Iterative:** a one-step proprioceptive policy warm-starts MPPI, which refines its behavior and provides new supervision. A terminal value improves short-horizon planning, and an IQL critic weights the policy updates. **(b) VAMPS-Batch:** MPPI uses task-specific estimates to autonomously collect RGB, sensor and action episodes on the real robot, and ACT trains offline on the resulting fixed dataset.

policies either in simulation or directly from real-robot data. (2) VAMPS-Iterative trains one-step proprioceptive policies. We show that iterative refinement outperforms training once on frozen MPPI data, a simple behavioral-cloning coupling between MPPI and the policy is sufficient, and a learned terminal value improves short-horizon planning. We transfer the resulting locomotion policy to a Unitree Go2. (3) VAMPS-Batch uses MPPI to autonomously collect a fixed real-robot dataset without human demonstrations and trains chunked ACT policies on it. We demonstrate this mode on pick-and-place and force-aware whiteboard erasing with a Flexiv Rizon 10S.

II. RELATED WORK

GPS [11] and its visuomotor extension [16] pioneered the idea of using local TO controllers to supervise policy learning. In the same cycle of ideas, [12] formalized the TO-policy coupling via ADMM to enforce agreement between them. [13] later used augmented-Lagrangian updates and sensitivity information to improve this coupling. CACTO-SL instead combines TO with actor-critic learning and uses the policy to warm-start the optimizer [17]. More recently, [18] trained diffusion policies to warm-start gradient-based TO. Their method alternates policy learning and trajectory refinement, and uses the feedback gains returned by the optimizer in the training loss. These methods rely mainly on gradient-based TO and differentiable dynamics.

PI²GPS [19] replaces the gradient-based local optimizer with the derivative-free PI² algorithm. However, it initializes the local policies from kinesthetic demonstrations and pretrains its visual features on a pose-prediction task. This work studies how to scale the GPS idea using modern tools from derivative-free optimization [20], offline RL [21] and imitation learning [1]. VAMPS-Iterative uses GPU-parallel MPPI in receding horizon. It requires a model and a task cost but does not require simulator derivatives, feedback gains or a human warm start. The policy repeatedly warm-starts MPPI, which refines its behavior and provides new supervision. To improve short-horizon planning, we introduce a learned terminal value trained on the planner transitions. To filter the planner actions, we train an IQL critic [14] on MPPI trajectories and policy rollouts, and use its advantage

in AWR [15] to give more weight to high-advantage actions.

Visuomotor imitation methods such as ACT [1] and Diffusion Policy [2] predict sequences of actions from visual observations. Action-sequence prediction reduces the effective prediction horizon and limits the accumulation of one-step errors. These methods are commonly trained from human demonstrations. VAMPS-Batch instead uses MPPI to collect real-robot RGB, sensor and action episodes, and trains ACT on the resulting fixed dataset. Task-specific state estimates guide MPPI during data collection but are not provided to the deployed policy. For force-aware erasing, MPPI records RGB images, the measured wrench and the desired normal force, and ACT learns a force-conditioned policy from these episodes. We deploy the proprioceptive policy learned by VAMPS-Iterative on a Unitree Go2 and the visuomotor policies learned by VAMPS-Batch on a Flexiv Rizon 10S.

III. PRELIMINARIES

A. Model Predictive Path Integral (MPPI)

MPPI [8] is a sampling-based trajectory optimization method. From the current state $\mathbf{x}_0$, it solves the following finite-horizon optimal control problem (OCP):

$$\begin{aligned} \min_{\mathbf{u}_{0:T-1}} \quad & J(\mathbf{x}_0, \mathbf{u}) = \sum_{t=0}^{T-1} \gamma^t c(\mathbf{x}_t, \mathbf{u}_t) + \gamma^T c_f(\mathbf{x}_T) \\ \text{s.t.} \quad & \mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t), \quad \forall t = 0, \dots, T-1, \\ & \mathbf{u}_t \in \mathcal{U}, \quad \mathbf{x}_t \in \mathcal{X}, \end{aligned} \quad (1)$$

Here, $\mathbf{u} = (\mathbf{u}_0, \dots, \mathbf{u}_{T-1})$ and γ is the discount factor. We use the cost convention $c = -r$ in the ocp, while the experiments report return, for which higher values are better.

MPPI samples K perturbations $w^{(i)} \sim \mathcal{N}(0, \Sigma_{0:T-1})$ around the nominal control sequence $\mathbf{u}$. The cost of each perturbed trajectory is $J(\mathbf{x}_0, \mathbf{u} + w^{(i)})$, where the states are obtained by rolling out the dynamics from $\mathbf{x}_0$. This cost gives the standard MPPI update:

$$\mathbf{u} \leftarrow \mathbf{u} + \frac{\sum_{i=1}^K \exp\left(-\frac{1}{\lambda} J(\mathbf{x}_0, \mathbf{u} + w^{(i)})\right) w^{(i)}}{\sum_{i=1}^K \exp\left(-\frac{1}{\lambda} J(\mathbf{x}_0, \mathbf{u} + w^{(i)})\right)}. \quad (2)$$

We run MPPI in a receding-horizon loop. A short horizon of the optimized control sequence is applied to the system,

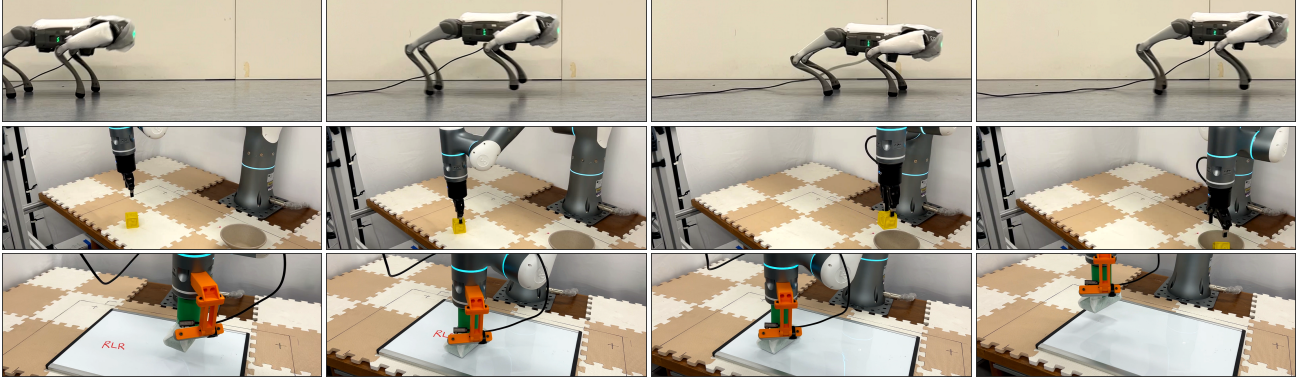


Fig. 2: The hardware that we consider, four frames each in temporal order. Top: bounding policy of the Unitree Go2 under a forward velocity command, with a proprioceptive policy trained in simulation. Middle: pick-and-place on the Flexiv Rizon 10S. Bottom: whiteboard erasing on the same arm. The two arm tasks are visuomotor and are learned directly from hardware data. Videos are provided in the supplementary video.

after which the control sequence is shifted and MPPI solves a new OCP from the resulting state.

B. Implicit Q-Learning

IQL [14] learns a state value V_ψ and an action value Q_ϕ from a dataset of transitions $\mathcal{D}$. It avoids maximizing Q_ϕ over actions outside the dataset, which can cause value overestimation [21]. Instead, it approximates the in-sample maximum with the asymmetric squared expectile loss L_2^τ :

$$\mathcal{L}_V(\psi) = \mathbb{E}_{(x,u) \sim \mathcal{D}} [L_2^\tau(Q_\phi(x, u) - V_\psi(x))], \quad (3)$$

$$\mathcal{L}_Q(\phi) = \mathbb{E}_{\mathcal{D}} [(r + \gamma V_\psi(x') - Q_\phi(x, u))^2], \quad (4)$$

where $L_2^\tau(e) = |\tau - \mathbf{1}[e < 0]|e^2$. The resulting advantage is $A_\phi(x, u) = Q_\phi(x, u) - V_\psi(x)$.

C. Action Chunking with Transformers (ACT)

ACT [1] is a visuomotor policy that predicts a short action trajectory (called a chunk) of T_u consecutive joint-position targets instead of one action. This reduces the accumulation of prediction errors.

Let $I_{t,c}$ denote the RGB image from camera c , $\mathbf{q}_t$ the robot sensor observation, and $F_{t,c} = \text{ResNet}(I_{t,c})$ the spatial image features. ACT is trained as a conditional variational autoencoder (CVAE). Its policy uses a Transformer encoder to combine $(\mathbf{q}_t, \{F_{t,c}\}, \mathbf{z})$, and a Transformer decoder predicts $\hat{\mathbf{u}}_{t:t+T_u-1}$. The latent variable $\mathbf{z}$ represents the variation between action trajectories. During training, a separate CVAE encoder infers $\mathbf{z}$ from $(\mathbf{q}_t, \mathbf{u}_{t:t+T_u-1})$. Training minimizes

$$\mathcal{L} = \mathbb{E}_{\mathbf{z} \sim q_\phi} [\|\mathbf{u}_{t:t+T_u-1} - \hat{\mathbf{u}}_{t:t+T_u-1}\|_1] + \beta_{\text{KL}} D_{\text{KL}}(q_\phi(\mathbf{z} | \mathbf{q}_t, \mathbf{u}_{t:t+T_u-1}) \| \mathcal{N}(\mathbf{0}, \mathbf{I})). \quad (5)$$

At inference the CVAE encoder is discarded and $\mathbf{z} = \mathbf{0}$, the mean of the prior.

IV. VAMPS: LEARNING POLICIES FROM SAMPLING-BASED PLANNING

VAMPS uses MPPI in two complementary modes. VAMPS-Iterative alternates between MPPI trajectory refinement and policy fitting, whereas VAMPS-Batch first collects a fixed dataset with MPPI and then trains a policy offline.

A. VAMPS-Iterative

Our goal is to learn a policy π_θ over a distribution of initial states p_0 . Rolling out π_θ from $\mathbf{x}_0$ produces a control sequence whose cost is J in Equation (1); we write $J^{\pi_\theta}(\mathbf{x}_0)$ for that cost. The policy search problem is

$$\min_{\theta} \mathbb{E}_{\mathbf{x}_0 \sim p_0} [J^{\pi_\theta}(\mathbf{x}_0)]. \quad (6)$$

In practice, we sample a finite set of initial states $\{\mathbf{x}_0^{(i)}\}_{i=1}^N \sim p_0$. Following GPS [22] and [13], we introduce a control sequence $\mathbf{u}^{(i)}$ for each sampled initial state and require these controls to be reproducible by the policy:

$$\begin{aligned} \min_{\theta, \{\mathbf{u}^{(i)}\}_{i=1}^N} \quad & \frac{1}{N} \sum_{i=1}^N J(\mathbf{x}_0^{(i)}, \mathbf{u}^{(i)}) \\ \text{s.t.} \quad & \mathbf{x}_{t+1}^{(i)} = f(\mathbf{x}_t^{(i)}, \mathbf{u}_t^{(i)}), \\ & \mathbf{u}_t^{(i)} = \pi_\theta(\mathbf{x}_t^{(i)}), \quad \forall i = 1, \dots, N, \forall t. \end{aligned} \quad (7)$$

We consider three strategies for handling this constraint in Sec. IV-A.2. For a Gaussian policy, we express the constraint as a KL divergence between the MPPI trajectory distribution and the policy distribution.

1) *Policy-Guided MPPI with a Learned Terminal Value:* VAMPS-Iterative alternates between trajectory refinement with MPPI and policy fitting. At outer iteration k , we run MPPI in receding-horizon fashion from each sampled initial state $\mathbf{x}_0^{(i)}$. At every replanning step, the current policy provides the nominal control sequence by rolling out the dynamics,

$$\bar{\mathbf{u}}_t^{(i)} = \pi_\theta(\mathbf{x}_t^{(i)}), \quad \mathbf{x}_{t+1}^{(i)} = f(\mathbf{x}_t^{(i)}, \bar{\mathbf{u}}_t^{(i)}), \quad (8)$$

and we write $\bar{\mathbf{u}}^{(i)} = (\bar{\mathbf{u}}_0^{(i)}, \dots, \bar{\mathbf{u}}_{T-1}^{(i)})$. MPPI refines this sequence, and the refined controls $\mathbf{u}^{(i)}$ supervise the policy. We extend the effective planning horizon through bootstrapping [10] by setting the terminal cost in (1) to $c_f(x_T) = -\hat{V}_\omega(x_T)$. The minus sign converts the predicted return into a cost. The value function is learned from MPPI transitions

Algorithm 1 VAMPS-Iterative

Require: initial-state distribution p_0 , number of states N , horizon T , outer iterations M , penalty ρ , coupling $\in \{\text{BC}, \text{Penalty}, \text{ADMM}\}$

- 1: Sample $\{x_0^{(i)}\}_{i=1}^N \sim p_0$ $\triangleright$ once, fixed for all iterations
- 2: Initialize policy θ , terminal value ω , critic (ψ, ϕ)
- 3: $\mathbf{y}^{(i)} \leftarrow \mathbf{0}$, $\mathcal{D} \leftarrow \emptyset$, $\mathcal{B} \leftarrow \emptyset$
- 4: **for** $k = 1$ **to** M **do**
- 5: **for** $i = 1$ **to** N **do** $\triangleright$ GPU-parallel
- 6: $(\bar{x}_t^{(i)}, \bar{u}_t^{(i)})_{t=0}^{T-1} \leftarrow \text{rollout } \pi_\theta$ $\triangleright$ warm start
- 7: $\mathbf{u}^{(i)} \leftarrow \text{MPPI}(J_i + \Phi_\theta(\mathbf{u}^{(i)}, \mathbf{y}^{(i)})) \triangleright c_f = -\hat{V}_\omega$
- 8: $\mathcal{G}^k \leftarrow \{(x_t^{(i)}, u_t^{(i)})\}_{i,t}$
- 9: $\mathcal{B} \leftarrow \mathcal{B} \cup \mathcal{G}^k$
- 10: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{G}^k \cup \{(\bar{x}_t^{(i)}, \bar{u}_t^{(i)})\}_{i,t}$
- 11: Update $\hat{V}_\omega$ on $\mathcal{B}$ $\triangleright$ (9)
- 12: Update V_ψ, Q_ϕ on $\mathcal{D}$ by IQL $\triangleright$ (3), (4)
- 13: Update θ on $\mathcal{G}^k$ $\triangleright$ (17)
- 14: **if** ADMM **then**
- 15: $y_t^{(i)} \leftarrow y_t^{(i)} + u_t^{(i)} - \mu_\theta(x_t^{(i)}), \quad \forall i, t$ $\triangleright$ (18)
- 16: **return** θ

by regression onto n -step bootstrapped returns:

$$\begin{aligned} \mathcal{L}_{\hat{V}}(\omega) &= \mathbb{E}_{(x_t, G_t) \sim \mathcal{B}} \left[(\hat{V}_\omega(x_t) - G_t)^2 \right], \\ G_t &= \sum_{j=0}^{n-1} \gamma^j r_{t+j} + \gamma^n \hat{V}_{\omega^-}(x_{t+n}), \end{aligned} \quad (9)$$

with γ the discount factor, ω^- the parameters of a target network updated by Polyak averaging, and $\mathcal{B}$ a dedicated buffer holding only (x_t, G_t) pairs from MPPI trajectories.

2) *Planner-Policy Coupling*: VAMPS-Iterative supports three coupling strategies: BC, Penalty and ADMM. They differ in how they enforce the constraint in (7). We write $J_i(\mathbf{u}) = J(x_0^{(i)}, \mathbf{u})$ and denote the control sequence returned by MPPI as $\text{MPPI}(J_i)$.

We measure the disagreement between a planner action and the policy with

$$d_\theta(x, u, y) = \begin{cases} \|u + y - \pi_\theta(x)\|^2, & \text{deterministic,} \\ D_{\text{KL}}(q_{u+y} \parallel \pi_\theta(\cdot \mid x)), & \text{Gaussian,} \end{cases} \quad (10)$$

where $q_v = \mathcal{N}(v, \sigma_{\text{MPPI}}^2 \mathbf{I})$ is the MPPI trajectory distribution for a Gaussian policy [23]. In ADMM, $y = \lambda/\rho \in \mathbb{R}^{n_u}$ is the scaled dual variable associated with the planner-policy consensus constraint, where λ is the corresponding Lagrange multiplier and $\rho > 0$ is the penalty parameter. We define the trajectory-level coupling penalty as

$$\Phi_\theta(\mathbf{u}^{(i)}, \mathbf{y}^{(i)}) = \frac{\rho}{2} \sum_t d_\theta(x_t^{(i)}, u_t^{(i)}, y_t^{(i)}). \quad (11)$$

Here, $\rho > 0$ is the coupling coefficient.

BC coupling. The planner is uncoupled from the current policy and runs MPPI on the task cost alone:

$$\mathbf{u}^{(i),k+1} \leftarrow \text{MPPI}(J_i). \quad (12)$$

Algorithm 2 VAMPS-Batch

Require: task cost J , initial-state distribution p_0 , episodes E

- 1: $\mathcal{D}_{\text{batch}} \leftarrow \emptyset$
- 2: **for** $i = 1$ **to** E **do**
- 3: Sample $x_0^{(i)} \sim p_0$
 # Replan online and record data
- 4: $\tau_i \leftarrow \text{ExecuteOnRobot}(\text{MPPI}(J(x_0^{(i)}, \mathbf{u})))$
- 5: $\mathcal{D}_{\text{batch}} \leftarrow \mathcal{D}_{\text{batch}} \cup \{\tau_i\}$
- 6: Freeze $\mathcal{D}_{\text{batch}}$ and train ACT with (5)
- 7: **return** θ

Penalty coupling. The coupling Φ_θ (with $\mathbf{y}^{(i)} = \mathbf{0}$) is added to the MPPI cost:

$$\mathbf{u}^{(i),k+1} \leftarrow \text{MPPI}(J_i + \Phi_\theta(\mathbf{u}^{(i)}, \mathbf{0})). \quad (13)$$

The penalty biases MPPI toward trajectories the current policy can represent.

ADMM coupling. The constraint of (7) is the consensus $\mathbf{u}^{(i)} = \bar{\mathbf{u}}^{(i)}$. Using the scaled dual variables $\mathbf{y}^{(i)}$, the augmented Lagrangian is:

$$\mathcal{L}_\rho = \sum_{i=1}^N J_i + \sum_{i=1}^N \Phi_\theta(\mathbf{u}^{(i)}, \mathbf{y}^{(i)}). \quad (14)$$

Its trajectory update is

$$\mathbf{u}^{(i),k+1} \leftarrow \text{MPPI}(J_i + \Phi_\theta(\mathbf{u}^{(i)}, \mathbf{y}^{(i),k})). \quad (15)$$

3) *Critic-Guided Policy Learning*: At each iteration, we add the refined MPPI trajectories and the policy rollouts used to warm-start MPPI to a replay buffer $\mathcal{D}$. We use both sources for two reasons. First, the warm-start rollouts provide transitions from the current policy, reducing extrapolation when we evaluate $Q_\phi(x, \pi_\theta(x))$. Second, combining them with the refined MPPI trajectories improves the action coverage used to estimate the advantages.

We train V_ψ and Q_ϕ with the IQL losses in (3)-(4). The buffer $\mathcal{D}$ is used only for these critic updates. AWR [15] converts the advantage $A_\phi(x, u) = Q_\phi(x, u) - V_\psi(x)$ into the normalized weight

$$w_\beta(x, u) = \frac{\exp(A_\phi(x, u)/\beta)}{\mathbb{E}_{(x', u') \sim \mathcal{G}^k} [\exp(A_\phi(x', u')/\beta)]}, \quad (16)$$

where $\beta > 0$ is the temperature. All three coupling strategies use these weights to update the policy on the current MPPI trajectories $\mathcal{G}^k = \{(x_t^{(i)}, u_t^{(i),k+1})\}_{i,t}$:

$$\theta^{k+1} = \arg \min_{\theta} \sum_{(x, u) \in \mathcal{G}^k} w_\beta(x, u) d_\theta(x, u, y). \quad (17)$$

BC and Penalty set $y = 0$ in this update. ADMM uses its current scaled dual variable y^k . After the policy update, ADMM updates this variable at each time step:

$$y_t^{(i),k+1} \leftarrow y_t^{(i),k} + u_t^{(i),k+1} - \mu_{\theta^{k+1}}(x_t^{(i)}), \quad (18)$$

where $\mu_\theta(x) = \mathbb{E}_{u \sim \pi_\theta(\cdot \mid x)}[u]$ for a Gaussian policy and $\mu_\theta(x) = \pi_\theta(x)$ for a deterministic policy. The dual variable accumulates the planner-policy residual.

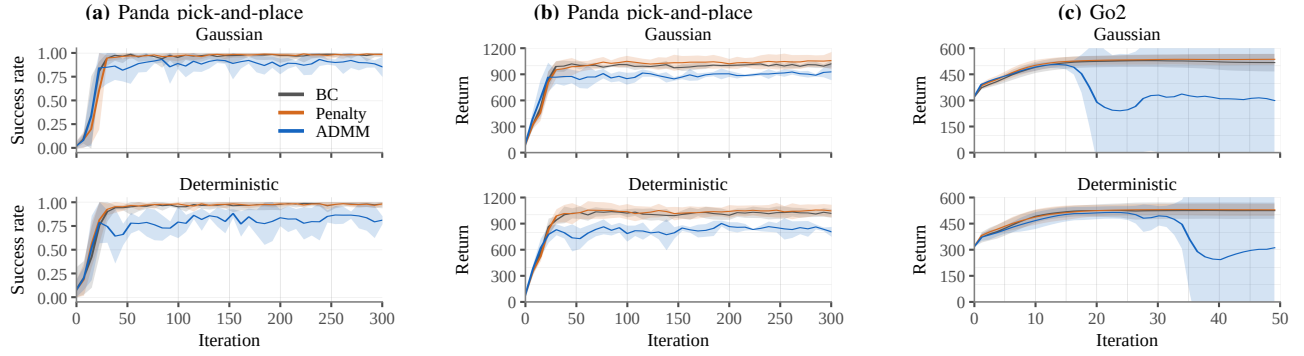


Fig. 3: The three VAMPS-Iterative coupling strategies. Each task shows a Gaussian policy and a deterministic policy. Shaded bands: 95 % CI over 4 seeds.

B. VAMPS-Batch

VAMPS-Iterative uses a one-step proprioceptive policy. Small prediction errors can move this policy away from the states in a fixed training dataset and compound over time [3]. Our frozen-data baseline show this effect in Sec. V-F. VAMPS-Iterative addresses it by repeatedly refining the policy trajectories with MPPI and using them as new supervision. VAMPS-Batch instead uses ACT [1], which predicts an action trajectory rather than one action at each step. This reduces the effective decision horizon and motivates training the visuomotor policy once on a fixed dataset.

VAMPS-Batch uses MPPI to autonomously collect this dataset on the real robot. MPPI evaluates candidate rollouts in simulation using task-specific state estimates and executes the refined controls on the robot. For example, it uses the cube position for pick-and-place and the contact height z for erasing. During execution, we record synchronized RGB images, robot observations, task sensors and actions. We freeze the resulting episodes and train ACT offline with (5). At deployment, ACT does not use the task-specific state estimates. Its evaluation is reported in Sec. VI.

C. Algorithm Summary

Algorithms 1 and 2 summarize the two VAMPS modes. Both use MPPI to generate supervision from a model and a task cost. VAMPS-Iterative updates the planner, policy and value functions together: the current policy warm-starts MPPI, and each refined trajectory updates the policy before the next outer iteration. VAMPS-Batch instead completes data collection first, freezes the recorded real-robot episodes and then trains ACT offline.

V. VAMPS-ITERATIVE EXPERIMENTS & RESULTS

A. Experimental Setup

VAMPS tasks. In Panda pick-and-place from MuJoCo Playground [24], a Franka Panda grasps a cube from randomized positions and lifts it to a target. We use this task for the coupling comparison, component ablations and standard baselines. Go2 evaluates the coupling on locomotion and real-robot policy transfer.

VAMPS-Iterative architecture. The policy π_θ , critic (V_ψ, Q_ϕ) and terminal value $\hat{V}_\omega$ use BroNet [25]: a Dense $\rightarrow$ LayerNorm $\rightarrow$ ReLU stem and residual blocks of width 512. The critic uses two blocks, while π_θ and $\hat{V}_\omega$ use one.

Q_ϕ is an ensemble of 10 networks [26], whose mean defines the advantage. The policy output layer uses orthogonal initialization with scale 0.01. We train all networks with AdamW [27].

VAMPS-Iterative training. At each outer iteration, the critic and terminal value take 15 gradient steps with batch sizes 1024 and 256, respectively; the terminal value uses $n=2$ -step targets. We use IQL expectile $\tau=0.7$, discount $\gamma=0.99$, and target update rate $\tau_{\text{net}}=0.005$. The policy takes 10 epochs on Panda and 15 on Go2, with learning rate 5×10^{-4} and AWR temperature $\beta=2.0$. MPPI uses horizon $T=16$, DIAL-MPC [9] annealing, and replans every two steps. Panda uses $N=80$, $K=512$, $N_{\text{diffuse}}=3$, and 300 outer iterations. Go2 uses $N=50$, $K=1024$, $N_{\text{diffuse}}=2$, 50 outer iterations, and domain randomization.

VAMPS-Iterative metrics. The primary metric is the mean return evaluated on a test set of 50 initial states per seed; shaded bands indicate 95 % confidence intervals across seeds.

Default configuration. Unless stated otherwise, the experiments use VAMPS-Iterative (Algorithm 1) with a Gaussian policy.

B. Q1: How does the coupling strategy between the trajectory optimizer and the policy affect performance?

Setup. We run the full method with each of the three coupling strategies of Sec. IV-A.2, and we do so twice: once with a Gaussian policy and once with a deterministic one. This gives six conditions per task, and we run each of them with four seeds. We denote the configurations VAMPS-Iterative (BC), VAMPS-Iterative (Penalty) and VAMPS-Iterative (ADMM). BC uses $\rho = 0$, Penalty uses $\rho = 0.005$ with the dual variables frozen, and ADMM uses $\rho = 0.005$ with the dual variables active.

Result. On the Panda task, BC and Penalty pass a success rate of 0.90 within about 30 iterations and then hold a plateau just under 1.0, with both the Gaussian and the deterministic policy (Fig. 3). ADMM climbs as fast at first but settles lower, at 0.88 with the Gaussian policy and 0.90 with the deterministic one. The return shows the same: BC and Penalty plateau between 1010 and 1060, with Penalty above BC by about 40 with both policies. ADMM plateaus between 860 and 875. The Go2 task gives the same result. BC and Penalty again behave identically. ADMM is again the worst,

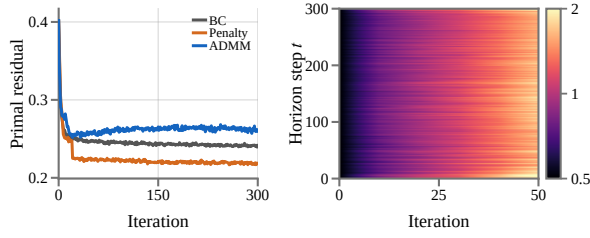


Fig. 4: **Left:** median primal residual over four seeds: BC already follows the planner to 0.24 with no consensus enforced. **Right:** 95th percentile of $|y_{t,j}^{(i)}|$ over states and action coordinates, on one Go2 seed. Colorbar in action units; 1 is the bound of π_θ .

and its instability is more visible here. The Go2 curves start from a high return because of the orthogonal action head initialized at a scale of 0.01: the untrained network outputs a near-zero action, which under position control is the default joint configuration.

Analysis. We attribute the instability of the ADMM variant to the dual variables $\mathbf{y}$. In the policy step (17), they shift the regression target from $\mathbf{u}$ to $\mathbf{u} + \mathbf{y}$. The right panel of Fig. 4 shows, at each outer iteration and horizon step t , the 95th percentile of $|y_{t,j}^{(i)}|$ over the N initial states and the action coordinates. A cell past 1 means at least 5% of those entries put $\mathbf{u} + \mathbf{y}$ outside $[-1, 1]$, the range of π_θ .

With $\mathbf{y}$ set to zero, Penalty at $\rho=0.005$ brings no gain over BC, and it brings no loss either. This is consistent with Sec. V-E, where performance degrades as ρ grows. We hypothesize the size of the policy explains this, and the left panel of Fig. 4 supports it. The coupling term exists to keep the planner inside the set of trajectories that the policy is able to reproduce, which is a real constraint when the policy is represented by a low-capacity neural network, as in GPS [11]. Our policy reproduces the MPPI controls with no coupling. The primal residual falls to 0.24 within 30 iterations and stays there, and the penalty only brings it to 0.21. The consensus constraint is therefore almost satisfied. The dual updates move it in the opposite direction, to 0.26. The displaced target of Fig. 4 pulls the planner and the policy apart rather than together, which is the behavior the method is meant to prevent. We keep Penalty as the proposed VAMPS-Iterative coupling and use it in the remaining experiments.

C. Q2: Does the learned terminal value function compensate for a shorter horizon?

Setup. We evaluate the terminal value $\hat{V}_\omega$ trained at each iteration on the MPPI transitions accumulated in $\mathcal{B}$. It accounts for the part of the task that lies beyond the MPPI horizon, so we vary the horizon as well: $T = 16$, $T = 8$ and $T = 4$, each with and without $\hat{V}_\omega$, for four seeds each.

Result. As expected, the terminal value improves the final policy return at every horizon, and the gain grows as the horizon shrinks (Fig. 5, left). It is large at $T=4$, still clear at $T=8$, and small at $T=16$. The terminal value is not a replacement for a long horizon, because $T=4$ with it stays well below $T=8$ without it. The figure therefore shows that a reasonable horizon is important, and that the terminal

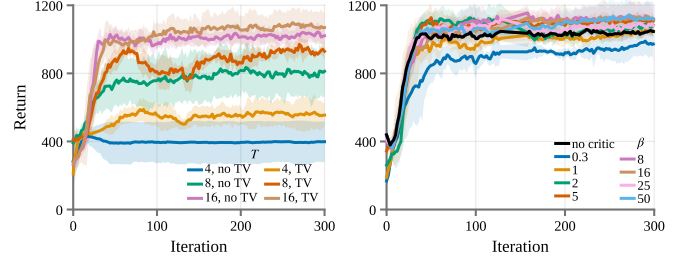


Fig. 5: Test return on Panda pick-and-place. **Left:** planning horizons $T \in \{16, 8, 4\}$ with and without the learned terminal value (TV). **Right:** VAMPS-Iterative without a critic and with the critic for different AWR temperatures β ; both retain the terminal value. Shaded bands: 95% CI across seeds.

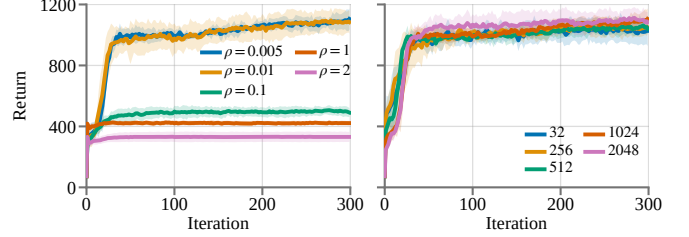


Fig. 6: Sensitivity to the hyperparameters. **Left:** penalty coefficient ρ , with the batch size fixed at 1024. **Right:** batch size, with ρ fixed at 0.005. Shaded bands: 95% CI over 4 seeds.

value lets us reduce the computational budget of MPPI by shortening the rollout.

D. Q3: Does the offline RL critic improve performance?

Setup. We compare VAMPS-Iterative with a no-critic ablation, which sets the weights in (17) to one and therefore fits the policy by uniform behavioral cloning on Panda pick-and-place. Both use a Gaussian policy and retain the learned terminal value. We use four seeds per condition. We sweep the AWR temperature β over the values in Fig. 5.

Result. VAMPS improves slightly over VAMPS-no-critic for the tested temperatures from $\beta=2$ to $\beta=50$. At $\beta=1$ the final return is similar to VAMPS-no-critic, while $\beta=0.3$ gives a lower return. The offline RL critic therefore brings a small gain on Panda pick-and-place for a suitable temperature.

E. Q4: How sensitive is performance to the penalty coefficient and batch size?

Setup. We vary the penalty coefficient ρ and the batch size, one at a time, over the values shown in the legend of Fig. 6. All other hyperparameters keep their values from Sec. V-A. All runs use a Gaussian policy.

Result. The batch size has no effect, since the five curves overlap over the whole run (Fig. 6). The penalty coefficient behaves as a threshold rather than as a gradual trade-off. Values of $\rho = 0.005$ and $\rho = 0.01$ give the same return, while $\rho \geq 0.1$ drops to a much lower level within the first iterations and stays there. The policy is untrained at the beginning, so a large ρ makes the penalty dominate the MPPI cost.

F. Q5: How do standard baselines perform in this regime?

Setup. We compare VAMPS-Iterative against PPO [28], frozen BC and frozen IQL on the Panda pick-and-place task,

TABLE I: Baselines on Panda: mean $\pm$ 95% CI over four seeds at each method’s best checkpoint. PPO is trained at the listed episode length and evaluated over 80 steps, as VAMPS-Iterative.

Method	Episode length	Return	Success
PPO	80	279 $\pm$ 84	0.00
PPO	500	975 $\pm$ 10	0.98 $\pm$ 0.03
PPO	1000	894 $\pm$ 163	0.74 $\pm$ 0.49
BC (frozen)	–	290 $\pm$ 85	0.02 $\pm$ 0.03
IQL (frozen)	–	240 $\pm$ 40	0.01 $\pm$ 0.01
VAMPS-Iterative	80	1109 $\pm$ 76	0.98 $\pm$ 0.02

with four seeds and 50 test initial states. PPO learns from the same reward. We train it at three episode lengths: 80, which matches VAMPS-Iterative, and 500 and 1000. Evaluation is always 80 steps. BC and IQL learn offline from one pass of MPPI transitions. We give PPO the same simulator budget as VAMPS-Iterative, 1.6×10^{10} steps. Of that budget VAMPS-Iterative uses only 1.3×10^6 policy–environment interaction steps, in episodes of length 80, the rest are planner rollouts, and those policy steps go into the critic’s replay buffer. Every PPO step is a policy–environment interaction step.

Result. Table I shows that PPO trained at episode length 80 stays at 279 return and 0.00 success where the gripper moves toward the cube and never picks it up. Increasing the training episode to 500 raises the mean to 975 / 0.98; at 1000 the mean is 894 / 0.74, pulled down by one seed that also never grasps, and stays below VAMPS-Iterative. Frozen BC and IQL are trained on one pass of successful MPPI trajectories, yet they stay with PPO at episode length 80 (290 / 0.02 and 240 / 0.01). VAMPS-Iterative, at episode length 80, reaches 1109 / 0.98. It passes a return of 1000 after 3.6×10^9 simulator steps, of which only 2.9×10^5 are policy–environment interaction steps used to warm-start the planner.

The key difference is exploration. At 50 Hz, VAMPS-Iterative uses 80-step (1.6 s) episodes, while PPO needs 500-step (10 s) episodes of an untrained policy to solve the task, which is unsafe on hardware. VAMPS-Iterative instead explores with MPPI on a model. At every iteration, the current policy warm-starts MPPI, which refines its actions and returns new trajectories to update the policy. This feedback loop focuses planning around the current policy and provides new supervision as the policy changes. Frozen BC compounds errors outside the MPPI states [3], while frozen IQL has no coverage beyond its single pass of 6,320 transitions [14], [21].

G. Q6: Does the policy transfer to real hardware?

We deploy the proprioceptive policy trained in simulation on a physical Unitree Go2 robot. Fig. 2 shows four frames of one bounding cycle recorded during such a run, where the robot moves under the commanded velocity from the joystick. We used domain randomization [29]. Each of the N initial states carries its own randomized model, drawn once before training and then held fixed for the whole run. We randomize the friction of the floor, the mass of every link, a payload of up to one kilogram on the torso, the position of

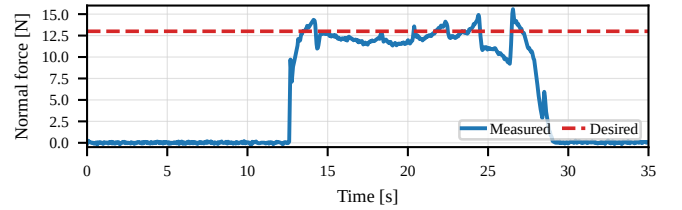


Fig. 7: Desired and measured normal force during one successful VAMPS-Batch erasing trial shown in Fig. 2.

the torso center of mass, the joint friction and armature, and the PD gains of the position actuators. MPPI plans in the perturbed model of its own scenario, so the policy is fitted on solutions that already account for the model mismatch.

VI. VAMPS-BATCH EXPERIMENTS & RESULTS

Both VAMPS-Batch experiments run on a 7-DOF Flexiv Rizon 10S with a gripper and a side-mounted Orbbec Femto Bolt camera. MPPI evaluates 2048 sampled rollouts in simulation over a horizon $T=20$. It replans in receding horizon and executes the refined controls on the robot at 25 Hz through an impedance controller. We record RGB images, sensor observations and actions, and train ACT on these real-robot episodes using the HuggingFace LeRobot implementation [30], [1]. At inference, the full action trajectory is executed open-loop before ACT is queried again.

A. Experiment 1: Visuomotor pick-and-place

Setup. The robot grasps a cube from a randomized position on the table and drops it in a fixed bowl. We localize the cube with HSV color filtering and a calibrated homography. This estimate is used only by MPPI during data collection. We autonomously collect 100 MPPI episodes. The policy predicts a chunk of $T_u=100$ joint-position targets.

Result. The VAMPS-Batch policy succeeds in 8 of 10 trials from randomized cube positions. The middle row of Fig. 2 shows four frames of one successful execution.

B. Experiment 2: Visuomotor and force-aware erasing policy

Setup. The robot erases a whiteboard placed on the table. The eraser has to stay in contact with the board while the arm moves over it, so MPPI needs the height at which to hold the eraser: too high and it loses contact and does not erase, too low and it presses through the board. During data collection, we obtain this height with the task-space impedance-admittance nonlinear MPC of [31]. It regulates the contact and gives the z position MPPI plans around. This contact-height estimate is used only during data collection. We collect demonstrations at discrete desired normal-force setpoints from 5 to 14 N to cover different contact conditions. The resulting dataset contains 84 episodes (1.59 hours on the robot). For this task, ACT also takes a second camera mounted on the wrist. Its observation at time t contains the two synchronized images and a 14-D vector with the seven joint positions, the measured 6-D wrench and the desired normal force. ACT reads the wrench at t without averaging a force window. The policy predicts a chunk of $T_u=50$ joint-position targets (2 s at 25 Hz), executed open-loop as in pick-and-place.

Result. The same VAMPS-Batch checkpoint succeeds in four consecutive trials and erases the writing in all of them. Fig. 7 shows the desired and measured normal force during one successful trial. The bottom row of Fig. 2 shows four frames of the same execution. These preliminary results show that VAMPS-Batch can train visuomotor policies from autonomously collected real-robot data without human demonstrations. We plan to extend this mode to more sensor-based tasks in future work.

VII. CONCLUSIONS

We presented VAMPS, a framework that uses a sampling-based planner to train reusable robot policies without human demonstrations or a differentiable simulator. We studied two modes of this framework. VAMPS-Iterative alternates MPPI trajectory refinement and policy learning for one-step proprioceptive policies. Our experiments show that iterative refinement outperforms training once on frozen MPPI data, simple planner-policy coupling is sufficient, and a learned terminal value improves short-horizon planning. We transfer the resulting locomotion policy to a Unitree Go2. VAMPS-Batch uses MPPI to autonomously collect a fixed real-robot dataset and trains chunked visuomotor policies offline with ACT. The dataset contains RGB images, robot observations, task sensors and actions. We demonstrate this mode on pick-and-place and force-aware whiteboard erasing with a Flexiv Rizon 10S.

The VAMPS-Batch hardware results are preliminary, and we plan to extend it to more sensor-based tasks. For long-horizon tasks, TAMP [32] could provide a high-level planner. Scene generators such as SimFoundry [33] could build simulations from real footage, allowing VAMPS to collect data and learn visuomotor policies in a real-to-sim-to-real loop. Future work will also study tighter planner-policy feedback in architectures such as TD-MPC2 [34] and CAST [35].

REFERENCES

- [1] T. Z. Zhao, V. Kumar *et al.*, “Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware,” in *Proceedings of Robotics: Science and Systems*, 2023.
- [2] C. Chi, Z. Xu *et al.*, “Diffusion policy: Visuomotor policy learning via action diffusion,” *The International Journal of Robotics Research (IJRR)*, 2024.
- [3] S. Ross, G. Gordon *et al.*, “A reduction of imitation learning and structured prediction to no-regret online learning,” in *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, vol. 15. PMLR, 2011.
- [4] J. Hwangbo, J. Lee *et al.*, “Learning agile and dynamic motor skills for legged robots,” *Science Robotics*, vol. 4, 2019.
- [5] X. Mao, Y. XU *et al.*, “Learning long-horizon robot manipulation skills via privileged action,” in *Conference on Robot Learning (CoRL)*, 2025.
- [6] H. Zhao, L. Righetti *et al.*, “Hippo: High-performance interior-point and projection-based solver for generic constrained trajectory optimization,” *IEEE Robotics and Automation Letters*, vol. 11, 2026.
- [7] A. Jordana, S. Kleff *et al.*, “Structure-exploiting sequential quadratic programming for model-predictive control,” *IEEE Transactions on Robotics*, vol. 41, 2025.
- [8] G. Williams, P. Drews *et al.*, “Information-theoretic model predictive control: Theory and applications to autonomous driving,” *IEEE Transactions on Robotics*, vol. 34, pp. 1603–1622, 2017.
- [9] H. Xue, C. Pan *et al.*, “Full-order sampling-based mpc for torque-level locomotion control via diffusion-style annealing,” *IEEE International Conference on Robotics and Automation (ICRA)*, 2024.
- [10] P. N. Crestaz, L. De Matteis *et al.*, “TD-CD-MPPI: Temporal-difference constraint-discounted model predictive path integral control,” *IEEE Robotics and Automation Letters*, vol. 11, no. 1, 2026.
- [11] S. Levine and V. Koltun, “Guided policy search,” in *Proceedings of the International Conference on Machine Learning (ICML)*, ser. Proceedings of Machine Learning Research, vol. 28. PMLR, 2013.
- [12] I. Mordatch and E. Todorov, “Combining the benefits of function approximation and trajectory optimization,” in *Robotics: Science and Systems*, 2014.
- [13] Q. L. Lidec, W. Jallet *et al.*, “Enforcing the consensus between trajectory optimization and policy learning for precise robot control,” in *IEEE International Conference on Robotics and Automation (ICRA)*, 2023.
- [14] I. Kostrikov, A. Nair *et al.*, “Offline reinforcement learning with implicit q-learning,” in *International Conference on Learning Representations (ICLR)*, 2022.
- [15] X. B. Peng, A. Kumar *et al.*, “Advantage-weighted regression: Simple and scalable off-policy reinforcement learning,” arXiv:1910.00177, 2019.
- [16] S. Levine, C. Finn *et al.*, “End-to-end training of deep visuomotor policies,” *Journal of Machine Learning Research (JMLR)*, vol. 17, no. 1, 2016.
- [17] E. Alboni, G. Grandesso *et al.*, “CACTO-SL: Using Sobolev learning to improve continuous actor-critic with trajectory optimization,” in *Proceedings of the Learning for Dynamics & Control Conference (LADC)*, ser. Proceedings of Machine Learning Research, vol. 242. PMLR, 2024.
- [18] T. L. Hellard, F. N. Tiofack *et al.*, “Accelerating trajectory optimization with sobolev-trained diffusion policies,” 2026.
- [19] Y. Chebotar, M. Kalakrishnan *et al.*, “Path integral guided policy search,” *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.
- [20] A. Jordana, J. Zhang *et al.*, “An introduction to zero-order optimization techniques for robotics,” arXiv:2506.22087, 2025.
- [21] S. Levine, A. Kumar *et al.*, “Offline reinforcement learning: Tutorial, review, and perspectives on open problems,” arXiv:2005.01643, 2020.
- [22] W. H. Montgomery and S. Levine, “Guided policy search via approximate mirror descent,” in *Advances in Neural Information Processing Systems*, vol. 29, 2016.
- [23] T. M. Cover and J. A. Thomas, *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, 2006.
- [24] K. Zakka, B. Tabanpour *et al.*, “Mujoco playground,” 2025.
- [25] M. Nauman, M. Ostaszewski *et al.*, “Bigger, regularized, optimistic: scaling for compute and sample efficient continuous control,” in *Conference on Neural Information Processing Systems*, 2024.
- [26] W.-D. Chang, M. Henaff *et al.*, “The surprising difficulty of search in model-based reinforcement learning,” arXiv:2601.21306, 2026.
- [27] I. Loshchilov and F. Hutter, “Decoupled weight decay regularization,” in *International Conference on Learning Representations*, 2019.
- [28] J. Schulman, F. Wolski *et al.*, “Proximal policy optimization algorithms,” arXiv:1707.06347, 2017.
- [29] J. Tobin, R. Fong *et al.*, “Domain randomization for transferring deep neural networks from simulation to the real world,” *IEEE International Conference on Intelligent Robots and Systems*, 2017.
- [30] R. Cadene, S. Alibert *et al.*, “Lerobot: State-of-the-art machine learning for real-world robotics in pytorch,” <https://github.com/huggingface/lerobot>, 2024.
- [31] “Task-space motion-force nonlinear model predictive control for precise and compliant control under uncertainties,” 2026, under review.
- [32] C. R. Garrett, R. Chitnis *et al.*, “Integrated task and motion planning,” arXiv:2010.01083, 2020.
- [33] N. Ranawaka, J. Wong *et al.*, “SimFoundry: Modular and automated scene generation for policy learning and evaluation,” arXiv:2606.28276, 2026.
- [34] N. Hansen, H. Su *et al.*, “TD-MPC2: Scalable, robust world models for continuous control,” arXiv:2310.16828, 2024.
- [35] P. N. Crestaz, M. Y. Kabouri *et al.*, “CAST: Alternating state-value targets and expanded policy gradients for model-based reinforcement learning,” arXiv:2609.08853, 2026.