Koopman-Accelerated Model-Based Diffusion for Real-Time Robot Control

***Abstract*— Conventional model-based diffusion (MBD) achieves effective trajectory optimization by leveraging noise annealing. However, its high computational cost, primarily arising from repeated rollouts of the plant dynamics, has largely confined its use to offline settings. To address this limitation, this paper proposes bilinear Koopman model-based diffusion (BK-MBD). The proposed method lifts the robot's state into a high-dimensional space only once per control step and propagates all candidates in the lifted space thereafter, so each rollout reduces to a fixed number of matrix–vector multiplications. The lifted dynamics are bilinear, allowing the predicted input gain to vary with the robot's configuration, which a linear lifted model cannot represent. In simulation, BK-MBD completed each planning update in at most 14.7 ms within a 50 ms control period and reached the goal on every trial, whereas a linear lift almost never did. The annealed schedule improves closed-loop accuracy over fixed-noise schedules under the learned rollout. Under the exact rollout, both the annealed and fixed-narrow schedules reach every goal, indicating that annealing reduces sensitivity to surrogate-model error. BK-MBD also threaded a passage that no single convex region covers, whereas a convexified bilinear controller rarely succeeded. On a physical manipulator, BK-MBD tracked an initially unknown moving target within the control period and was the only method that met both the tracking task and the deadline. The project page is available at `https://anonymous.4open.science/w/bkmbd2026/`.**

I. Introduction

Sampling-based predictive control is widely used for robotic systems with nonlinear dynamics. Model predictive path integral (MPPI) [1], [2] and model-based diffusion (MBD) [3] evaluate candidate control sequences via forward rollout and update the nominal sequence using a cost-weighted average. MBD treats each such update as a step of reverse diffusion. The weighted average estimates the score of a Gaussian-smoothed target [4], [5], while a decreasing noise schedule guides the search from coarse exploration to fine refinement. This approach requires neither a differentiable nor a convex cost function and can accommodate nonconvex feasible sets.

However, MBD remains limited to offline trajectory optimization, leaving its application to receding-horizon control as an open challenge [3]. The primary bottleneck is the computational cost of the rollout. A single control step evaluates candidate rollouts at every annealing stage, and receding-horizon control draws a new population after each executed action, so a simulation-based rollout is evaluated far too often to be affordable. Xue et al. [6] mitigated this cost with a vectorized physics engine on a GPU. On a CPU, meeting the control rate still calls for a rollout model far lighter than a full simulation.

A Koopman lift enables such low-cost computation by acting as a surrogate for the plant. Lifting the state into a high-dimensional space through hand-designed [7], [8] or learned [9] observables turns the nonlinear rollout into simple matrix–vector multiplications. Hao et al. [10] applied this by integrating a linear Koopman model into MPPI. Placing a finite-dimensional learned model inside a sampling planner, however, raises two challenges. The rollout must preserve the configuration-dependent effect of the input, and the optimizer must be robust to local minima that residual model error may introduce into the surrogate cost landscape.

The first challenge originates in how the input is represented. A linear lift processes the input through a constant matrix, which amounts to assuming an input gain that does not depend on the configuration of the robot. On a velocity-commanded robot, the input gain instead varies continuously with the configuration. This mismatch is particularly detrimental to sampling-based planning, where candidates are selected according to their predicted cost differences. Holding the gain constant misrepresents how different control sequences affect the robot across configurations and can therefore corrupt the ranking of candidates. Increasing the number of candidates does not remove this error, since it is structural rather than statistical. A bilinear Koopman model allows the predicted input gain to depend on the configuration through state–input product terms [11]. Linear lifted models have nonetheless remained attractive because they allow the control problem to be cast as a quadratic program (QP) [12], and Bruder et al. [13] retained bilinear dynamics but froze the state-dependent terms along a nominal trajectory to preserve convexity, yielding a local approximation of the full state–input coupling. Thus, the practical obstacle lies not in the bilinear representation itself, but in optimizing it using conventional convex methods.

The second challenge originates in model error. A learned rollout model is approximate, and its residual error may introduce local minima into the surrogate cost landscape. An optimizer that settles into one of these minima selects a sequence that looks favorable under the surrogate yet performs poorly on the real system. Raising the expressiveness of the rollout model is thus not sufficient on its own, and the optimizer has to be robust to such error as well.

This paper accordingly proposes bilinear Koopman model-based diffusion (BK-MBD), which combines a bilinear Koopman rollout with the annealed sampling of MBD. The bilinear structure carries the configuration-dependent input gain, and the sampling planner evaluates only the total cost of a sequence without requiring convexity, so it uses that state dependence as it stands rather than freezing or convexifying it. The annealed schedule smooths the cost-induced target broadly at high noise levels and refines the plan under a less-smoothed target as the noise decreases, which can reduce sensitivity to these surrogate-induced local minima. To test whether this benefit originates in surrogate error, fixed-noise and annealed schedules are compared under both learned and

exact rollouts. Computationally, the state is lifted once per control step and every candidate is then propagated in the lifted space, so each rollout reduces to a fixed number of matrix–vector multiplications.

The diffusion in BK-MBD is a property of the optimizer rather than a learned motion prior. Existing diffusion-based motion planners learn task-conditioned trajectory distributions from task-specific data [14]–[16], whereas BK-MBD learns only the rollout model; goals and constraints enter at deployment through the cost function.

The main contributions are as follows.

- **MBD at the control rate.** BK-MBD employs a bilinear Koopman rollout that represents the configuration-dependent input gain without freezing or convexification. Lifting the state once per control step reduces all subsequent candidate propagation to a fixed sequence of matrix–vector multiplications, allowing the full annealing process to run within the control period.
- **Annealing under a learned rollout.** Under the learned surrogate, annealing attains higher reaching accuracy than every fixed-noise schedule. Under the exact rollout, both the annealed and fixed-narrow schedules reach every goal, indicating that annealing reduces sensitivity to surrogate-model error.
- **Real-time validation on a physical robot.** On a physical FR3 tracking a moving target whose trajectory is unavailable in advance, BK-MBD completes all ten trials and returns every command within the 50 ms control period, whereas every baseline fails the task, the deadline, or both.

II. Preliminaries

In this section we formulate the control problem, review the MBD update, and state the two classes of lifted rollout that the study compares.

A. Problem Formulation

Consider the discrete-time system

$$s_{t+1} = f(s_t, u_t), \quad s_t \in \mathcal{X} \subset \mathbb{R}^n,\ u_t \in \mathcal{U} = [\underline{u}, \bar{u}]^m, \quad (1)$$

obtained from a control period Δt. Here f is unknown or too costly to integrate at the sampling rate, so a true-dynamics rollout serves as an oracle reference alone, and the rollout model is learned from interaction data $\mathcal{D} = \{(s_i, u_i, s_i^+)\}$. Throughout, $\mathcal{X}$ is the operating set, the region to which the task confines the plant, and every statement below is made over it. The task defines features $b(s) \in \mathbb{R}^d$, which the rollout model predicts and the cost reads. The controller tracks an output $y = S_y\, b(s) \in \mathbb{R}^{d_y}$ selected from those features by a constant S_y of full row rank.

The plants considered here are control-affine and velocity-commanded, $\dot{s} = f_c(s) + G(s)u$ with output $y = h(s)$. Integrating over one control period under a zero-order hold gives the response of the tracked output,

$$y_{t+1} - y_t = \big(\omega(s_t) + \Phi(s_t)\, u_t\big)\Delta t + O(\Delta t^2), \quad (2)$$

with drift response $\omega = \nabla h\, f_c \in \mathbb{R}^{d_y}$ and *input gain* $\Phi = \nabla h\, G \in \mathbb{R}^{d_y \times m}$. The drift response is a function of the state alone and a model of the state may absorb it, so $\Phi(s)u$ is the only term in which the state and the input meet. The input gain is in general state-dependent. How much it varies over $\mathcal{X}$ is a property of the plant and the task rather than of the controller applied to them.

Everything the controller optimizes is produced by a *rollout model* $\mathcal{R} : (b_t, U) \mapsto (\hat{b}_1, \dots, \hat{b}_T)$ acting on a candidate sequence $U = (u_0, \dots, u_{T-1})$. The cost depends on the candidate only through $\mathcal{R}$, so two controllers that share an optimizer differ exactly in $\mathcal{R}$. At state s_t the controller solves

$$\begin{aligned} \min_{U}\ & J(U) = \sum_{k=1}^{T} c(\hat{b}_k, u_{k-1}) + \ell_T(\hat{b}_T) \\ \text{s.t. }\ & U \in \mathcal{U}^T, \quad \hat{b}_k \in \mathcal{F}, \quad k = 1, \dots, T, \end{aligned} \quad (3)$$

where $\mathcal{F}$ is the free set the task admits. It then applies u_0 to (1) and replans at the next period.

Neither c nor ℓ_T is assumed differentiable or convex in U, and $\mathcal{F}$ is assumed neither convex nor connected. Since no structure is imposed on $\mathcal{F}$, an optimizer may enforce it as a constraint or absorb it into the cost as a penalty. The controller is real-time in the sense that u_0 must be returned within Δt, so the admissibility of a rollout model depends on what the optimizer asks of it in one control step.

B. The Model-Based Diffusion Update

MBD [3] treats (3) as sampling from the Boltzmann target $p_0(U) \propto \exp(-J(U)/\alpha)$ with temperature $\alpha > 0$. It descends a ladder of Gaussian-smoothed marginals $p_\sigma = p_0 * \varphi_\sigma$ over a decreasing schedule $\sigma_1 > \cdots > \sigma_S$.[1] Applying Tweedie's formula to a smoothed marginal expresses its score as a denoising mean shift,

$$\nabla_U \log p_\sigma(U) = \big(\mathbb{E}[\,U_0 \mid U\,] - U\big)/\sigma^2. \quad (4)$$

Since φ_σ is the density of the proposal $U' \sim \mathcal{N}(U, \sigma^2 I)$, the posterior mean in (4) is a ratio of expectations under that proposal, and drawing N candidates $U_k = U + \sigma\varepsilon_k$ with $\varepsilon_k \sim \mathcal{N}(0, I)$ and projecting them onto $\mathcal{U}^T$ gives its self-normalized estimator,

$$\widehat{\mathbb{E}}[U_0 \mid U] = \sum_{k=1}^{N} w_k U_k, \quad w_k = \frac{e^{-(J(U_k) - J_{\min})/\alpha}}{\sum_l e^{-(J(U_l) - J_{\min})/\alpha}}. \quad (5)$$

The update at stage j is a preconditioned score step with optional Langevin noise,

$$U \leftarrow U + \eta_j\, \widehat{\nabla_U \log p_{\sigma_j}}(U) + \sqrt{2\eta_j}\,\xi, \qquad \eta_j = \eta\,\sigma_j^2, \quad (6)$$

which at $\eta = 1$ and without Langevin noise, the setting used throughout, is the cost-weighted mean $U \leftarrow \sum_k w_k U_k$.

Three properties of (6) are used later, and none of them depends on the rollout model that supplies J. First, the weights (5) are invariant to any term shared by the population. Only differences of cost within a population enter

[1] We write the ladder in variance-exploding form, in which the smoothing kernel alone carries the schedule. The formulation of [3] additionally rescales the iterate, which affects no statement below.

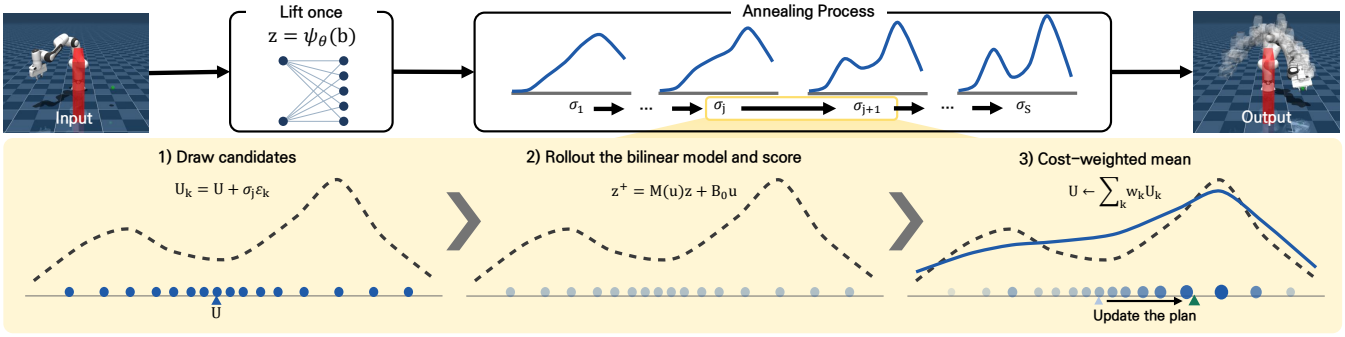


Fig. 1: One control step of BK-MBD. The state is lifted once, and every rollout of the annealed search is then a product of learned matrices.

the update, and the candidates of a population differ in U alone. Second, the S stages are serial, since stage j samples around the iterate produced by stage $j-1$. The N candidates within a stage and the T steps of a rollout admit batching. Third, a single stage at a fixed noise level, with $\eta = 1$ and no Langevin noise, is one cost-weighted average over a Gaussian population drawn around the warm-started iterate. That average is the MPPI update [2], so the annealed schedule is the property separating (6) from the planner it generalizes, and a comparison at a matched total sample budget is available by construction.

The rollout model is evaluated once per candidate at every stage and over the full horizon, so a control step issues NST queries to $\mathcal{R}$.

C. Linear and Bilinear Lifted Rollouts

The Koopman operator [17], [18] advances observables of the state along the flow and is linear on the space of observables, but it is infinite-dimensional. For a system driven by an input, an exact realization requires observables of the input as well as of the state [11]. A finite-dimensional realization therefore truncates the representation of the state and, separately, fixes the form in which the input enters the lifted dynamics. The two classes below agree in the first respect and differ in the second.

We use a learned lift with an exact-decoder convention, $z = \Psi_\theta(b) \in \mathbb{R}^r$, whose leading d coordinates hold b itself and whose remainder is learned. Placing the features in the leading coordinates makes the decoder C with $Cz = b$ a constant selector, and composing it with the output map gives the constant readout $C_y = S_y C \in \mathbb{R}^{d_y \times r}$. Two classes of lifted dynamics are available,

$$\text{linear:} \quad z^+ = Az + Bu, \tag{7}$$

$$\text{bilinear:} \quad z^+ = Az + B_0 u + \textstyle\sum_{i=1}^m u_i B_i z, \tag{8}$$

the latter written compactly as $z^+ = M(u)z + B_0 u$ with $M(u) = A + \sum_i u_i B_i$. For control-affine systems the continuous-time Koopman generator carries a bilinear input structure [11], of which (8) is the finite-dimensional, discrete-time counterpart. Realizations of this form have been identified from data and used for robot control [13].

The two classes differ in exactly one place, the input tangent map. For (8), $\partial z^+/\partial u_i = B_0 e_i + B_i z$ depends on the state through $B_i z$, while for (7), $\partial z^+/\partial u = B$ is one matrix for the whole of $\mathcal{X}$. Both remain in the same computational regime, a bilinear step costing $m+1$ matrix–vector products of size r against one for the linear form. The cost of a rollout step is therefore set by the lifted dimension and the number of input channels alone.

III. Bilinear Koopman Model-Based Diffusion

In this section we propose BK-MBD, which keeps the annealed sampling optimizer of MBD and has every rollout performed by a lifted surrogate. Sec. III-A fixes the class of that surrogate and Sec. III-B identifies a model of that class from data. The rollout is then a learned object rather than a simulator, and Sec. III-C treats the role this leaves to the noise schedule. Fig. 1 shows one control step.

A. The Bilinear Rollout

Sec. II-C leaves the learned coordinates of the lift unspecified. We take

$$z = \Psi_\theta(b) = \begin{bmatrix} b \\ \psi_\theta\big(\tau(b)\big) \end{bmatrix} \in \mathbb{R}^r, \tag{9}$$

where ψ_θ is a small neural network and τ is a fixed feature map that supplies terms the learned part would otherwise have to construct. We write Ψ_i for the ith component of Ψ_θ.

What governs the optimizer is the response the rollout predicts to an applied input, since the weights (5) read only differences of cost within a population whose members differ in the control sequence alone. Decoding one step of the linear class (7) gives $C_y z^+ = C_y A z + C_y B u$, so that response has sensitivity $C_y B$, a single matrix over the whole of $\mathcal{X}$. The lifted dimension r and the encoder Ψ_θ change the value of $C_y B$ and leave it a single matrix, since C_y is a constant selector and B carries no dependence on the state. The plant answers with $\Delta t\, \Phi(s)$ by (2), which takes a different value at every configuration.

Lemma 1 (Input-channel error of a linear lift): Let $\nu_\Phi = \sup_{s,s' \in \mathcal{X}} \|\Phi(s) - \Phi(s')\|$ denote the variation of the input gain over the operating set. For every constant matrix $M \in \mathbb{R}^{d_y \times m}$,

$$\sup_{s \in \mathcal{X}} \big\| \Delta t\, \Phi(s) - M \big\| \;\geq\; \tfrac{1}{2}\, \nu_\Phi\, \Delta t. \tag{10}$$

Algorithm 1 One control step of BK-MBD

Require: state s_t, warm-started nominal U, schedule $\sigma_1 > \cdots > \sigma_S$, temperature α, model $(A, B_0, \{B_i\}_{i=1}^m, \Psi_\theta, C)$

$z \leftarrow \Psi_\theta\big(\tau(b(s_t))\big)$ ▷ encoder once per control step
for $j = 1$ to S **do**
  for $k = 1$ to N, batched **do**
    $U_k \leftarrow \mathrm{proj}_{\mathcal{U}^T}\big(U + \sigma_j \varepsilon_k\big), \quad \varepsilon_k \sim \mathcal{N}(0, I)$
    $z_0^k \leftarrow z$
    for $l = 1$ to T **do**
      $z_l^k \leftarrow M\big(u_{l-1}^k\big) z_{l-1}^k + B_0 u_{l-1}^k$ ▷ $m+1$ mat–vec
      $\hat{b}_l^k \leftarrow C z_l^k$
    end for
    $J_k \leftarrow \sum_{l=1}^T c\big(\hat{b}_l^k, u_{l-1}^k\big) + \ell_T\big(\hat{b}_T^k\big)$
  end for
  $w_k \leftarrow$ softmax weights (5) at temperature α
  $U \leftarrow \sum_{k=1}^N w_k U_k$ ▷ stage update (6)
end for
shift U by one step to warm-start the next period
return u_0

In particular the bound holds at $M = C_y B$ for every pair (A, B) of (7), and therefore at every lifted dimension and for every encoder.

Proof: For any $s, s' \in \mathcal{X}$ the triangle inequality gives

$$\big\|\Delta t\,\Phi(s) - M\big\| + \big\|\Delta t\,\Phi(s') - M\big\| \;\geq\; \Delta t \big\|\Phi(s) - \Phi(s')\big\|,$$

so the larger of the two terms is at least $\frac{\Delta t}{2}\|\Phi(s) - \Phi(s')\|$. Taking the supremum over pairs (s, s') yields (10). The decoded input gain of (7) is one such constant matrix at every lifted dimension and for every encoder, which gives the final claim. ∎

The bound depends on the variation of the input gain alone, and the deficit is not confined to the first step, since $\partial(C_y z_k)/\partial u_j = C_y A^{\,k-j-1} B$ depends on the elapsed steps $k - j$ and not on the state at which the input is applied. A linear rollout therefore predicts the same response to a given input at every configuration a candidate visits. The same deficit underlies the linear–bilinear criterion of [19].

The bilinear class is not subject to this bound, since its decoded input gain is not a constant matrix. That gain

$$\frac{\partial(C_y z^+)}{\partial u_i} = C_y b_{0,i} + C_y B_i z, \qquad i = 1, \ldots, m, \tag{11}$$

is affine in the lifted state, with $b_{0,i}$ the ith column of B_0. The predicted response then matches (2) to first order at every configuration whenever

$$C_y b_{0,i} + C_y B_i\, \Psi_\theta\big(b(s)\big) = \Delta t\, \Phi_i(s), \qquad s \in \mathcal{X}, \tag{12}$$

where Φ_i is the ith column of Φ. Since $C_y = S_y C$ composes two selectors of full row rank, the bilinear class can represent (12) when the entries of Φ lie in the span of $\{1, \Psi_1, \ldots, \Psi_r\}$ over $\mathcal{X}$. Other classes whose input gain may depend on the state can also satisfy (12). What (8) adds is that the dependence enters in the affine form that (2) exhibits.

On a velocity-commanded platform the map from the command frame to the world frame turns as the platform reconfigures. It ranges over its full extent across the postures a task visits, so ν_Φ stays bounded away from zero and (10) bars the linear class. Such a map is built from sines and cosines of the configuration angles and is multilinear in those terms. Letting τ supply those trigonometric functions places the leading terms in the lift and leaves ψ_θ to carry the products among them, the form (12) asks for. On a plant whose input gain varies little the bound is small, and [19] turns the variation of Φ into a test on a given operating set.

The cost follows from the count of Sec. II-B. One control step lifts the measured features once and propagates every candidate in the lifted space afterwards, so the encoder is evaluated once per control step rather than once per candidate, which places it outside both loops of Algorithm 1. The S stages run in series, so the control period is divided among S batched rollouts of N candidates over T steps.

B. Identification

The lifted features and the matrices of the lifted dynamics are fitted jointly to recorded trajectories. Koopman models are ordinarily identified by a one-step fit [9], which suits a model queried one step at a time. A sampling planner instead propagates a candidate over the whole horizon and separates candidates by the cost that propagation accumulates. A model that is accurate at one step and drifts over the horizon has its drift read as a property of the plant. We therefore match the fitting length to the planning horizon and fit every model on length-T snippets by

$$\mathcal{L} = \frac{1}{T} \sum_{k=1}^{T} \Big[\big\| C\hat{z}_k - b_k \big\|^2 + \gamma \big\| \hat{z}_k - \Psi_\theta(b_k)^{\mathrm{sg}} \big\|^2 \Big], \tag{13}$$

starting the rollout at $\hat{z}_0 = \Psi_\theta(b_0)$ and proceeding by (7) or (8). The first term measures the prediction error of the decoded features, which is the quantity the cost reads. The second term, weighted by $\gamma > 0$, holds the propagated latent state to the image of the encoder, which (7) and (8) leave it free to leave. The stop-gradient on the latent target prevents the encoder from lowering the loss by pulling the target toward the prediction.

The bilinear input matrices are initialized at $B_i = 0$. Training then begins exactly at the linear model (7), and the coupling term departs from zero under the fitting objective alone. The two classes share a starting point, an encoder, a dataset and an objective, and they differ in whether the coupling term is permitted to move.

C. Annealing under a Learned Rollout

A learned rollout evaluates $\hat{J} = J + \Delta J$, where ΔJ collects the effect of model error. The weights (5) are invariant to the part of ΔJ common to the population, and the remaining part reorders candidates. A learned rollout therefore admits minima of its own, and under receding horizon the robot executes whatever the sampler settles on. The schedule cannot remove this error from the rollout. Its role is instead to reduce how strongly one control step depends on a single local view of the surrogate cost.

Two properties of the update (6) bear on this, and both belong to the sampling rather than to the model. The first is how widely one stage samples. The proposals $U_k = U + \sigma\varepsilon_k$ are spread about the current plan at a scale set by σ, so σ fixes how far from that plan the cost is evaluated at all. The same scale bounds the update, which is a convex combination of

TABLE I: Implementation settings.

Setting	Manipulator	Drone
Control period Δt	50 ms	50 ms
Budget per trial	120 steps	120 steps
Candidates N, stages S	800, 5	800, 5
Horizon T	15	25
Noise σ	$1.2 \to 0.3$	$1.6 \to 0.3$
Temperature α	0.4	0.5
Lift dimension r, latent weight γ	20, 0.1	13, 0.1
Fixed feature map τ	$[\sin q, \cos q]$	identity
Encoder ψ_θ, layers $\times$ width	2×96, tanh	2×64, tanh
Training snippets, T steps each	6000	4000

those proposals, so the width a stage explores is set primarily by the noise level. That scale is an upper limit rather than a description, and a stage whose population holds nothing better than the current plan returns a weighted mean close to that plan at any σ.

The second is what one stage does with that width. The update moves the nominal by $\sigma \sum_k w_k \varepsilon_k$, which carries σ as a factor, so the noise level sets the size of the step. It also sets where the step points, since by (4) the stage ascends $\log p_\sigma$ and p_σ is the target smoothed at that level. A stage therefore descends the cost smoothed at its own noise level.

One noise level fixes both quantities, so a fixed-noise sampler must trade exploration against refinement. A small σ gives local proposals, a less-smoothed cost and a small final displacement, but it can remain tied to a surrogate-induced local minimum. A large σ evaluates a broader neighborhood, but it reads a more heavily smoothed cost and produces a coarser update. This does not establish that a larger noise level is better; it only characterizes the scale at which the stage reads and moves. The empirical question is whether combining scales is preferable to assigning the same sample budget to one scale.

The annealed schedule separates the two roles across successive stages. It opens at a wide level, where the population is spread furthest from the current plan, and closes at a narrow level, where the smoothing and the final displacement are smallest. Under receding horizon the executed command comes from the last stage. The early stages therefore expose the optimizer to a broader neighborhood of the current plan, while the last stage refines the executed command under a less-smoothed surrogate cost. In the single-basin reaching task below, this suggests that the schedule's advantage should be tied to learned-rollout error, a point tested by repeating the schedules with an exact rollout.

IV. Experiments

We evaluate BK-MBD on two velocity-commanded platforms in simulation and on a physical manipulator.

The experiments that follow ask four questions. What must the rollout model represent for the planner to reach the goal, and does it meet the control period? Where does the advantage of the annealed schedule come from? Does the structure of the free space separate the two optimizers on a passage no single convex region covers? And do both requirements bind on a physical robot?

TABLE II: Reaching accuracy under an identical planner.

Rollout model	Reach 5 cm	Reach 1 cm
MBD-TRUE	10/10	10/10
DK-MBD	10/50	1/50
DK-MBD-LARGE	11/50	1/50
MLP-MBD	41/50	0/50
DK-MBD-SPLIT	50/50	50/50
BK-MBD (ours)	**50/50**	**50/50**

A. Setup

Platforms. Sec. III-A leaves the input gain to be instantiated on each platform, and Table I gives the fixed feature map τ of (9) that follows from it. The Franka Research 3 (FR3) accepts joint velocities through a gravity-compensated servo, tracks the tool center point (TCP) between its fingertips, and is observed as $b = [q, p_{\mathrm{tcp}}]$. Its input gain is the manipulator Jacobian.[2] The drone accepts body-frame linear velocities and a yaw rate, tracks its world-frame position, and is observed as $b = [p, \sin\psi, \cos\psi]$. Its input gain is the yaw rotation. The manipulator therefore carries the experiments of Secs. IV-B and IV-C and the hardware experiment of Sec. IV-E, and the drone carries the non-convex passage of Sec. IV-D.

Task and protocol. Every comparison below holds the optimizer or the rollout model fixed and exchanges the other. In the reaching experiments each trial starts from rest and drives the tracked output toward a goal over the fixed budget of Table I, without early termination. A trial reaches a tolerance at the first control step whose tracking error falls below it, and we report the 5 cm and 1 cm tolerances. Where a condition fails to reach, we report the tracking error at the end of the budget. Candidates are scored by the squared tracking error with a control penalty and a terminal penalty. Each dataset is collected from randomized initial conditions over the operating set under randomized velocity commands that keep a coherent direction across a snippet, so the configuration travels far enough for the state dependence of the input gain to be visited. The reaching experiments cross 5 training seeds, each with its own dataset, with 10 goals for 50 trials per condition, and the trials are paired across conditions. A condition that carries no learned model is run over the 10 goals alone. All timings are measured on an Intel Core Ultra 5 225F with ten cores and no GPU. Simulation uses ten PyTorch CPU threads, whereas physical-robot trials limit PyTorch inference to four threads.

B. Rollout Class and Planning Latency

We settle whether the input gain is what decides reaching by exchanging the rollout model alone. Table II lists them by how the input gain enters. DK-MBD leaves it a constant matrix. DK-MBD-LARGE keeps that matrix but raises the lifted dimension, and MLP-MBD lets it depend on the state without structure, so the two separate capacity and

[2]The FR3 is used deliberately because its analytic input gain provides a reference for testing whether the learned rollout recovers the same configuration dependence from data, while also enabling the DK-MBD-SPLIT ablation. The goal is not to replace the known Jacobian, but to validate a rollout structure applicable when the corresponding input map is unknown or not readily available.

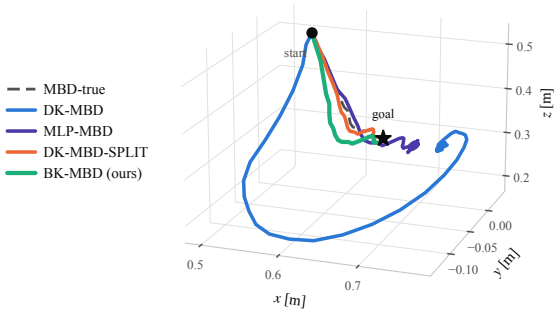


Fig. 2: Executed tool paths on one goal, one of the trials Table II counts. The five runs share the planner, the training seed and the random stream.

TABLE III: End-to-end planning latency per receding-horizon control step over all trials.

Rollout model	Median [ms]	Worst [ms]	Misses
MBD-TRUE	2598	2745	1200
DK-MBD	7.6	12.5	0
BK-MBD (ours)	11.0	14.7	**0**

TABLE IV: Noise schedules at an equal total sample budget. The rows marked (oracle) use the true-dynamics rollout, which carries no model error.

Schedule	Reach 1 cm	Median final error [m]
$1 \times NS$ wide (MPPI)	33/50	0.010
$1 \times NS$ narrow (MPPI)	15/50	0.041
$S \times N$ wide	46/50	0.012
$S \times N$ narrow	26/50	0.045
$S \times N$ anneal (ours)	**50/50**	**0.0097**
$S \times N$ narrow (oracle)	10/10	0.009
$S \times N$ anneal (oracle)	10/10	0.007

expressiveness from the input gain. BK-MBD represents the state-dependent gain inside the model, and DK-MBD-SPLIT is the ablation that supplies it from the analytic forward kinematics instead. MBD-TRUE rolls out the true dynamics as the oracle reference.

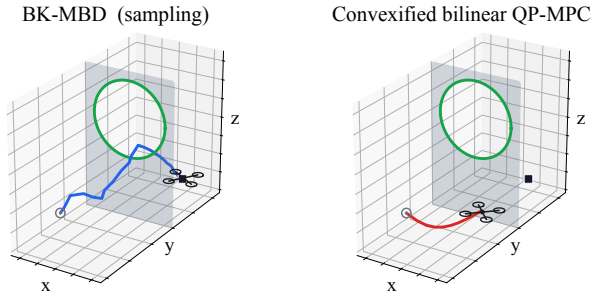
On the manipulator, BK-MBD reached the 1 cm tolerance on $50/50$ trials, matching the oracle, while DK-MBD reached it on $1/50$ and came to rest a median 0.248 m from the goal. DK-MBD-SPLIT also reached $50/50$, and it differs from DK-MBD in the input gain alone, so that is what separated the two, as Lemma 1 predicts.

DK-MBD-LARGE and MLP-MBD stopped a median 0.239 m and 0.054 m away. The bound (10) carries no lifted dimension, so a larger dictionary still leaves a single matrix, and MLP-MBD can represent a state-dependent gain but did not identify it from the same data under the same objective. Neither capacity nor expressiveness therefore removes the failure, which leaves the input gain as the cause. Fig. 2 draws one goal of the comparison.

We then measure whether the rollout model meets the control period. Table III reports the end-to-end planning time per control step, which covers every line of Algorithm 1 over all five annealing stages. The BK-MBD planner returned at a median of 11.0 ms with a worst case of 14.7 ms and missed no deadline over all 50 trials, while the oracle took a median of 2598 ms and overran the 50 ms period at every one of its 1200 control steps. The gap follows from what one control step asks of the rollout model. The 2 ms integrator turns $NST = 60{,}000$ rollout steps into 1.5 million physics substeps, and the $ST = 75$ batched steps among them lie on a serial chain. On the manipulator, a bilinear step issues $m+1 = 8$ matrix–vector products instead of one, yet shared lifted-state batching raises the planner time only from 7.6 ms to 11.0 ms, less than a factor of 1.5.

TABLE V: Step size and step quality at a fixed noise level. Efficiency is the error removed per unit displacement.

σ	$\lvert\Delta U\rvert$	Error removed	Efficiency ($\times 10^3$)
0.3	0.109	0.00665	61.7
0.8	0.292	0.00571	19.2
1.2	0.418	0.00409	9.9
anneal	0.109	0.00525	48.4



(a) (b)

Fig. 3: Non-convex passage on the drone. The wall admits a single circular aperture. (a) BK-MBD threads the aperture and reaches the far-side goal. (b) the convexified controller holds the near side.

C. The Contribution of the Annealed Schedule

We settle whether the advantage of the annealed schedule comes from learned-rollout error by varying the noise schedule alone over the bilinear model. The five schedules of Table IV share one total budget of NS candidates per control step, written as stages $\times$ candidates per stage. $1 \times NS$ draws the whole budget at once and $S \times N$ spreads it over S stages, at the wide level $\sigma = 1.2$, the narrow level $\sigma = 0.3$, or annealed from wide to narrow. By the third property of Sec. II-B, the single-stage conditions are the MPPI update at the same budget. The obstacle-free task has no true-cost multimodality, so annealing gains cannot arise from it.

Table IV reports the outcome at the 1 cm tolerance, where the schedules separate. The annealed schedule reached on all 50 trials and left the lowest tracking error at the end of the budget. It gains 17 and 35 goals over the single-stage wide and narrow MPPI conditions, and 4 and 24 over the conditions that repeat the same level S times. Neither the sample budget nor the application of several serial stages therefore accounts for the result, and the changing noise scale is the separating factor.

Table V reports the displacement one control step produces

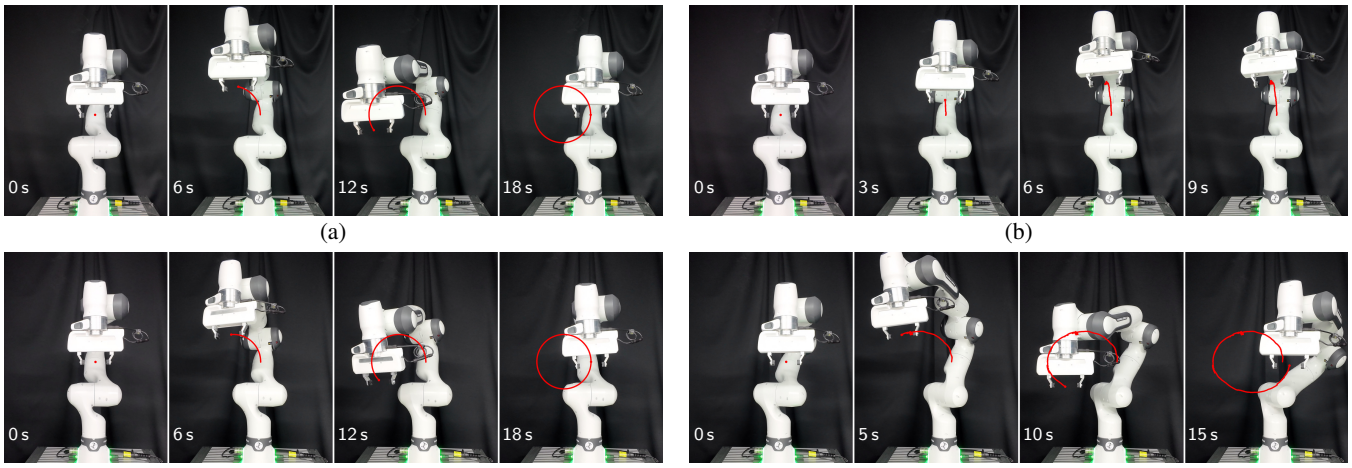


Fig. 4: Circular tracking on the physical FR3, with postures sampled at equal intervals over one revolution and the executed tool path traced in red. (a) BK-MBD, which completes the revolution. (b) DK-MBD, from the start of the command to the guard stop. (c) DK-MBD-SPLIT. (d) BK-MPPI.

and the tracking error it removes, averaged over control steps and trials, and their ratio as an efficiency. From $\sigma = 0.3$ to $\sigma = 1.2$ the displacement grows by a factor of 3.8 while the efficiency drops by a factor of six. The annealed schedule instead executes at the displacement of the narrow level, 0.109, and retains an efficiency of 48.4 against the 9.9 of the wide level. The final stage produces the executed step, while the preceding wider stages have already exposed the iterate to a broader neighborhood of the current plan.

The model and the true dynamics disagree on the settled plan of the narrow schedule, 0.024 m against 0.053 m, and agree more closely on that of the annealed schedule, 0.020 m against 0.015 m. Repeating the two schedules on the true-dynamics rollout, over the same goals and the same planner stream, removes the separation. The narrow schedule reaches $10/10$ as the annealed schedule does, against $26/50$ under the learned rollout, and its settled error moves from 0.045 to 0.009 m. The narrow level settles quickly on either rollout, so what it loses under the learned one is consistent with sensitivity to the surrogate landscape rather than with an inability of the sampler to refine a plan.

D. Non-Convex Passage

We settle whether the structure of the free space is what separates the two optimizers by exchanging the optimizer alone. A single circular aperture in a wall is the only route to the goals on the far side, and the straight chord to any goal crosses the wall below the opening. The drone flies 50 trials, and both planners receive the same trained bilinear model. BK-MBD takes the wall as a cost penalty, while the convexified controller relinearizes the keep-out set into one convex region per step and solves the resulting QP over four iterations along the nominal rollout [13].

BK-MBD reached the 1 cm tolerance on $50/50$ goals and the convexified controller on $10/50$, and BK-MBD executed no constraint violation although it treats the wall as a penalty alone. Fig. 3 draws the paths of the two planners.

TABLE VI: Circular tracking on the physical FR3 over ten planner seeds.

Method	Full run	RMSE [mm]	Median / worst [ms]
BK-MBD (ours)	$10/10$	7.09 ± 0.12	22.9 / 44.7
DK-MBD	$0/10$	39.75 ± 0.67	10.9 / 23.4
DK-MBD-SPLIT	$10/10$	4.97 ± 0.09	28.6 / 83.0
BK-MPPI	$8/10$	15.76 ± 5.23	17.0 / 53.1
MBD-TRUE	$0/10$	41.22 ± 0.07	1617 / 1762

The free space is the two sides of the wall joined through the aperture. Any convex region that excludes the wall lies entirely on the near side, so the far-side goal is infeasible at every step and the solve converges to the point of that region closest to the goal. The convexified controller settled a median of 0.030 m from the wall, the margin of its own keep-out set, and the ten goals it reached were those closest to the aperture axis, at most 0.148 m against a median of 0.291 m. A convexification of the dynamics and one of the free space act on it at once, but frozen dynamics would instead fail by tracking a poor nominal, so both point to the free space.

E. Validation on the Physical Robot

Five methods listed in Table VI, track a target moving along a circle on the physical FR3 under the same CPU budget. The target traverses a circle of radius 0.10 m and period 16 s in the yz plane, and the trajectory is not given in advance, so the controller sees only the target position at each control step. Fig. 4 shows the postures over one revolution. Each trial runs for 35 s, a 3 s lead-in followed by two revolutions. A guard commands zero velocity and ends the trial when the Cartesian tracking error exceeds 80 mm for 0.25 s. Each rollout class is run ten times, over planner seeds 0 to 9. The models are identified from data recorded on this robot under the protocol of Sec. IV-A. BK-MPPI uses the single-stage narrow condition, which shows a distinct learned-rollout effect in Table IV.

Only BK-MBD and DK-MBD-SPLIT completed all ten trials. DK-MBD triggered a guard stop in every trial, while

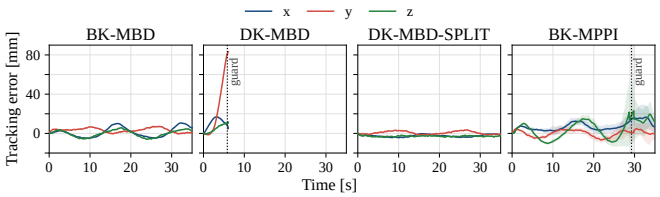


Fig. 5: Signed TCP tracking error per axis on the physical FR3, mean over ten planner seeds with one standard deviation. The dotted line marks the guard stop.

BK-MPPI completed eight and exceeded the guard in the remaining two. MBD-TRUE issued no nonzero command because its planning latency exceeded the control period. BK-MBD returned at a median of $22.9\,\mathrm{ms}$ with a worst case of $44.7\,\mathrm{ms}$ and stayed within the $50\,\mathrm{ms}$ period over all ten trials, whereas DK-MBD-SPLIT overran it at $83.0\,\mathrm{ms}$, so BK-MBD is the only method here that met both the tracking task and the deadline.

The failure of DK-MBD lies in the representation and that of MBD-TRUE in the computation. The state-independent input derivative CB_0 of DK-MBD represents only the mean input effect over the training distribution, as reflected by the axis errors of Fig. 5 and its directional cosine of 0.318, against 0.914 for BK-MBD. The successful tracking of DK-MBD-SPLIT corroborates on hardware the representation limit identified in the preceding simulation experiment. Likewise, the contrasting closed-loop outcomes of BK-MBD and BK-MPPI under the same BK rollout show that the benefit of the annealed schedule persists on the physical robot. Rolling the settled plans of the completed BK-MPPI trials through the model and the robot gave terminal errors of $8.90 \pm 2.08\,\mathrm{mm}$ and $19.25 \pm 8.21\,\mathrm{mm}$, respectively, with the robot error larger in all but one trial. Because this discrepancy belongs to the learned BK rollout shared by both planners rather than to MPPI itself, the result supports the earlier conclusion that the annealed schedule is less sensitive to surrogate error than a single-stage narrow update.

V. Conclusion

We introduced BK-MBD, which runs the annealed sampling of MBD at the control rate by performing every rollout with a bilinear Koopman model identified from interaction data. In simulation, exchanging one component of the controller at a time separated three effects: the input gain of the rollout, the noise schedule under a learned rollout, and the structure of the free space. On a physical manipulator, BK-MBD tracked a moving target and met the control deadline at every step; no other method did both. The input gain and the noise schedule kept the roles that simulation assigned them. MBD owes its behavior in contact-rich settings to a rollout that contains the contact, and a lifted model would instead have to represent it. Every task here is contact-free, so whether the substitution preserves that behavior is left for future work.

References

[1] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Aggressive driving with model predictive path integral control," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2016, pp. 1433–1440.

[2] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou, "Information theoretic MPC for model-based reinforcement learning," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2017, pp. 1714–1721.

[3] C. Pan, Z. Yi, G. Shi, and G. Qu, "Model-based diffusion for trajectory optimization," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 37, pp. 57 914–57 943, 2024.

[4] Y. Song and S. Ermon, "Generative modeling by estimating gradients of the data distribution," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 32, 2019.

[5] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole, "Score-based generative modeling through stochastic differential equations," in *International Conference on Learning Representations (ICLR)*, 2021.

[6] H. Xue, C. Pan, Z. Yi, G. Qu, and G. Shi, "Full-order sampling-based mpc for torque-level locomotion control via diffusion-style annealing," in *2025 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2025, pp. 4974–4981.

[7] M. O. Williams, I. G. Kevrekidis, and C. W. Rowley, "A data-driven approximation of the Koopman operator: Extending dynamic mode decomposition," *J. Nonlinear Sci.*, vol. 25, no. 6, pp. 1307–1346, 2015.

[8] J. L. Proctor, S. L. Brunton, and J. N. Kutz, "Dynamic mode decomposition with control," *SIAM J. Appl. Dyn. Syst.*, vol. 15, no. 1, pp. 142–161, 2016.

[9] B. Lusch, J. N. Kutz, and S. L. Brunton, "Deep learning for universal linear embeddings of nonlinear dynamics," *Nat. Commun.*, vol. 9, p. 4950, 2018.

[10] W. Hao, Y. Fang, Z. Lu, and S. Mou, "Accelerating sampling-based control via learned linear Koopman dynamics," *arXiv preprint arXiv:2603.05385*, 2026. [Online]. Available: https://arxiv.org/abs/2603.05385

[11] L. C. Iacob, R. Tóth, and M. Schoukens, "Koopman form of nonlinear systems with inputs," *Automatica*, vol. 162, p. 111525, 2024.

[12] M. Korda and I. Mezić, "Linear predictors for nonlinear dynamical systems: Koopman operator meets model predictive control," *Automatica*, vol. 93, pp. 149–160, 2018.

[13] D. Bruder, X. Fu, and R. Vasudevan, "Advantages of bilinear Koopman realizations for the modeling and control of systems with unknown dynamics," *IEEE Robot. Autom. Lett.*, vol. 6, no. 3, pp. 4369–4376, 2021.

[14] M. Janner, Y. Du, J. B. Tenenbaum, and S. Levine, "Planning with diffusion for flexible behavior synthesis," in *International Conference on Machine Learning (ICML)*, 2022.

[15] C. Chi, Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake, and S. Song, "Diffusion policy: Visuomotor policy learning via action diffusion," *The International Journal of Robotics Research*, vol. 44, no. 10-11, pp. 1684–1704, 2025.

[16] J. Carvalho, A. T. Le, M. Baierl, D. Koert, and J. Peters, "Motion planning diffusion: Learning and planning of robot motions with diffusion models," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2023, pp. 1916–1923.

[17] B. O. Koopman, "Hamiltonian systems and transformation in Hilbert space," *Proc. Natl. Acad. Sci.*, vol. 17, no. 5, pp. 315–318, 1931.

[18] I. Mezić, "Spectral properties of dynamical systems, model reduction and decompositions," *Nonlinear Dyn.*, vol. 41, pp. 309–325, 2005.

[19] K. Lee and S. Kim, "Linear or bilinear: A criterion for Koopman rollouts in sampling-based predictive control," *Research Square*, Aug. 2026, preprint. [Online]. Available: https://doi.org/10.21203/rs.3.rs-10732691/v1