

Optimal Control with Learned Critics under Unmodeled State Dependencies

Philipp Schoch, Markus Ryll
Autonomous Aerial Systems Lab
Technical University of Munich
{philipp.schoch,markus.ryll}@tum.de

Abstract: Model Predictive Control (MPC) provides a structured and constraint-aware mechanism for decision-making, but its reliance on optimization-friendly analytical dynamics models limits its use in tasks with contacts and other hard-to-model state dependencies. Model-free reinforcement learning avoids explicit modeling assumptions but typically requires large amounts of interaction data. We present a learning-based MPC framework that combines the data efficiency and structure of local model-based planning with learned components that compensate for incomplete dynamics and finite-horizon myopia. The method augments a nominal analytical model with a residual dynamics network that learns missing state-dependent effects from data and combines the resulting planner with a learned action-value critic that injects long-horizon MDP structure into the local iLQR optimization. To make this practical at reinforcement-learning scale, we develop a GPU-accelerated batched iLQR solver that evaluates learned dynamics and critic networks inside the optimal-control loop and solves thousands of trajectory-optimization problems in parallel. The complete system is integrated into a robotics simulator, enabling scalable model-based reinforcement learning under incomplete dynamics. Experiments on biased and incompletely modeled control tasks show that the approach improves closed-loop control performance while preserving the model-based structure needed for efficient constrained trajectory optimization.

Keywords: Model Predictive Control, Model-Based Reinforcement Learning, iLQR, Value Function Approximation

1 Introduction

Computing optimal control policies for agents acting in stochastic environments is a central challenge in sequential decision making. Formally, this amounts to solving a Markov Decision Process (MDP) [1]: finding a policy that maximizes expected cumulative reward over potentially infinite horizons. The field has produced a rich landscape of approaches, broadly organized into trajectory optimization, model-free reinforcement learning (RL), and model-based reinforcement learning methods [2].

Model Predictive Control (MPC) performs trajectory optimization in a receding-horizon fashion, enabling precise control and explicit satisfaction of state and input constraints. However, MPC requires an optimization-friendly analytical dynamics model, often difficult or impossible to derive for systems involving contacts or other highly nonlinear effects, and breaks down if relevant state variables remain unmodeled. Contact-rich tasks are a representative case, because making and breaking contact introduces discontinuous phase boundaries over which gradient information can become uninformative [3], while classical multi-phase or mixed-integer formulations lead to large nonlinear programs that are brittle and rarely compatible with real-time replanning [4]. A second limitation is the finite horizon itself. MPC applies a finite-horizon surrogate to an infinite-horizon MDP, so

a horizon that is too short can make the controller greedy, trap it in local minima, or prevent task completion when intermediate subgoals must be reached before the final objective becomes attainable [5]. A third limitation is that standard MPC formulations operate on explicit state variables and analytical models, making it nontrivial to use high-dimensional sensor observations such as rgb or depth images directly inside the planner.

Model-free RL sidesteps these limitations: learned policies are cheap at inference and can consume high-dimensional observations. However, model-free methods are notoriously sample-inefficient, scale poorly with state-space dimensionality, and struggle with long-horizon tasks where reward-sparsity impedes exploration [1, 6].

Model-based RL promises to combine the data efficiency of planning with the flexibility of learning [2]. A compelling paradigm in this space is to use MPC as a critic: the planner provides a structured, locally accurate estimate of the action-value function, which can be used to bootstrap a global Q -approximator which after training supplies long-horizon guidance back to the MPC, allowing even to correct for mismatches in the dynamics model. [7] demonstrates this in the setting of gradient-free sampling-based control. A key remaining challenge is that MPC critics still require an accurate dynamics model, a requirement that fails whenever relevant state variables lack a tractable analytical description, e.g. object contact dynamics in manipulation.

In this work, we develop a learning-augmented constrained MPC framework that embeds two learned components directly into the planner. A residual dynamics model augments the nominal analytical model and exposes otherwise unmodeled state dependencies to optimization, while a learned critic injects global MDP information into the finite-horizon objective to reduce myopia and improve guidance under multi-phase cost-structure; unlike in sampling-based MPC [7], the critic enters the Riccati recursion through its first- and second-order derivatives. Following the DDP formulation, we realize this approach with a GPU-accelerated batched ALM-iLQR solver that integrates both learned components inside the optimal-control loop and scales to thousands of parallel environments, enabling efficient model-based reinforcement learning.

Core contributions: (i) a constrained MPC formulation that combines residual dynamics learning with a learned critic, and (ii) a learning-based DDP-style solver that makes this formulation practical for model-based reinforcement-learning, scaling to thousands of parallel environments.

2 Preliminaries

We consider a discrete-time MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, l, P, \gamma, p_0)$ [1], where $\mathcal{S}$ is the state space, $\mathcal{A}$ the action space, $l(x, u)$ a stage cost, P a stochastic transition model with $x_{k+1} \sim P(\cdot \mid x_k, u_k)$, $\gamma \in (0, 1]$ a discount factor, and p_0 the initial state distribution. Let $\mathbb{P}_{\mathcal{M}}^{\pi}(\cdot \mid x, u)$ denote the trajectory distribution induced by $(x_0, u_0) = (x, u)$ and following π thereafter. The action-value and value functions are $Q_{\mathcal{M}}^{\pi}(x, u) = \mathbb{E}_{\mathbb{P}_{\mathcal{M}}^{\pi}(\cdot \mid x, u)} \left[\sum_{i=0}^{\infty} \gamma^i l(x_i, u_i) \right]$ and $V_{\mathcal{M}}^{\pi}(x) = \mathbb{E}_{u \sim \pi(\cdot \mid x)} [Q_{\mathcal{M}}^{\pi}(x, u)]$. The control objective is $\pi^* = \arg \min_{\pi} \mathbb{E}_{x \sim p_0} [V_{\mathcal{M}}^{\pi}(x)]$. For any horizon $H \geq 2$, the action-value function admits the finite-horizon decomposition

$$Q_{\mathcal{M}}^{\pi}(x, u) = \mathbb{E}_{\mathbb{P}_{\mathcal{M}}^{\pi}(\cdot \mid x, u)} \left[\sum_{i=0}^{H-2} \gamma^i l(x_i, u_i) + \gamma^{H-1} Q_{\mathcal{M}}^{\pi}(x_{H-1}, u_{H-1}) \right]. \quad (1)$$

Classical MPC closes this recursion with a terminal cost, $Q_{\mathcal{M}}^{\pi}(x_{H-1}, u_{H-1}) \approx l_T(x_{H-1}, u_{H-1})$ [8]. While sufficient when the terminal cost is well designed, this local approximation may miss the global structure of the MDP, especially when the objective requires reaching an intermediate subgoal [5]. In the following we work with deterministic planned trajectories to simplify notation; the stochastic setting is retained by taking expectations over the induced trajectory distribution.

The paper is structured as follows: Section 3 introduces the blended-critic objective and explains how learning of the residual dynamics function and the learned critic integrates into the framework. Section 4 describes the iLQR solver and the specific realization of the blended-critic objective.

Section 5 evaluates the method on a set of simulated control tasks, and Sections 6 and 7 discuss the method’s shortcomings and give an outlook on potential extensions.

3 Optimal Control Objective

3.1 Blending the Learned Critic

As established in Eq. (1), MPC estimates $Q_{\mathcal{M}}^{\pi}$ by rolling out a local trajectory. However, the accuracy of this estimate depends heavily on the length of the rollout and well-designed terminal cost function. A global learned critic $\hat{Q}_{\theta}(x, u)$, parameterized by a neural network with weights θ , can in principle capture the full MDP structure, but suffers from the usual approximation errors of neural networks, making it both hard to train and hard to integrate into a planner (cf. Section 3.2, cf. [1]). Following [7], we therefore blend both estimates along the H -step planned trajectory $(x_i, u_i)_{i=0}^{H-1}$. As can be seen later, this will allow us to trade off bias in the nominal model with approximation bias in the learned critic, while reducing variance in the value targets during training.

Following the general Bellman recursion $Q_i = l_i + \gamma Q_{i+1}$, for any $\lambda \in [0, 1]$, we define the blended Q -estimate recursively for $i = 0, \dots, H-2$ as

$$Q^{\lambda}(x_i, u_i) = (1 - \lambda)\hat{Q}(x_i, u_i) + \lambda(l(x_i, u_i) + \gamma Q^{\lambda}(x_{i+1}, u_{i+1})), \quad (2)$$

but contrary to [7], we terminate the recursion at the last step by $Q^{\lambda}(x_{H-1}, u_{H-1}) = (1 - \lambda)\hat{Q}(x_{H-1}, u_{H-1}) + \lambda l(x_{H-1}, u_{H-1})$, as a weighted average between the finite horizon approximation of MPC and the learned critic. Following Appendix A we can rollout this recursion into the explicit form

$$Q_0^{\lambda} = \hat{Q}_0 + \lambda \left(\sum_{k=0}^{H-2} (\lambda\gamma)^k B_k + (\lambda\gamma)^{H-1} (l_{H-1} - \hat{Q}_{H-1}) \right), \quad (3)$$

where $B_i = l_i + \gamma\hat{Q}_{i+1} - \hat{Q}_i$ is the one-step Bellman residual of the learned critic. Setting $\lambda = 0$ recovers $\hat{Q}$ directly; $\lambda = 1$ recovers the standard MPC rollout; intermediate values let $\hat{Q}$ act as a global shaping term while preserving local model accuracy. In Section 5, we additionally vary the blend weight of the critic at the end of the horizon separately, denoted λ_f ; in all other experiments, $\lambda_f = \lambda$.

3.2 Learning the Critic

The critic $\hat{Q}_{\theta}$ is trained by stochastic gradient descent on the mean squared Bellman error [1]

$$\mathcal{L}_Q(\theta) = \mathbb{E}_{(x_k, u_k, y_k) \sim \mathcal{D}} \left[\left(\hat{Q}_{\theta}(x_k, u_k) - y_k \right)^2 \right]. \quad (4)$$

The target y_k is constructed either from short rollout fragments or from the MPC continuation value $V^{\lambda}(x_{k+1}) = \min_u Q^{\lambda}(x_{k+1}, u)$ which can be obtained for free after each MPC solve.

TD(n) targets. A first option is to use standard n -step targets built from contiguous rollout fragments [1]. This is simple and efficient, and larger n can stabilize the targets through the usual bias-variance trade-off. However, the method becomes sensitive to stale replay data when buffers are small or policy updates are infrequent.

MPC-based targets. A second option is to use this continuation value and form the one-step target $y_k^{\text{MPC}} = l(x_k, u_k) + \gamma V^{\lambda}(x_{k+1})$, with the usual truncation at terminal transitions. In the offline variant, the planner stores the successor blended- Q -estimate during data collection and training reuses this cached value [7]. This is fast and benefits from the full planner lookahead, but the targets become stale as the critic changes. In the online variant, the successor value is recomputed during training by solving a fresh MPC problem for each minibatch element. This removes staleness and supports fully off-policy replay, but is typically prohibitively expensive: with iLQR, each batch element requires a converged optimal control solve, and shuffled minibatches prevent amortizing convergence across time steps.

Value ensembles. Both target constructions remain bootstrapped and therefore inherit the usual risk of amplifying value-estimation errors [9]. In our setting this is especially problematic because MPC will immediately exploit critic mismatches induced by epistemic uncertainty. To reduce this effect, we follow the spirit of double Q-learning and ensemble bootstrapping [9, 10] and use an ensemble critic with N value heads. During each SGD step, one head is randomly selected for updating, while the remaining $N - 1$ heads provide the target values. Their outputs can be aggregated by a (soft/hard) maximum, minimum, or mean operator; under our cost convention, the maximum is conservative, the minimum is optimistic, and we use the mean as a balanced compromise [10].

At inference time, the ensemble disagreement and the disagreement of the ensembles gradients also provide an uncertainty estimate of the learned critic. This gives us a way to downweigh the λ -contribution of in the MPC in regions where the Value-heads disagree. Contrary to [5], where a softmax (softmin under our cost-convention) is used to encourage exploration, we compute the standard deviation across the normalized value-head outputs (and their gradients) and attenuate the learned-critic contribution by an exponential factor $\alpha = e^{-\kappa\sigma}$ where κ is a tuning parameter.

3.3 Learning the Dynamics Residual

Many control tasks involve state variables that are directly observable, but for which no sufficiently accurate analytical dynamics model is available. In manipulation, for example, one may observe the pose and velocity of a grasped object without modeling the underlying contact dynamics explicitly. To allow the planner to reason over such quantities, we augment the state with components whose dynamics are incomplete and learn only the missing part from data. Formally, we partition the state as $x = [x_p^\top, x_s^\top]^\top$, where $x_p \in \mathbb{R}^{n_p}$ denotes primary states with known dynamics and $x_s \in \mathbb{R}^{n_s}$ denotes secondary states whose evolution must be learned. We then use the composite dynamics

$$\dot{x} = f_{\text{tot}}(x, u) = f_p(x, u) + \mathbf{I} f_\phi(x, u), \quad (5)$$

where $\mathbf{I} = \text{diag}(m_1, \dots, m_n)$ with $m_i \in \{0, 1\}$ is a binary mask selecting which state dimensions receive a learned residual correction, while dimensions with trusted dynamics, such as trivial integrators, are left unchanged.

We fit the residual from rollout data. Given samples (x_k, u_k, x_{k+1}) , a one-step target is obtained from the finite difference $\dot{x}_k \approx (x_{k+1} - x_k)/\Delta t$; for states containing quaternion components, standard finite differences are replaced by logarithmic-map differences on $SO(3)$ so that the targets are well defined despite the $q \equiv -q$ sign ambiguity, as described in Appendix H. Because one-step fitting alone does not control the error accumulated over repeated numerical integration inside the planner, we optimize both over mini-batches of the local regression loss and a rollout-consistency loss on contiguous trajectory fragments,

$$\mathcal{L}(\phi) = \sum_k \|\mathbf{I}(f_{\text{tot}}(x_k, u_k) - \dot{x}_k)\|^2, \quad \mathcal{L}_{\text{roll}}(\phi) = \frac{1}{R} \sum_{t=1}^R \|\mathbf{I}(x_{k+t}^{\text{pred}} - x_{k+t}^{\text{true}})/\Delta t\|^2, \quad (6)$$

where x_{k+t}^{pred} is obtained by recursively integrating f_{tot} from the previous predicted state. This term penalizes compounding rollout error on the residual-enabled state components, rather than only minimizing local one-step errors. The resulting model allows MPC to plan directly in the augmented state space. Training then alternates between collecting rollouts, fitting the residual dynamics, and updating the learned critic.

3.4 Guiding the Critic

Integration of Auxiliary Information. Rather than learning directly from raw (x, u) , both learned modules in the solver have the option to tap into task-specific feature pre-processors,

$$\dot{x} = f_p(x, u) + \mathbf{I} f_\phi(g_f(x, u, o)), \quad \hat{Q}_\theta(x, u \mid o) = \hat{Q}_\theta(g_Q(x, u, o)), \quad (7)$$

where o denotes optional task context, such as a goal, object pose, or a latent visual embedding $z = e_\psi(o)$ produced by a shared visual encoder. The maps g_f and g_Q may differ and let the

learned components operate in task coordinates rather than embodiment-specific raw states. For manipulation tasks, for instance, it can be convenient to compute inside g_Q forward-kinematics-derived features such as $p_{\text{cube}} - p_{\text{EE}}$, $p_{\text{goal}} - p_{\text{cube}}$, and v_{EE} instead of raw joint variables.

4 OCP Solver

4.1 iLQR

Approaching the MPC from the MDP perspective (cf. Section 2), the DDP [8] based iLQR approach [11] is a natural choice for our optimal control solver. Given an initial control sequence $(u_i)_{i=0}^{H-1}$, a forward rollout under $x_{i+1} = f(x_i, u_i)$ produces a reference trajectory $(x_i, u_i)_{i=0}^{H-1}$ and its rollout value V . Following DDP, the backward pass uses the Bellman recursion $Q_i = l_i + \gamma Q_{i+1}$ and forms a local quadratic model of the action-value function at each time step,

$$Q(\delta x, \delta u) \approx \frac{1}{2} \begin{bmatrix} 1 \\ \delta x \\ \delta u \end{bmatrix}^T \begin{bmatrix} 0 & Q_x^T & Q_u^T \\ Q_x & Q_{xx} & Q_{xu} \\ Q_u & Q_{ux} & Q_{uu} \end{bmatrix} \begin{bmatrix} 1 \\ \delta x \\ \delta u \end{bmatrix}. \quad (8)$$

Minimizing this local model yields the affine feedback update

$$\delta u_i = k_i + K_i \delta x_i, \quad k_i = -Q_{uu,i}^{-1} Q_{u,i}, \quad K_i = -Q_{uu,i}^{-1} Q_{ux,i}. \quad (9)$$

iLQR drops the second-order dynamics tensor terms of full DDP, yielding the more economic Gauss-Newton approximation [11]. In the backward sweep, we compute the feed-forward terms k_i , feedback gains K_i , and the expected value-change terms used by the line search. The stage derivatives, dynamics Jacobians, and Hessians can be batch-computed before this sequential sweep, which allows us to GPU-parallelize the heavy computations related to auto-diff gradient and Hessian evaluation beforehand and keep the backward-sweep loop as lightweight as possible. We regularize Q_{uu} using the scheme of Tassa et al. [12], with an aggressive schedule for decreasing or increasing the regularizer depending on line-search success.

The forward line search rolls out candidate controls $\hat{u}_i = u_i + \alpha k_i + K_i dx(\hat{x}_i, x_i)$ from $\hat{x}_0 = x_0$, where α is the line-search step size [12]. The state displacement $dx(\cdot, \cdot)$ is Euclidean for vector states and uses the manifold distance for quaternion states, as described in Appendix H. We accept a candidate when the actual rollout improvement agrees with the expected local-model improvement, using either an Armijo condition or the ratio rule of Tassa et al. [12]; otherwise α is reduced and the rollout is repeated.

4.2 Augmented Lagrangian Method

To handle equality and inequality constraints, we wrap the iLQR primal solve in an Augmented Lagrangian Method (ALM). Let $\tau = (x_0, u_0, \dots, x_{H-1}, u_{H-1})$ denote the planned trajectory. We write the constrained finite-horizon problem with a generic additive objective $J(\tau) = \sum_{i=0}^{H-1} \tilde{l}_i(x_i, u_i)$, where $\tilde{l}_i$ will later be instantiated by the blended MPC stage objective:

$$\begin{aligned} \min_{\tau} \quad & J(\tau) = \sum_{i=0}^{H-1} \tilde{l}_i(x_i, u_i) \\ \text{subject to} \quad & x_0 = x, \\ & x_{i+1} = f(x_i, u_i), \\ & g_i(x_i, u_i) \leq 0, \\ & h_i(x_i, u_i) = 0, \quad i = 0, \dots, H-1. \end{aligned} \quad (10)$$

Let $c_i = [h_i^\top, g_i^\top]^\top$ collect constraint residuals and let $\mu_i = [\mu_{h,i}^\top, \mu_{g,i}^\top]^\top$ collect the corresponding multipliers. For fixed multipliers and penalties, the iLQR subproblem uses the augmented Lagrangian stage objective [13, 14],

$$L_i^{\text{AL}}(x_i, u_i, \mu_i, \rho_i) = \tilde{l}_i(x_i, u_i) + \mu_i^\top c_i(x_i, u_i) + \frac{1}{2} c_i(x_i, u_i)^\top I_{\rho,i} c_i(x_i, u_i). \quad (11)$$

The diagonal penalty matrix is $I_{\rho,i} = \text{diag}(\rho_{h,i}, m_i \odot \rho_{g,i})$, where $m_{ij} = 0$ when $g_{ij}(x_i, u_i) < 0$ and $\mu_{g,ij} = 0$, and $m_{ij} = 1$ otherwise. Thus, inactive inequality constraints do not contribute residuals, gradients, or Hessians to the local model.

ALM alternates between a primal and a dual step. In the primal step, iLQR minimizes $\sum_{i=0}^{H-1} L_i^{\text{AL}}$ with μ_i and $I_{\rho,i}$ fixed, using the rollout and line-search procedure of Section 4.1. After the inner solve returns the primal trajectory (x_i^*, u_i^*) , the dual step updates

$$\mu_{h,i}^+ = \mu_{h,i} + \rho_{h,i} h_i(x_i^*, u_i^*), \quad \mu_{g,i}^+ = \max(0, \mu_{g,i} + \rho_{g,i} g_i(x_i^*, u_i^*)). \quad (12)$$

For constraints that remain violated, the penalty is increased as $\rho_i^+ = \min(\Phi \rho_i, \rho_{\max})$ with $\Phi > 1$. When the multiplier update vanishes, the residuals approach zero, and stationarity of the augmented subproblem reduces to the KKT stationarity conditions of the original constrained problem [15].

4.3 Blended-iLQR

The constrained blended-MPC objective is obtained by substituting the ALM stage objective $L_i^{\text{AL}}(x_i, u_i, \mu_i, \rho_i)$ into the blended recursion of Section 3.1. Define the marginalized continuation value $V_{i+1}^\lambda(x_{i+1}) = \min_{u_{i+1}} Q_i^\lambda(x_{i+1}, u_{i+1})$. For fixed ALM multipliers and penalties, the constrained recursion is

$$\begin{aligned} Q_i^\lambda(x_i, u_i) &= (1 - \lambda) \hat{Q}(x_i, u_i) + \lambda (L_i^{\text{AL}}(x_i, u_i, \mu_i, \rho_i) + \gamma V_{i+1}^\lambda(x_{i+1})) \\ &= \underbrace{\lambda L_i^{\text{AL}}(x_i, u_i, \mu_i, \rho_i)}_{\tilde{L}_i^\lambda(x_i, u_i)} + (1 - \lambda) \hat{Q}(x_i, u_i) + \lambda \gamma V_{i+1}^\lambda(f(x_i, u_i)). \end{aligned} \quad (13)$$

At the last control step, the continuation is omitted and $Q_{H-1}^\lambda = \tilde{L}_{H-1}^\lambda$. Thus the blended AL-iLQR has the same dynamic-programming form as standard iLQR, but with blended ALM stage-objective $\tilde{L}_i^\lambda$ and effective discount $\lambda\gamma$. The backward pass therefore uses the same quadratic expansion as Eq. (8); only the derivatives of the local objective change. In particular, the constrained stage derivatives combine the ALM derivatives with the learned critic derivatives, e.g.,

$$\tilde{L}_{x,i}^\lambda = \lambda(L_i^{\text{AL}})_x + (1 - \lambda) \hat{Q}_x, \quad \tilde{L}_{u,i}^\lambda = \lambda(L_i^{\text{AL}})_u + (1 - \lambda) \hat{Q}_u, \quad (14)$$

and analogously for $\tilde{L}_{xx,i}^\lambda$, $\tilde{L}_{uu,i}^\lambda$, and $\tilde{L}_{ux,i}^\lambda$. The full expansions are given in Appendix B. The remaining recursion is unchanged, except that the next-state value contribution is scaled by $\lambda\gamma$.

5 Results

We evaluate the proposed framework in four complementary experiments. We begin with a runtime benchmark to verify our solver is amenable to the large-throughput requirement of reinforcement learning. We test the solver on randomly generated optimal-control problems to characterize how our implementation scales with batch size, hardware, and the method of derivative computation. Next, we isolate the effect of the critic blend on a point-mass task, where the optimal critic is available in closed form. We then study a quadrotor task with a biased dynamics model and an unmodeled suspended payload, which tests whether joint residual-dynamics learning and critic learning can compensate for model bias, capture missing state couplings, and remain compatible with quaternion kinematics. Finally, we consider Franka cube stacking as a contact-rich manipulation task to assess whether the framework can plan over incompletely modeled contact interactions and benefit from visual and haptic information for grasp detection.

Speed Benchmarking. The experimental setup and detailed timing curves are given in Appendix C. The main result is that GPU execution becomes decisive once learned dynamics and critic models are part of the solve: on the tested hardware, batched GPU execution is roughly one to two orders of magnitude faster than CPU execution over the relevant batch-size range. In the purely non-learning setting, CPU execution is competitive only for the smallest batches, after which GPU execution overtakes it as batching exposes more parallel work. Figure 4 further shows that the amortized runtime per problem decreases strongly up to about 10^4 parallel QPs before saturating, and that automatic differentiation remains close to analytical derivatives in this benchmark.

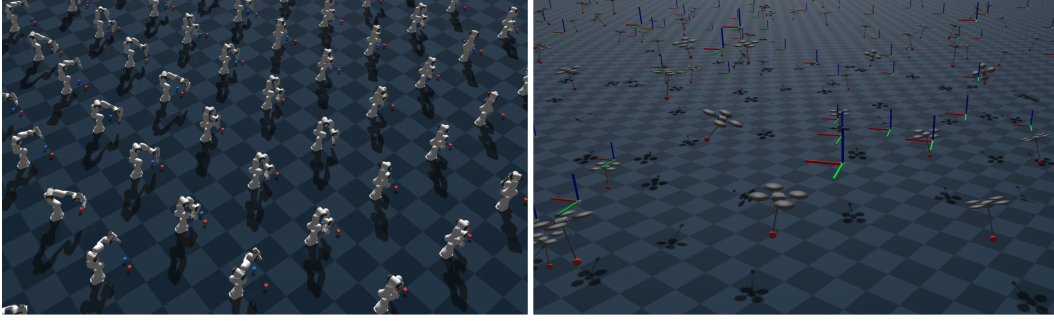


Figure 1: Parallel environments used for training and evaluation. Left: The stack-cube task with a 6-dof manipulator. The robot must pick up the target cube (blue) and stack it on top of the goal cube (red). Right: The quadrotor task, where the drone must hover at a target position while carrying an unmodeled, suspended payload. Both tasks are simulated using Genesis [16].

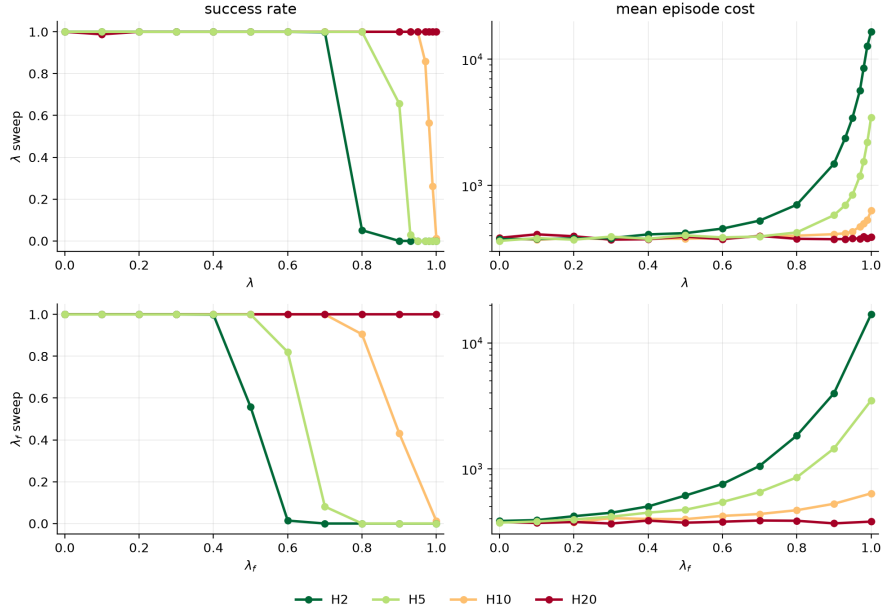


Figure 2: Point-mass reach task with the analytical LQR critic Q^* , V^* . Columns: success rate and mean episode cost. Top: all-stage blend, $\lambda = \lambda_f$ swept jointly. Bottom: terminal-only blend, $\lambda = 1$ and λ_f swept.

Isolating the Effect of the Critic Blend. To separate the effect of the blend from critic approximation errors, a force-controlled 2-D point mass must reach a fixed goal position and come to rest. For this linear-quadratic task, the optimal infinite-horizon discounted Q^* and V^* follow in closed form from the discrete algebraic Riccati equation and take the place of the learned critic (Appendix D). We sweep the blend for horizons $H \in \{2, 5, 10, 20\}$ in two settings: blending Q^* at all stages with $\lambda_f = \lambda$, and blending only the terminal value with λ_f , while $\lambda = 1$ at all other stages. Figure 2 shows the results. Without the critic ($\lambda = \lambda_f = 1$), MPC overshoots the goal for $H \in \{2, 5, 10\}$ and fails the task, whereas $H = 20$ succeeds. Lowering λ_f towards zero recovers the success rate and cost of the optimal policy at all horizons. Blending at all stages reaches the same performance already at values of λ closer to 1. We interpret this as the stage-wise critic terms supplying long-range information along the whole planned trajectory, in addition to the terminal value. These results rely on the exact Q^* ; Section 6 discusses why learned critics restrict λ to values close to 1.

Overcoming Biased Dynamics and Unmodeled State-Dependencies. In this experiment, a quadrotor starts in a random pose and must hover to a target position while carrying an unmodeled payload suspended by a rod. With all learning components disabled, the nominal MPC baseline reaches a success rate of only 0.05 and a mean cost of $2.6e4$, with most failures caused by crashes. The learned critic variant, which combines residual-dynamics learning with the blended critic objective, raises the success rate to 0.79 and lowers the mean cost to $4.7e3$. As an adaptive-control baseline, an idealized incremental nonlinear dynamic inversion (INDI) controller [17] without sensor noise or delay reaches a higher success rate of 0.94, but a higher mean cost of $5.6e3$, since it overshoots more and settles more slowly around the goal (Appendix E.1). In our setting, learning the dynamics model alone is not sufficient to solve the task reliably. We attribute this to the short look-ahead of 1s for the MPC, not capturing the fatal swing-up motion of the pendulum early enough, and the approximation bias of the learned dynamics model, requiring careful alignment of planning frequency, integration scheme, and look-ahead distance to trade off approximation bias with real-time-ness. However, we observe that combining the learned dynamics model with the learned critic, which injects infinite-horizon MDP information into the MPC, yields more conservative approaches towards the goal, keeping the oscillations bounded and avoiding crashing. The learned critic can drive the system away from high action-value regions that lead to failure, including the ones with high pendulum deflection. This observation is broadly in line with the remarkable success of model-free policy-gradient methods, which achieve high task performance not by meticulously modeling the underlying dynamics but by directly learning mitigation strategies that avoid task failure [18]. To further demonstrate the sample-efficiency claim, we establish a PPO baseline [19] with the hyperparameters of Table 6 reaching comparable asymptotic performance but requiring training for 1.5 hours on $1e4$ parallel environments, collecting roughly $1e8$ environment transitions (collected at 200 Hz, the policy frequency), whereas our learned-critic achieves the reported performance after roughly 30 minutes using 10^5 transitions (collected at 20 Hz, the planner frequency, cf. Appendix G). Figure 3 shows how this improvement emerges during training: as λ is decreased, the learned critic gradually takes over a larger share of the objective, correcting the nominal behavior and improving both success rate and cost. Detailed task and training settings are given in Appendix E.1, whereas empirical evidence for the efficacy of our approach and further insights into the learned critic are given in Appendix E.2.

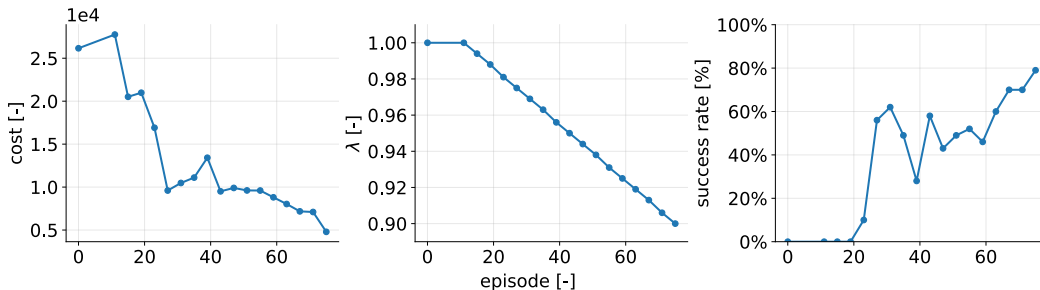


Figure 3: Quadrotor learning progress while decreasing the critic blend parameter λ . Early updates behave like nominal MPC and fail under the biased incomplete model (cf. Section 3.1); once the learned critic is allowed to shape the objective more strongly, the success rate increases and the cost drops.

Learning to Plan over Contacts Using Visual and Haptic Information. In this experiment, a 6-DOF manipulator must pick up a cube and stack it precisely on top of another cube using only a simple kinematic robot model and no explicit contact model. Unsurprisingly, a naive grasp-uninformed nominal MPC with a single global cost function fails completely (success rate of 0, stuck in pre-grasp pose) since it cannot reason over the grasp. Following Appendix F.1, we add to the state vector the cube’s position and a grasp-score, which is initialized from an external grasp detection using tactile sensors, and replace the global cost with a multi-phase cost function. This modification

raises the nominal baseline to 0.361 success rate. Further learning a residual to the grasp-score state lifts the success rate to moderate 0.401. Employing a learned critic, which is conditioned on visual depth and haptic observations (cf. Figure 9) improves task performance significantly to 0.94 success rate. We observe more delicate and better-informed recovery behaviors than with nominal MPC, including corrections after failed grasp attempts and poor approaches that tip over the target cube. More insights into the task setup and further discussions of the results are given in Appendix F.1 and Appendix F.2.

6 Limitations

A first limitation is that the solver relies on first- and second-order derivatives of the learned residual and critic models, and optimization remains sensitive to irregular derivatives. This restricts the blend: with the exact critic of Section 5, lower values of λ consistently recover the performance of the optimal policy, whereas with learned critics only modest reductions from $\lambda = 1$, roughly in the range 0.98 to 0.9, outperformed the nominal baselines in our experiments. Since the weight of the nominal rollout decays as $(\gamma\lambda)^k$ along the horizon (Eq. (3)), low λ removes the model information that would compensate imperfect critic derivatives, particularly at long horizons (cf. Appendix F.2). We therefore start training at $\lambda = 1$, lower λ only once the critic has acquired sufficient state-space coverage, and mitigate irregular derivatives by disagreement-based attenuation of the critic, smoothing, and clipping. Nevertheless, poor gradients can still steer the optimizer toward uninformative regions of the state space, degrade data collection early-on, and ultimately stall training; stronger training-time regularization, efficient off-policy target construction and priority-sampling from the replay-buffer should therefore be investigated. A second limitation concerns real-time deployment on a single system. While all solver computations are batched across environments, and costs and derivatives are evaluated in batch over all time steps, the forward rollout, the Riccati recursion, and the line search remain sequential over the horizon, which precludes replanning at rates above 100 Hz in the current implementation. However, iLQR returns the time-indexed local feedback policy $\hat{u}_i = u_i + K_i dx(\hat{x}_i, x_i)$ around the planned trajectory, so the planner can run at a low rate while this policy closes the loop at the control rate.

7 Conclusion

To conclude, this paper presented a learning-augmented constrained MPC framework for planning under model mismatch. While dynamics learning enabled the planner to account for directly observable but incompletely modeled state variables, the main driver of the observed performance gains was the integration of a learned critic into the gradient-based optimization framework. By providing corrective descent directions during the backward recursion, the critic injected long-horizon behavioral information that is inaccessible to a finite-horizon MPC objective alone. In this way, the method mitigated the inherent myopia of MPC by leveraging value information from the underlying infinite-horizon MDP. Across the reported runtime and control experiments, the framework scaled to thousands of parallel optimal control problems and improved performance under biased and incomplete dynamics models. Future work will focus on addressing the aforementioned limitations and on stronger critic pre-training, including representation learning from video demonstrations.

A Bellman-Residual Form of the Blended Objective

We have H control steps, hence the planned rollout contains the state-action pairs $(x_i, u_i)_{i=0}^{H-1}$ and terminal state x_H . For compactness, write $Q_i = Q(x_i, u_i)$, $\hat{Q}_i = \hat{Q}(x_i, u_i)$, and $l_i = l(x_i, u_i)$.

For $i = 0, \dots, H-2$, the blended recursion in the main text reads

$$Q_i^\lambda = (1 - \lambda)\hat{Q}_i + \lambda(l_i + \gamma Q_{i+1}^\lambda),$$

where $\hat{Q}_i$ is the learned Q-estimate and Q_i^λ is the MPC-blended Q estimate at timestep i . At the last control step we close the recursion with

$$Q_{H-1}^\lambda = (1 - \lambda)\hat{Q}_{H-1} + \lambda l_{H-1}.$$

To expose the relation to temporal-difference style corrections, define the deviation of the blended objective from the learned critic by

$$\delta_i = Q_i^\lambda - \hat{Q}_i.$$

Subtracting $\hat{Q}_i$ from both sides of the recursion gives

$$\delta_i = (1 - \lambda)\hat{Q}_i + \lambda(l_i + \gamma Q_{i+1}^\lambda) - \hat{Q}_i = \lambda(l_i + \gamma Q_{i+1}^\lambda - \hat{Q}_i).$$

Now add and subtract $\gamma\hat{Q}_{i+1}$ inside the parentheses:

$$\delta_i = \lambda(l_i + \gamma\hat{Q}_{i+1} - \hat{Q}_i + \gamma(Q_{i+1}^\lambda - \hat{Q}_{i+1})).$$

This separates the local Bellman residual of the learned critic from the downstream blended correction. Define the one-step Bellman residual

$$B_i = l_i + \gamma\hat{Q}_{i+1} - \hat{Q}_i.$$

Using $\delta_{i+1} = Q_{i+1}^\lambda - \hat{Q}_{i+1}$, we obtain the recursion

$$\delta_i = \lambda(B_i + \gamma\delta_{i+1}),$$

At the last control step, the terminal approximation gives

$$\delta_{H-1} = Q_{H-1}^\lambda - \hat{Q}_{H-1} = \lambda(l_{H-1} - \hat{Q}_{H-1}).$$

For any intermediate step we get

$$\delta_i = \lambda \left(\sum_{k=0}^{H-i-2} (\lambda\gamma)^k B_{i+k} + (\lambda\gamma)^{H-i-1} \delta_{H-1} \right).$$

Setting $i = 0$ and substituting back $Q_0^\lambda = \hat{Q}_0 + \delta_0$ yields

$$Q_0^\lambda = \hat{Q}_0 + \lambda \left(\sum_{k=0}^{H-2} (\lambda\gamma)^k B_k + (\lambda\gamma)^{H-1} (l_{H-1} - \hat{Q}_{H-1}) \right).$$

Hence, the blended objective can be interpreted as the learned critic plus a geometrically weighted accumulation of local Bellman-residual corrections along the planned trajectory. In particular, when the learned critic satisfies the Bellman relation well along the planned rollout, the residual terms B_i are small and the blended objective stays close to $\hat{Q}_0$; when the local model-based rollout disagrees with the learned critic, the residual terms accumulate and correct the objective accordingly.

B Quadratic Q Expansion for blended ALM-iLQR

For the ALM-iLQR the gradients and Hessians for the quadratic Q expansion in each timestep read:

$$\begin{aligned} Q_x &= \tilde{L}_x + f_x^\top V'_x, \\ Q_u &= \tilde{L}_u + f_u^\top V'_u, \\ Q_{xx} &= \tilde{L}_{xx} + f_x^\top V'_{xx} f_x + V'_x \cdot f_{xx}, \\ Q_{uu} &= \tilde{L}_{uu} + f_u^\top V'_{uu} f_u + V'_u \cdot f_{uu}, \\ Q_{ux} &= \tilde{L}_{ux} + f_u^\top V'_{xx} f_x + V'_x \cdot f_{ux}. \end{aligned}$$

with the learned Q-function terms, and using a locally-linear approximation of the constraint function $c_i \approx c_i^0 + c_{x,i}\delta x_i + c_{u,i}\delta u_i$,

$$\begin{aligned}\tilde{L}_x &= \lambda(L_i^{\text{AL}})_x + (1 - \lambda)\hat{Q}_x, & (L_i^{\text{AL}})_x &= l_x + c_{x,i}^\top(\mu_i + I_{\rho,i}c_i), \\ \tilde{L}_u &= \lambda(L_i^{\text{AL}})_u + (1 - \lambda)\hat{Q}_u, & (L_i^{\text{AL}})_u &= l_u + c_{u,i}^\top(\mu_i + I_{\rho,i}c_i), \\ \tilde{L}_{xx} &= \lambda(L_i^{\text{AL}})_{xx} + (1 - \lambda)\hat{Q}_{xx}, & (L_i^{\text{AL}})_{xx} &\approx l_{xx} + c_{x,i}^\top I_{\rho,i}c_{x,i}, \\ \tilde{L}_{uu} &= \lambda(L_i^{\text{AL}})_{uu} + (1 - \lambda)\hat{Q}_{uu}, & (L_i^{\text{AL}})_{uu} &\approx l_{uu} + c_{u,i}^\top I_{\rho,i}c_{u,i}, \\ \tilde{L}_{ux} &= \lambda(L_i^{\text{AL}})_{ux} + (1 - \lambda)\hat{Q}_{ux}, & (L_i^{\text{AL}})_{ux} &\approx l_{ux} + c_{u,i}^\top I_{\rho,i}c_{x,i}.\end{aligned}$$

the next-state value-function gradient V'_x and hessian V'_{xx} are carried over in the recursion.

C Runtime Scaling Benchmark

To isolate solver throughput, we benchmark the batched iLQR implementation on random control instances with an 8-dimensional state, a 4-dimensional action, a planning horizon of 32 steps, and a planning time step of 0.05 s. The learned dynamics model and learned critic are fully connected MLPs with three hidden layers of width 256 and ELU activations. Each timing run measures a single iLQR iteration, comprising one forward rollout, one backward pass, up to six regularization trials, and up to eight line-search evaluations. We discard two warmup runs and report averages over 20 measured runs. The benchmark is executed on an NVIDIA RTX 5080 mobile GPU (175 W) and an AMD Ryzen 9 9955HX CPU with 64 GB RAM.

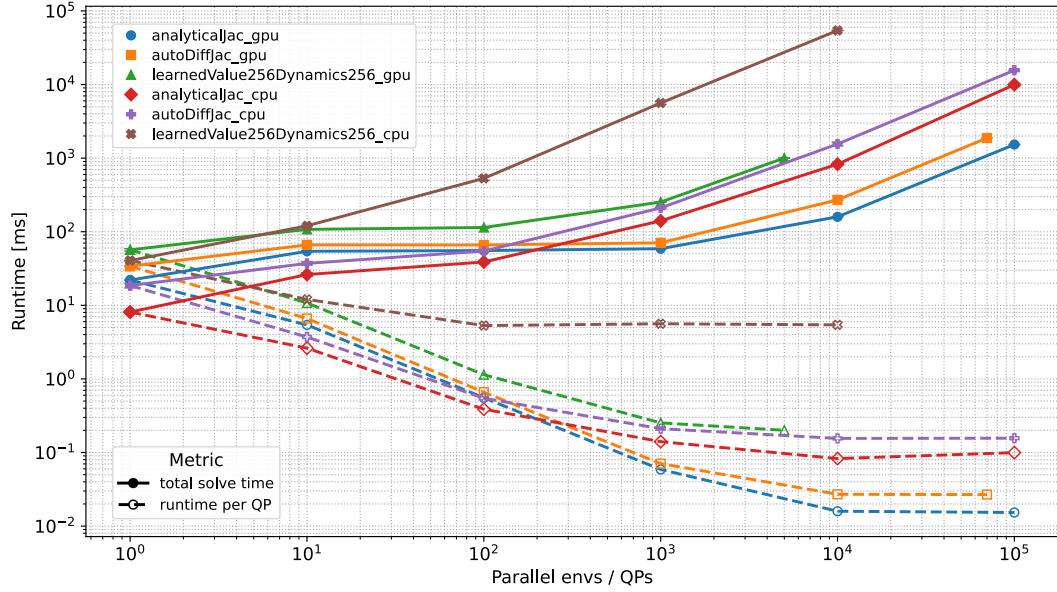


Figure 4: Runtime scaling of one batched iLQR iteration with analytical and autodiff derivatives on CPU and GPU. GPU execution becomes decisive once learned models are part of the solve, while autodiff remains close to analytical derivatives; runtime per QP decreases with batching up to about 10^4 parallel problems before flattening on the tested hardware.

The figure separates total solve time from amortized runtime per QP. For the learned-model setting, the GPU curves stay well below the CPU curves across the relevant batch sizes, yielding roughly one to two orders of magnitude lower runtime on the tested hardware. For the analytical setting, CPU execution is competitive only at the smallest batches, after which GPU execution overtakes it as batching exposes more parallel work. The amortized runtime decreases strongly up to about 10^4

parallel QPs and then saturates, which in our setup coincides with the practical hardware limit of the tested mobile GPU.

D Point-Mass Experiment Details

The point mass has mass $m = 2$ kg, state $x = [p^\top, v^\top]^\top \in \mathbb{R}^4$ with planar position p and velocity v , and is driven by a planar force $u \in [-20, 20]^2$ N with $\dot{p} = v$ and $\dot{v} = u/m$. The stage cost is $\ell(x, u) = \beta_p \|p - p^*\|_2^2 + \beta_v \|v\|_2^2 + \beta_u \|u\|_2^2$ with $\beta_p = 1$, $\beta_v = 0.05$, and $\beta_u = 0.1$. Linearizing the discretized dynamics at $\Delta t_{\text{plan}} = 0.1$ s gives (A, B) , and the discounted algebraic Riccati equation with $\gamma = 0.95$ gives P , so that $V^*(x) = e^\top P e$ and $Q^*(x, u) = \ell(x, u) + \gamma V^*(Ae + Bu)$ in the error coordinates $e = x - [p^*, 0]^\top$. An episode of 10 s counts as successful once the position error falls below 0.1 m and the speed below 0.05 m/s. Each sweep runs 10^3 parallel environments and holds each blend value for one episode, $\lambda \in \{1, 0.99, 0.98, 0.97, 0.95, 0.93, 0.9, 0.8, \dots, 0\}$ for the all-stage sweep and $\lambda_f \in \{1, 0.9, \dots, 0\}$ for the terminal-only sweep.

E Quadrotor Experiment Details

E.1 Task Setup

Dynamics Model. The nominal MPC model uses the rigid-body state and control given by

$$x = \begin{bmatrix} p_{WB}^W \\ q_{WB}^W \\ v_{WB}^W \\ \omega_{WB}^B \end{bmatrix}, \quad u = \begin{bmatrix} a_z^B \\ \alpha_{WB}^B \end{bmatrix},$$

The virtual controls are subject to componentwise box constraints,

$$u_{\min} \leq u \leq u_{\max}, \quad a_z^B \in [-20, 40] \text{ m s}^{-2}, \quad \alpha_{WB}^B \in [-40, 40]^3 \text{ rad s}^{-2},$$

equivalently written in the inequality-constraint form as $h_{\min}(u) = u_{\min} - u \leq 0$ and $h_{\max}(u) = u - u_{\max} \leq 0$. For the quaternion block, the solver uses the reduced local coordinates and projection matrix

$$x_{\text{red}} = \begin{bmatrix} p_{WB}^W \\ \phi_{WB}^W \\ v_{WB}^W \\ \omega_{WB}^B \end{bmatrix}, \quad E = \begin{bmatrix} I & 0 & 0 & 0 \\ 0 & G(q_{WB}^W) & 0 & 0 \\ 0 & 0 & I & 0 \\ 0 & 0 & 0 & I \end{bmatrix},$$

so the quaternion state is optimized in a three-dimensional tangent coordinate $\phi_{WB} \in \mathbb{R}^3$ rather than directly in $q_{WB} \in S^3$. The dynamics are

$$\dot{x} = f(x, u) = \begin{bmatrix} v_{WB}^W \\ \frac{1}{2} q_{WB}^W \otimes \begin{bmatrix} 0 \\ \omega_{WB}^B \end{bmatrix} \\ g^W + R(q_{WB}^W) \begin{bmatrix} 0 \\ 0 \\ a_z^B \end{bmatrix} \\ \alpha_{WB}^B \end{bmatrix}.$$

where $g^W = \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix}$ and $R(q_{WB}^W)$ maps body-frame accelerations into the world frame. To inject structured model mismatch, the simulator additionally exposes a swing-load state with roll- and pitch pendulum coordinates,

$$x^{\text{aug}} = \begin{bmatrix} x \\ \theta \\ \dot{\theta} \end{bmatrix}, \quad \theta \in \mathbb{R}^2,$$

corresponding to a 0.172 kg suspended load (0.15 kg payload, 0.02 kg rod, 0.002 kg bearing) attached to a 0.15 m rod. These state variables are observed during training but are not modeled in

the nominal rigid-body dynamics above; their effect on the translational and rotational motion must therefore be captured from data alone.

The model bias $\bar{m} = m/1.5$ and $\bar{J}_B = 1.5J_B$ is introduced in the the actuator model

$$\begin{aligned}\dot{v}_{WB}^W &= g^W + \bar{m}^{-1}R(q_{WB})f_B(\omega^2), \\ \dot{\omega}_{WB}^B &= \bar{J}_B^{-1}(\tau_B(\omega^2) - \omega_{WB}^B \times \bar{J}_B\omega_{WB}^B),\end{aligned}$$

which computes the the squared rotor-speed commands ω_i^2 from the virtual controls u using the wrench model

$$f_B(\omega^2) = \begin{bmatrix} 0 \\ k_f \sum_{j=1}^4 \omega_j^2 \end{bmatrix}, \quad \tau_B(\omega^2) = \sum_{j=1}^4 r_j \times \begin{bmatrix} 0 \\ k_f \omega_j^2 \end{bmatrix} + \begin{bmatrix} 0 \\ k_d \sum_{j=1}^4 s_j \omega_j^2 \end{bmatrix},$$

with k_f and k_d the force and drag coefficients, r_j the rotor-arm vectors, and $s_j \in \{-1, +1\}$ the propeller spin directions. Hence the benchmark combines parameter bias on the modeled rigid-body states with additional unmodeled state dependencies from the swing-load coordinates.

Cost Function The quadrotor stage cost is a quadratic hover-tracking objective, where $(\cdot)^*$ denotes the hover goal state,

$$\begin{aligned}\ell(x, u) &= \frac{1}{2} \left(\beta_p \|p_{WB}^W - p_{WB}^{W,*}\|_2^2 + \beta_q \|\text{rotvec}(q_{WB}^* \otimes q_{WB}^{-1})\|_2^2 \right. \\ &\quad + \beta_v \|v_{WB}^W - v_{WB}^{W,*}\|_2^2 + \beta_\omega \|\omega_{WB}^B - \omega_{WB}^{B,*}\|_2^2 \\ &\quad \left. + \beta_u \|u\|_2^2 \right),\end{aligned}$$

where $\beta_p, \beta_q, \beta_v, \beta_\omega$, and β_u are positive cost weights.

INDI Baseline. INDI [17] computes control increments from measured accelerations and thereby compensates unmodeled forces and moments, including those of the suspended payload, without an explicit model of them. We evaluate an idealized variant, in which these measurements are free of noise and delay, on the same task and with the same cost function as the MPC variants.

E.2 Experimental Insights

Failure mode of nominal MPC. Figure 5 compares representative closed-loop trajectories of nominal MPC and the blended-critic controller under the biased incomplete model (random seeds). The nominal controller initially reduces the position error, but it does so with the wrong inertial model and without explicit payload dynamics. This injects excessive rotational motion into the suspended load, which amplifies and destabilizes the quadrotor. Around the end of the episode, the nominal controller tips over, loses altitude and crashes. The learned critic avoids this failure: by learning to favor a lower-velocity approach, it keeps the pendulum motion bounded, avoiding catastrophic tip-over even under presence of severe cross-couplings.

Monte Carlo analysis. The single-rollout picture is confirmed by the Monte Carlo envelopes in Figure 6, computed from $N = 300$ randomized closed-loop runs. Under the learned controller, the mean trajectories converge toward hover and the state dispersion contracts after the initial transient. Under nominal MPC, the dispersion grows late in the episode as the quadrotor tips over and crashes during breaking. The nominal mean also develops a clear downward drift in z . This is the aggregate version of the same failure mode seen in Figure 5.

Local structure of the learned critic. Figures 7 and 8 show local x - y slices of the learned critic. Each slice is generated from one state-action sample taken during MPC optimization: only the quadrotor position coordinates p_x and p_y are swept, while the remaining state and action entries are fixed. The plots therefore describe the value density around this specific rollout condition; they

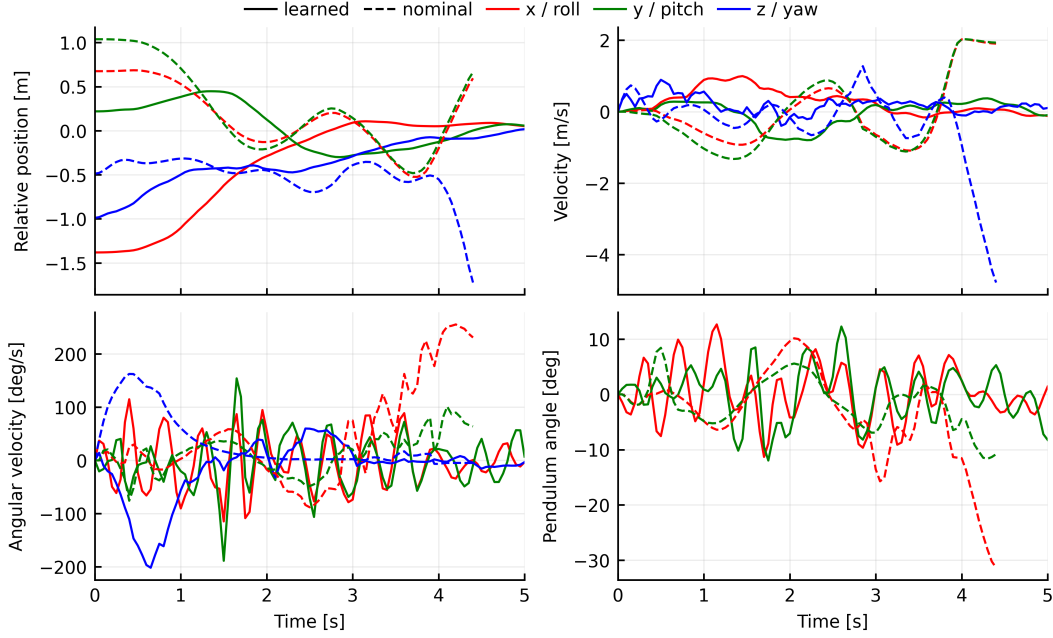


Figure 5: Two closed-loop trajectories under the biased incomplete quadrotor model. Solid lines denote the trajectories stabilized by the learned critic controller; dashed lines denote the nominal MPC. Different random initial states are shown for the two controllers. The nominal controller initially moves toward the goal but excites the suspended load, which then lead to tip-over late into the flight. The learned controller approaches the goal more conservatively, keeping the pendulum motion bounded and stabilizing the system throughout the episode.

are not global projections of the learned Q function, which is fitted over the full 21-dimensional state-action input. Figure 7 shows that the four value heads converge to almost identical spatial value distributions in this slice. The critic is therefore certain about the correction signal used at this rollout state.

Figure 8 compares this learned correction to the nominal stage cost. The critic remains consistent with the stage cost in the important failure case: it does not create spurious low-value basins far away from the goal and remains locally smooth. At the same time, the value minimum is shifted relative to the instantaneous stage-cost minimum. For the fixed velocity, attitude, and payload state in this snapshot, positions closer to the goal have higher predicted continuation cost; the same non-position states would be preferable farther back along the approach direction. This is the correction to account for the increased momentum the nominal model cannot express. The critic surface is smooth over the plotted region, which is required for stable iLQR derivatives. The final panel shows the uncertainty-based scaling derived from value-head variance, which is used to weight the critic against the nominal MPC objective (cf. Section 3.2).

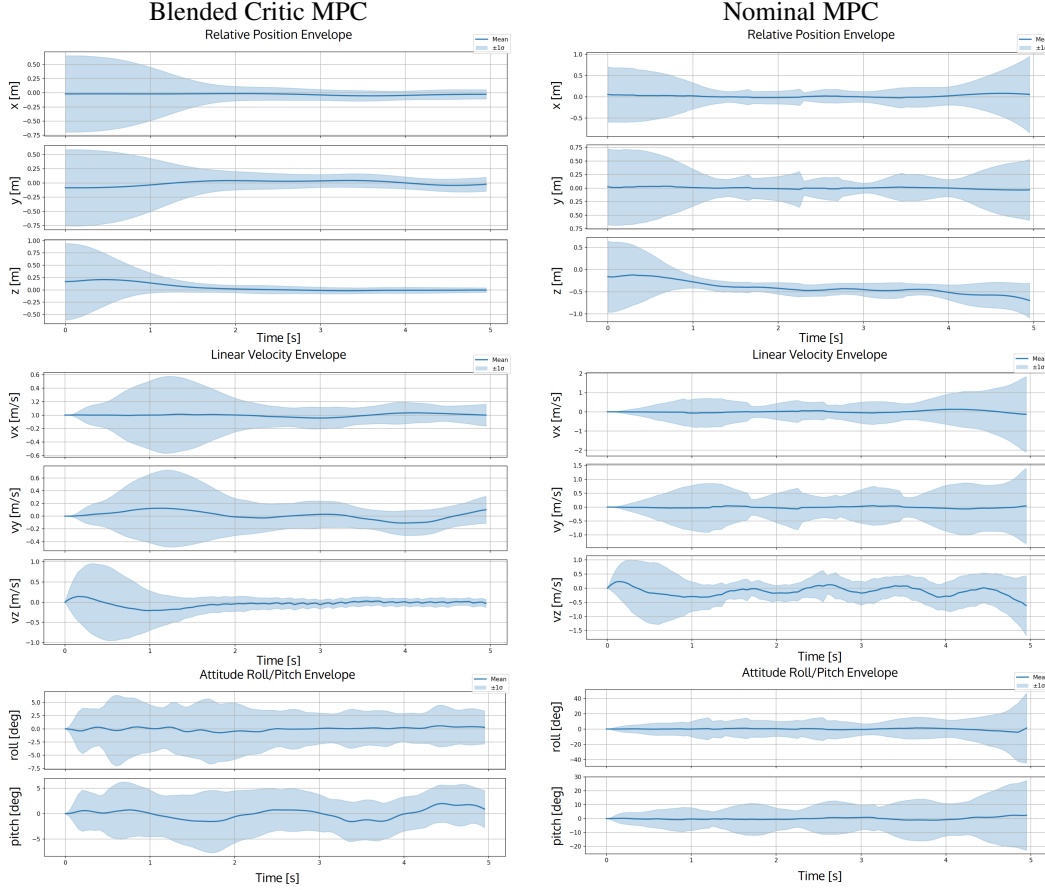


Figure 6: Monte Carlo envelopes over $N = 300$ randomized rollouts. Left: blended-critic MPC. Right: nominal MPC. The learned controller keeps the mean close to the hover target and prevents late-episode variance growth. The nominal controller develops large dispersion and a systematic loss of altitude, consistent with repeated crash trajectories.

F Cube Stacking Experiment Details

F.1 Task Setup

Dynamics Model. The manipulation benchmark asks the robot to pick up a target cube and place it on top of a support cube. The planner uses the reduced state and control

$$x = \begin{bmatrix} q_{\text{arm}} \\ w_g \\ p_{WT}^W \\ g \end{bmatrix}, \quad u = \begin{bmatrix} \dot{q}_{\text{arm}} \\ \dot{w}_g \end{bmatrix}, \quad \dot{x} = f(x, u) = \begin{bmatrix} \dot{q}_{\text{arm}} \\ \dot{w}_g \\ g v_{WE}^W(q_{\text{arm}}, \dot{q}_{\text{arm}}) \\ 0 \end{bmatrix},$$

where q_{arm} are arm joints, w_g is gripper width, p_{WT}^W is target-cube position, and $g \in [0, 1]$ is a learned continuous grasp gate. Thus the nominal model transports the cube kinematically only in proportion to the current grasp estimate: for $g \approx 0$ the target cube is effectively decoupled from the end effector, whereas for $g \approx 1$ it follows the commanded end-effector motion.

The end-effector translational velocity is computed from differentiable forward kinematics of the manipulator. Let $p_{WE}^W = \text{FK}_E(q_{\text{arm}})$ denote the end-effector position obtained from the robot's batched forward-kinematics model. The corresponding geometric Jacobian is

$$J_E^W(q_{\text{arm}}) = \frac{\partial p_{WE}^W}{\partial q_{\text{arm}}}, \quad v_{WE}^W(q_{\text{arm}}, \dot{q}_{\text{arm}}) = J_E^W(q_{\text{arm}}) \dot{q}_{\text{arm}}.$$

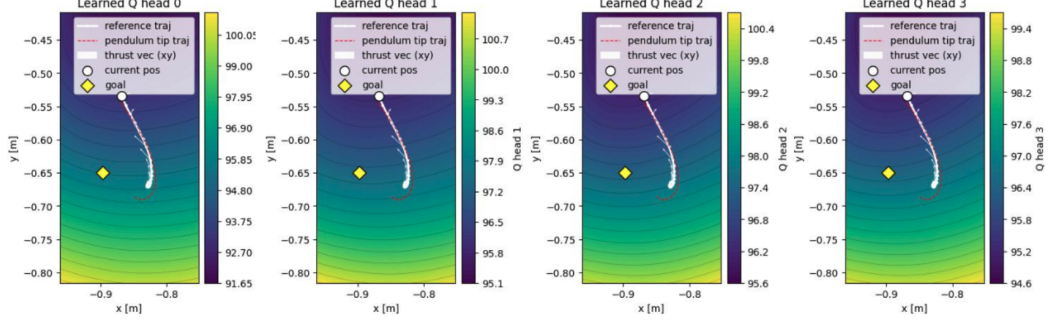


Figure 7: Local x - y slices of the four learned value heads at one MPC optimization snapshot. All state and action entries except planar position are held fixed. The heads agree on the spatial value distribution, indicating low ensemble uncertainty for the critic correction at this rollout state.

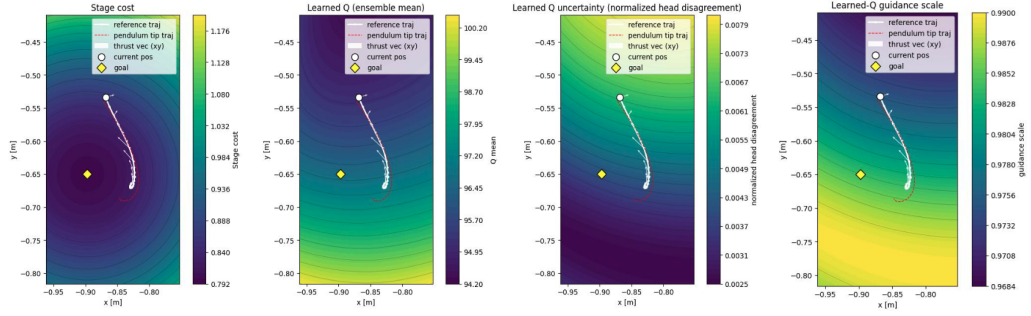


Figure 8: Local critic analysis for the same slice as Figure 7. The learned value remains consistent with the stage cost by avoiding low-value regions far from the goal, but shifts the local optimum according to the fixed velocity, attitude, and payload state. The critic is smooth in the plotted region, and the value-head variance defines the guidance scale used to blend the critic with the nominal MPC objective.

In the implementation, the forward kinematics and geometric Jacobians are evaluated in batch across all environments and are fully differentiable. Contact dynamics are not modeled explicitly; instead, the missing interaction effects must be absorbed by the learned residual and the learned value function.

Cost Function. Let p_{WA}^W denote the support-cube center and define the desired stacked target position $p_{WG}^W = p_{WA}^W + [0, 0, 1.2s_z]^\top$, with s_x and s_z the cube width and height. We implement a multi-phase cost function, where the pregrasp-term and the transport term are weighted by the grasping gate

$$\begin{aligned} \ell(x, u) = & \beta_p \|p_{WT}^W - p_{WE}^W\|_2 + \beta_p \text{clip}(1.8s_x - p_{WT,z}^W, 0, 1.8s_x) \\ & + g \beta_p \|p_{WG}^W - p_{WE}^W\|_2 + (1 - g) \text{sg}[\beta_p \|p_{WG}^W - p_{WE}^W\|_2] \\ & - \beta_g \left(\rho(p_{WT}^W) \dot{w}_g - (1 - \rho(p_{WT}^W)) g \dot{w}_g \right) + \beta_u \|\dot{q}_{\text{arm}}\|_2^2 + \beta_g \dot{w}_g^2, \end{aligned}$$

where β_p , β_g , and β_u are positive cost weights, $\rho(p_{WT}^W) = \exp(-\|p_{WG}^W - p_{WT}^W\|_2 / (0.4s_x))$, and $\text{sg}[\cdot]$ denotes the stop-gradient evaluation used in the implementation.

Preprocessed Inputs. The residual dynamics model receives the feature vector

$$\phi_f = [x^\top \quad u^\top \quad (f_L^W)^\top \quad (f_R^W)^\top \quad g \quad z_{\text{img}}^\top]^\top,$$

Table 1: Influence of iLQR horizon length on cube-stacking performance at $\Delta t_{\text{ctrl}} = 0.05$ s. Nominal MPC requires an intermediate horizon, whereas the learned critic maintains strong performance even for short horizons.

H	Nominal SR	Nominal cost	Critic SR	Critic cost
2	0.00	–	0.03	278
5	0.116	280	0.685	266
10	0.318	254	0.885	223
20	0.361	251	0.941	220
30	0.298	253	0.018	335
50	0.0	411	0.02	421

Table 2: Influence of critic input features on cube-stacking performance. Adding depth and haptic observations improves success rate monotonically, indicating that richer observations improve grasp-phase inference and critic stability.

Critic input	SR	Mean cost
$[x, u, p_{\text{goal}}]$	0.230	238.0
$[x, u, p_{\text{goal}}, \text{depth}]$	0.543	234.0
$[x, u, p_{\text{goal}}, \text{depth}, \text{haptic}]$	0.941	257.0

that is, the full state, control, left and right finger contact forces, the scalar grasp score, and optional image features z_{img} . The value model receives

$$\phi_Q = [(p_{WG}^W)^\top \quad x^\top \quad u^\top \quad (f_L^W)^\top \quad (f_R^W)^\top \quad z_{\text{img}}^\top]^\top,$$

so it additionally gets the goal location but does not take the grasp score as a separate input in the current implementation.

F.2 Experimental Insights

The ablations in Tables 1, 2, and 3 support the findings summarized in Section 5. On this task, the main benefit comes from critic learning with informative observations, whereas residual dynamics learning yields only moderate improvement.

Effect of planning horizon. Table 1 shows that nominal MPC works best with a medium length optimization horizon. Very short horizons do not result in a sufficiently strong grasp, keeping the robot stuck in the approach phase, while very long horizons degrade due to unsteadiness in the iLQR backward pass when optimizing over the grasp. The learned critic improves success across the full horizon range and already reaches 0.685 success rate at $H = 5$, where nominal MPC barely succeeds. This supports the main-text claim that the critic gives iLQR consistently a descent direction aligned with the long-horizon stacking objective, leading to more directed behavior and more robust grasping. The learned critic also degrades at long horizons; unlike the exact critic of Section 5, it supplies imperfect gradients and curvature at every stage, while the weight of the nominal rollout decays as $(\gamma\lambda)^k$, and both effects can destabilize the backward pass over long horizons (cf. Section 6).

Effect of visual and haptic inputs. Table 2 shows a consistent gain when depth and haptic observations are added to the critic input. Using only state, action, and goal features yields a success rate of 0.23. Adding depth raises this to 0.543, and adding haptic information further raises it to 0.941. Figure 9 illustrates the transport phase together with the predicted cube and end-effector trajectories available to the controller. This matches the main-text result: visual and haptic observations stabilize the critic by reducing ambiguity about grasp quality and task phase, which is the dominant source of uncertainty in this benchmark.

Effect of residual dynamics learning. Table 3 shows only minor benefit from learning residual dynamics on this task. The learned variants remain close to or below the baseline success rate and

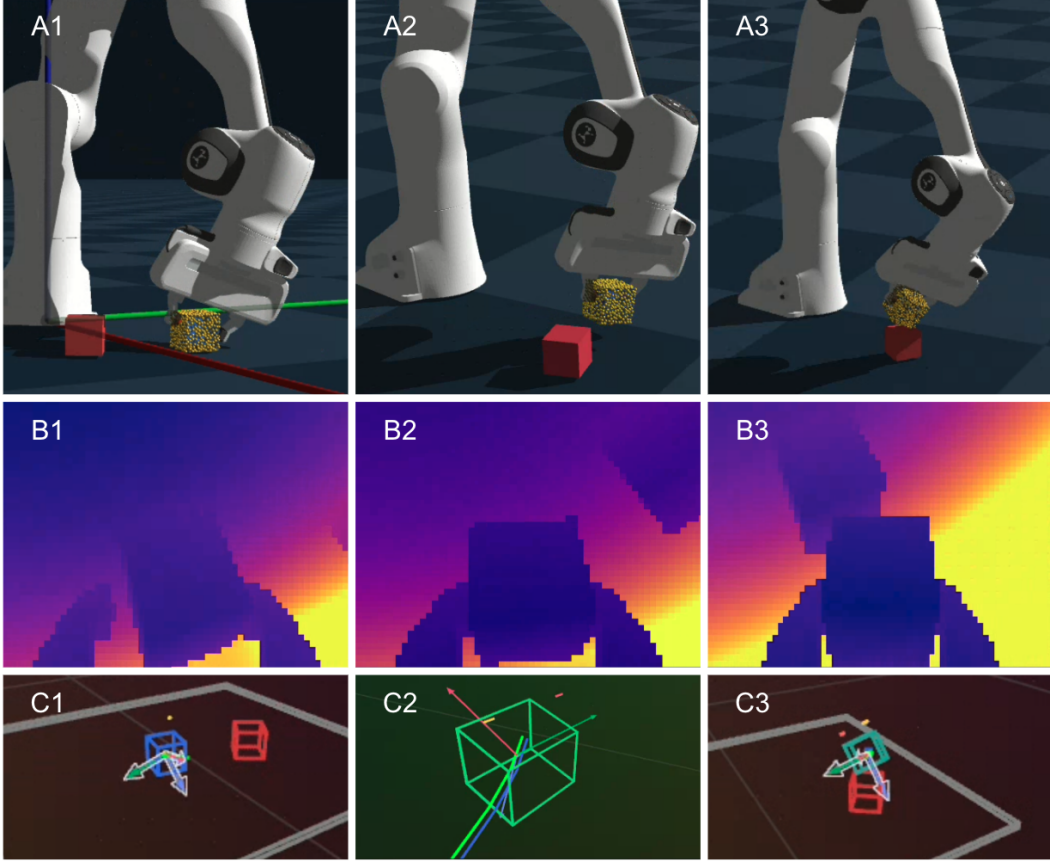


Figure 9: Cube-stacking phases. (1) Approach and Grasp; (2) Pickup and Transport; (3) Align and Drop; Depth (B) and haptic observations (contact-probes in gripper fingers visualized by red markers, figure A1..3) improve grasp estimation. The predicted cube trajectory (C2, blue) matches the predicted end-effector trajectory (C2, green) during the transport phase (2).

Table 3: Influence of residual dynamics learning on cube-stacking performance. Residual variants do not improve significantly over the nominal baseline and often increase cost.

Experiment	SR	Mean cost
Baseline	0.394	234.7
$\dot{g} \leftarrow [x, u, \text{contact}]$	0.341	275.6
$\dot{g} \leftarrow [\text{contact}, \text{depth}]$	0.401	278.2
$\dot{g} \leftarrow [x, u, \text{contact}, \text{depth}]$	0.323	273.4
$\dot{x} \leftarrow [x, u, \text{contact}]$	0.361	296.4

consistently increase the mean cost. This supports the hypothesis stated in the results section: modeling the exact dynamics of the cube is not the bottleneck in this task. Once the task phase is inferred robustly, the phased nominal model is already sufficient to plan approach, lift, and transport, whereas additional residual dynamics introduces optimization non-convexity that hurts iLQR. In practice, the added residual model occasionally induced erratic motions, such as brief gripper opening or slight end-effector jerks. During the transport phase, when the end effector had to maintain a firm grasp, such events often led to dropping the object, reducing success rate and increasing cost.

G Solver and Learning Settings

Tables 4, 5, and 6 summarize the shared solver and learning configuration, the task-specific settings used in the quadrotor and cube-stacking experiments, and the PPO hyperparameters.

Table 4: Shared solver, optimization, and data-collection settings used in both tasks.

Setting	Value
Planning time step Δt_{plan}	0.05 s
Horizon H	20
Parallel environments	100
iLQR iterations	8
Maximum line-search trials	9
Maximum regularization updates	10
Executed control steps per replan	3
Residual-model initialization	Gaussian, standard deviation 10^{-3}
Residual rollout loss	10 steps, weight 0.1, midpoint integration
Critic initialization	Gaussian, standard deviation 10^0
Value-target construction	TD(5)
Variance-based λ attenuation	$\kappa = 4.0$
Blend schedule	$1.0 \rightarrow 0.9$
Update interval	every 20 control steps
Transitions per update	2×10^3
Mini-batches per update	10
Critic replay buffer	10^4
Dynamics replay buffer	10^7

Table 5: Task-specific timing and network settings.

Setting	Quadrotor	Cube stacking
Episode length	5 s	8 s
Control time step Δt_{ctrl}	0.005 s	0.01 s
Residual model	ELU MLP, 3 layers, width 128	ELU MLP, 3 layers, width 256
Critic trunk	ELU MLP, 3 layers, width 128	ELU MLP, 3 layers, width 256
Value ensemble	4 heads, 1 layer, width 128, mean aggregation	4 heads, 1 layer, width 256, mean aggregation

Table 6: PPO hyperparameters.

Setting	Value
Parallel Environments	10^4
Learning epochs	5
Mini-batches	4
Clipping parameter	0.2
Discount factor γ	0.99
GAE parameter λ	0.95
Value-loss coefficient	1.0
Entropy coefficient	0.01
Learning rate	10^{-3}
Desired KL	0.01

H Quaternion Projection

Often in aerospace and robotics, one is interested in optimizing over singularity-free rotation parametrizations. To make the solver accessible to a broad range of tasks, we require native support for state vectors that mix Euclidean states with quaternion components. During the forward rollout, quaternion states are renormalized after numerical integration, and during line search, the displacement $\text{d}x(\hat{x}_i, x_i)$ uses Euclidean differences on vector blocks and local rotation errors on

quaternion blocks. In the backward pass, quaternion derivatives are projected to local tangent coordinates because direct perturbations in the four-dimensional quaternion coordinates introduce radial components that violate $\|q\| = 1$ and make the local quadratic model inconsistent after renormalization [20]. We therefore map derivatives from the quaternion representation q to a local angle-axis coordinate $\phi \in \mathbb{R}^3$ using the geometric Jacobian $G(q)$. For a quaternion-dependent function h , this gives $h_x^{\text{red}} = E^\top h_x$ and $h_{xx}^{\text{red}} \approx E^\top h_{xx} E$, where the Hessian projection omits second-order curvature terms in the spirit of the iLQR Gauss-Newton approximation. For dynamics, input and output states lie in different tangent spaces, so $f_{x,i}^{\text{red}} = E_{i+1}^\top f_{x,i} E_i$ and $f_{u,i}^{\text{red}} = E_{i+1}^\top f_{u,i}$ [20]. Appendix E.1 gives a concrete quadrotor example, and in the following we will detail the construction of $G(q)$.

Given a current unit quaternion q and a local Rodrigues perturbation $\psi \in \mathbb{R}^3$, define

$$\delta q(\psi) = \frac{1}{\sqrt{1 + \|\psi\|^2}} \begin{bmatrix} 1 \\ \psi \end{bmatrix}, \quad q^+(\psi) = q \otimes \delta q(\psi).$$

For a function $h : S^3 \rightarrow \mathbb{R}^p$, the local rotational Taylor expansion is

$$h(q^+(\psi)) = h(q) + \left. \frac{\partial h(q \otimes \delta q(\psi))}{\partial \psi} \right|_{\psi=0} \psi + \mathcal{O}(\|\psi\|^2).$$

Using left quaternion multiplication $q \otimes \delta q = L(q)\delta q$, the projected Jacobian is

$$\left. \frac{\partial h}{\partial \psi} \right|_{\psi=0} = \left. \frac{\partial h}{\partial q} L(q) \frac{\partial \delta q}{\partial \psi} \right|_{\psi=0}.$$

For the Cayley parametrization,

$$\left. \frac{\partial \delta q}{\partial \psi} \right|_{\psi=0} = \begin{bmatrix} 0_{1 \times 3} \\ I_3 \end{bmatrix} \quad \text{and therefore} \quad G(q) = L(q) \begin{bmatrix} 0_{1 \times 3} \\ I_3 \end{bmatrix}, \quad (15)$$

which gives the result in [20].

with $q = [w, v^\top]^\top$ and quaternion left-multiplication matrix

$$L(q) = \begin{bmatrix} w & -v^\top \\ v & wI_3 + [v]_\times \end{bmatrix}.$$

For the Hessian, we reuse the first-order projection matrix from Eq. (15), yielding the projected Hessian (without 2nd-order curvature)

$$h_{\phi\phi} = G(q)^\top h_{qq}(q) G(q).$$

References

- [1] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL <http://incompleteideas.net/book/the-book-2nd.html>.
- [2] R. Reiter, J. Hoffmann, D. Reinhardt, F. Messerer, K. Baumgärtner, S. Sawant, J. Boedecker, M. Diehl, and S. Gros. Synthesis of model predictive control and reinforcement learning: Survey and classification, 2025. URL <https://arxiv.org/abs/2502.02133>.
- [3] H. J. T. Suh, M. Simchowitz, K. Zhang, and R. Tedrake. Do differentiable simulators give better policy gradients? *CoRR*, abs/2202.00817, 2022. URL <https://arxiv.org/abs/2202.00817>.
- [4] T. Marcucci, J. Umenberger, P. A. Parrilo, and R. Tedrake. Shortest paths in graphs of convex sets. *CoRR*, abs/2101.11565, 2021. URL <https://arxiv.org/abs/2101.11565>.
- [5] K. Lowrey, A. Rajeswaran, S. M. Kakade, E. Todorov, and I. Mordatch. Plan online, learn offline: Efficient learning and exploration via model-based control. *CoRR*, abs/1811.01848, 2018. URL <http://arxiv.org/abs/1811.01848>.
- [6] A. Aubret, L. Matignon, and S. Hassas. A survey on intrinsic motivation in reinforcement learning. *ArXiv*, abs/1908.06976, 2019. URL <https://api.semanticscholar.org/CorpusID:201103910>.
- [7] M. Bhardwaj, S. Choudhury, and B. Boots. Blending MPC & value function approximation for efficient reinforcement learning. *CoRR*, abs/2012.05909, 2020. URL <https://arxiv.org/abs/2012.05909>.
- [8] D. MAYNE. A second-order gradient method for determining optimal trajectories of nonlinear discrete-time systems. *International Journal of Control*, 3(1):85–95, 1966. doi:10.1080/00207176608921369. URL <https://doi.org/10.1080/00207176608921369>.
- [9] H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015. URL <http://arxiv.org/abs/1509.06461>.
- [10] O. Peer, C. Tessler, N. Merlis, and R. Meir. Ensemble bootstrapping for q-learning. *CoRR*, abs/2103.00445, 2021. URL <https://arxiv.org/abs/2103.00445>.
- [11] E. Todorov and W. Li. A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. *Proceedings of the 2005, American Control Conference, 2005.*, pages 300–306 vol. 1, 2005. URL <https://api.semanticscholar.org/CorpusID:14762932>.
- [12] Y. Tassa, T. Erez, and E. Todorov. Synthesis and stabilization of complex behaviors through on-line trajectory optimization. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4906–4913, 2012. doi:10.1109/IROS.2012.6386025.
- [13] M. J. D. Powell. A method for nonlinear constraints in minimization problems. 1969. URL <https://api.semanticscholar.org/CorpusID:115810962>.
- [14] T. A. Howell, B. E. Jackson, and Z. Manchester. Altro: A fast solver for constrained trajectory optimization. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7674–7679, 2019. doi:10.1109/IROS40897.2019.8967788.
- [15] *Penalty and Augmented Lagrangian Methods*, pages 497–528. Springer New York, New York, NY, 2006. ISBN 978-0-387-40065-5. doi:10.1007/978-0-387-40065-5_17. URL https://doi.org/10.1007/978-0-387-40065-5_17.
- [16] G. Authors. Genesis: A generative and universal physics engine for robotics and beyond, December 2024. URL <https://github.com/Genesis-Embodied-AI/Genesis>.

- [17] E. J. J. Smeur, Q. Chu, and G. C. H. E. de Croon. Adaptive incremental nonlinear dynamic inversion for attitude control of micro air vehicles. *Journal of Guidance, Control, and Dynamics*, 39(3):450–461, 2016. doi:10.2514/1.G001490. URL <https://doi.org/10.2514/1.G001490>.
- [18] J. Peters and J. A. Bagnell. *Policy Gradient Methods*, pages 774–776. Springer US, Boston, MA, 2010. ISBN 978-0-387-30164-8. doi:10.1007/978-0-387-30164-8_640. URL https://doi.org/10.1007/978-0-387-30164-8_640.
- [19] C. Schwarke, M. Mittal, N. Rudin, D. Hoeller, and M. Hutter. Rsl-rl: A learning library for robotics research. *arXiv preprint arXiv:2509.10771*, 2025.
- [20] B. E. Jackson, K. Tracy, and Z. Manchester. Planning with attitude. *IEEE Robotics and Automation Letters*, 6(3):5658–5664, 2021. doi:10.1109/LRA.2021.3052431.