PUSHING YOUR ROBOT POLICY TOWARD ITS SPEED LIMITS VIA COUPLED GAIN–DURATION OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Robot policies learned from demonstrations can perform increasingly complex tasks with high reliability, but their execution speed often remains constrained by the demonstrations from which they are learned. We argue that an important bottleneck lies one level below the policy—the low-level controller that determines how quickly and how long each action is physically realized. We introduce Coupled Gain–Duration Optimization (CGDO), which accelerates pretrained policies by jointly optimizing the controller gain and execution duration for each action with reinforcement learning (RL) without modifying the policy. These two variables respectively govern how aggressively the commanded action target is approached and how long the action is executed, and their effects are tightly coupled, making joint optimization critical for pushing execution toward the speed limits without sacrificing task performance. CGDO is policy-agnostic by design and can be combined with other RL methods for further acceleration. Across a suite of experiments, CGDO achieves substantially shorter execution times than other compared methods, while in most cases can also improve success rates. See Videos and code at `cgdo-anonymous.pyrite-apogee.workers.dev`.

1 INTRODUCTION

Robot policies learned through imitation can solve increasingly complex manipulation tasks (Chi et al., 2023; Zhao et al., 2023), yet successful execution can remain slow. Demonstrations show how to complete a task, but their timing also reflects teleoperation constraints and the operator's preference for reliable motion (Guo et al., 2025).

One way to accelerate these policies is to improve the actions they generate. Policy-level reinforcement learning (RL) can refine action sequences to reduce detours, hesitation, and repeated attempts (Wagenmaker et al., 2025; Ankile et al., 2025; Nam et al., 2026; Yuan et al., 2026). These methods optimize *what* actions the policy produces and the robot executes. Execution speed, however, also depends on *how* those actions are executed. The same action sequence can produce different robot motions under different execution settings specified by the underlying controller.

Position controllers are the de facto choice for low-level control in robot learning (Bronars et al., 2026). A policy predicts target joint or end-effector poses, and the controller converts the resulting pose error into torques that drive the robot toward those targets. The *proportional gain* determines how strongly the controller responds to this pose error, so increasing the gain can drive faster motion toward the same target. *Action execution duration*, on the other hand, determines how long each action remains active before the next action takes over. Shortening duration advances the action sequence sooner and can avoid unnecessary slowing near intermediate targets.

Recent methods already consider using high gains (Guo et al., 2025) or adaptive action execution duration (Arachchige et al., 2025; Kim et al., 2026) for policy acceleration. Importantly, these methods assume policies trained on data with *reached-pose* targets, where each action is labeled by a pose actually attained during data collection. At inference time, these predicted targets act as explicit waypoints; as long as they are still reached, gain and duration can vary substantially while preserving the intended sequence of poses, making them easier to tune for acceleration. However, many recent policies are instead trained on data with *commanded-pose* targets, where each action is labeled by the target issued to the controller during data collection, which need not be reached before the next command is issued. As we show in Section 3, for commanded poses, changing

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

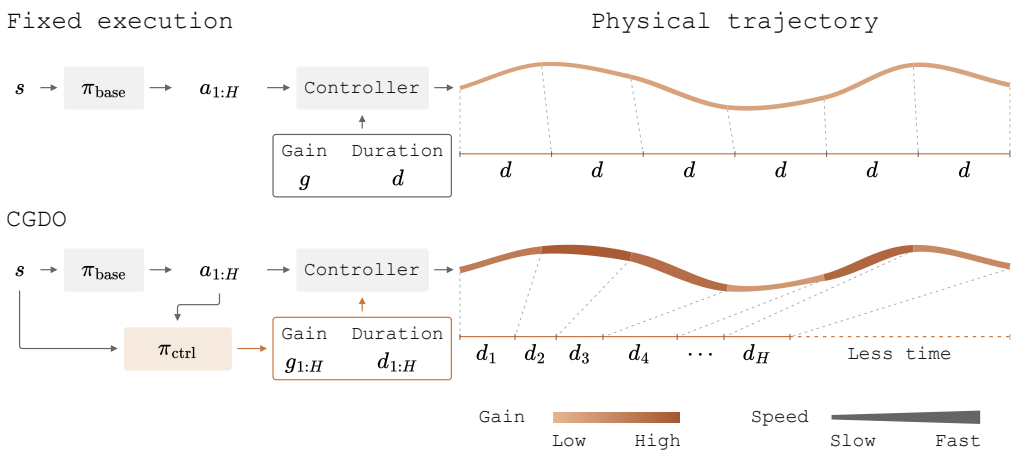


Figure 1: **Coupled gain–duration optimization accelerates policy execution.** Fixed execution (top) applies every predicted action with the same gain g and duration d. CGDO (bottom) keeps the base policy frozen and jointly adapts per-action gain and duration. Higher gain can drive the robot toward the current target faster, while shorter duration switches to the next action sooner; coordinating the two therefore allows the action sequence to be executed in less physical time.

the gains or execution duration can alter the resulting robot motion rather than simply reproduce the same trajectory faster. This can drive the robot into out-of-distribution states and degrade task performance, limiting the direct application of existing acceleration methods.

Our key insight is that, rather than relying on heuristic or analytic rules tied to reached-pose targets, we can learn controller parameters directly from task feedback with reinforcement learning, allowing action execution to be adaptively optimized according to their actual consequences, regardless of whether the policy uses reached- or commanded-pose targets. In addition, we find that gain and duration must also be coordinated rather than adjusted independently. As we show in Section 3, coordinated changes can approximately preserve the original path while executing it faster, whereas changing either alone can substantially alter the resulting motion. The appropriate coordination depends on the current state and dynamics, motivating joint learning rather than a fixed rule.

We introduce *Coupled Gain–Duration Optimization* (CGDO; Figure 1), which freezes a pretrained base policy and learns an adapter that predicts per-action gain and duration conditioned on the current state and the predicted action chunk. We train CGDO with off-policy RL using an elapsed-time-aware objective that directly rewards faster physical execution in a semi-MDP framework. CGDO is agnostic to the base-policy architecture and can also be applied on top of policies already improved by RL. It is also complementary to asynchronous execution methods such as RTC (Black et al., 2025), allowing execution-level acceleration to be combined with reduced policy-inference time.

Across simulated and real-world manipulation tasks, CGDO substantially accelerates execution while also improving performance on most tasks. Relative to the pretrained base policies, it achieves up to 12$\times$ faster execution and improves success rates across all evaluated tasks. When combined with policy-level RL, CGDO achieves the highest throughput-with-regret on eight of nine tasks, with particularly strong advantages when policies use commanded-pose targets.

2 Preliminaries

Action-chunked policy. At query n with state s_n, a base policy π_{base} samples an action chunk $A_n = (a_{n,1}, \ldots, a_{n,H}) \sim \pi_{\text{base}}(\cdot \mid s_n)$ and executes it open-loop. Each action specifies a controller target, such as a joint/pose target position or displacement. Let x_t denote the recorded robot joint/pose during data collection. Reached-pose datasets use $a_t^{\text{label}} = x_{t+1}$ (or $x_{t+1} - x_t$ for delta actions) as action labels, whereas commanded-pose datasets use $a_t^{\text{label}} = a_t^{\text{cmd}}$, where a_t^{cmd} is the target issued to the controller at time t and need not equal x_{t+1}.

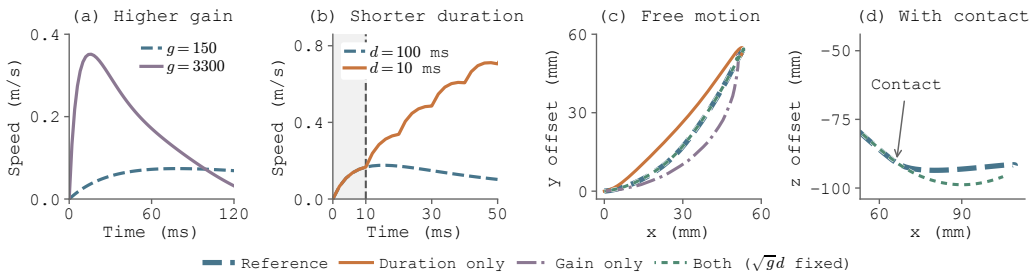


Figure 2: **Gain–duration dynamics study.** (a) Higher gain accelerates the initial speed. (b) Shorter duration reduces braking by advancing to the next target. (c) Either change alone alters the path; increasing gain and shortening duration with $\sqrt{g}d$ fixed approximately preserves it at fourfold peak speed. (d) The same rule fails to preserve the path when a separate downward motion encounters soft contact.

Controller gain. For a robot with n_q joints, a joint-space proportional–derivative (PD) position controller produces the torque command

$$\tau(t) = \mathcal{F}(q(t), \dot{q}(t), a; g) = \mathbf{K}_p(g)(q_{\text{des}}(a) - q(t)) - \mathbf{K}_d(g)\dot{q}(t), \tag{1}$$

where $q(t), \dot{q}(t) \in \mathbb{R}^{n_q}$ are joint positions and velocities at time t, and $q_{\text{des}}(a)$ is the joint target induced by action a. The gain matrices $\mathbf{K}_p(g), \mathbf{K}_d(g) \in \mathbb{R}^{n_q \times n_q}$ set the responses to position error and velocity, respectively, and are typically parameterized by a scalar proportional gain $g \in \mathbb{R}$, with $\mathbf{K}_d(g)$ adjusted according to the controller's damping rule.

Action execution duration. The duration $d > 0$ specifies how long an action a and its gain g remain active before the next action takes over. The controller continuously responds to the evolving robot state during the interval $[t, t + d]$, and switching occurs after d, regardless of target attainment.

Markov and semi-Markov decision processes. A Markov decision process (MDP) is specified by $(\mathcal{S}, \mathcal{A}, P, r, \rho_0, \gamma)$, with states, actions, transition kernel, reward function, initial-state distribution, and discount factor. At each decision, $a_n \sim \pi(\cdot \mid s_n)$ yields reward r_n and next state $s_{n+1} \sim P(\cdot \mid s_n, a_n)$. Its return and action-value function are defined as $G_n = \sum_{k=0}^{\infty} \gamma^k r_{n+k}$ and $Q^\pi(s, a) = \mathbb{E}_\pi[G_n \mid s_n = s, a_n = a]$, respectively.

A semi-Markov decision process (SMDP) allows variable time between decisions (Bradtke & Duff, 1994; Sutton et al., 1999). Its transition kernel $\mathcal{P}(s_{n+1}, \Delta_n \mid s_n, a_n)$ jointly models the next state s_{n+1} and elapsed time $\Delta_n > 0$. Let $t_{n+1} = t_n + \Delta_n$. With exponential time-discount rate $\beta \geq 0$, the return is

$$G_n = \sum_{k=0}^{\infty} e^{-\beta(t_{n+k} - t_n)} r_{n+k}, \tag{2}$$

and Q^π is the corresponding conditional expectation. The continuation discount $e^{-\beta\Delta_n}$ accounts for actual elapsed time, reducing to a fixed discount $\gamma = e^{-\beta}$ for unit durations and $\beta > 0$.

3 Why Couple Gain and Duration?

We first study the execution dynamics underlying gain–duration control in a controlled setting. Using the `RoboMimic Lift` simulation scene, we execute prescribed sequences of pose targets from stationary initial states while varying proportional gain g and execution duration d (the targets are not necessarily reached). The motions are driven by these prescribed targets rather than policy outputs, allowing us to isolate how the two execution variables affect the resulting trajectory.

Gain and duration provide complementary forms of acceleration. As shown in Figure 2a, a higher gain produces a faster initial acceleration toward the current target. However, as the robot approaches the target, the position error decreases and the motion slows down. In contrast, Figure 2b shows that, a shorter duration switches to the next target sooner, before the robot has substantially slowed down near the current target. Higher gain therefore increases the speed of the response, while shorter duration reduces the time spent slowing near intermediate targets.

Gain and duration must be coordinated. Because gain accelerates the response to each target while duration determines how soon execution advances to the next one, faster execution without unnecessarily altering the resulting motion requires coordinating the two. We illustrate this in free motion using a quarter-circle target sequence, where deviations from the reference trajectory are easy to expose geometrically.

As a simple reference for coordinating the two, we derive a gain–duration scaling rule under an ideal free-motion model (Appendix F). For an ideal mass-normalized, critically damped controller, the response speed scales with $\sqrt{g}$ (Spong et al., 2006; Lynch & Park, 2017). Therefore, increasing g by b^2 makes the response b times faster. If the duration of each target is simultaneously reduced by the same factor (i.e., $\sqrt{g}d$ is fixed), then the controller will switch to the next target at the same stage of the motion. Therefore, the resulting trajectory will theoretically have the same shape, but is traversed b times faster. Figure 2c shows that the simulated robot closely follows this prediction. Increasing gain or shortening duration alone noticeably changes the trajectory, whereas the coordinated change approximately preserves the reference path while reaching about four times the peak speed.

The appropriate coupling depends on the interaction. The fixed rule above relies on all relevant dynamics accelerating together with the controller. For example, this assumption can fail during object contact, because scaling g and d speeds up the commanded motion without equivalently scaling the contact interaction, so the robot no longer reaches the same states at corresponding action switches. Figure 2d validates this, where the same gain–duration rule ceases to preserve the trajectory after robot contacts the table.

These results show that gain and duration should be coordinated, but that the appropriate coordination depends on the state and interaction. This motivates learning the two jointly from task feedback, rather than tuning either variable independently or prescribing a fixed relation between them.

4 Coupled Gain–Duration Optimization

CGDO adapts the controller gain and action execution time of a pretrained action-chunked policy. We first define the execution adapter, then formulate learning over its variable-duration decisions using a physical-time objective with reinforcement learning.

Execution interface. Given a frozen base policy π_{base}, we learn an execution policy π_{ctrl} parameterized by ϕ. At query n, it receives the current state s_n and predicted chunk A_n, and selects gain–duration pairs for all H actions:

$$u_n = \big((g_{n,1}, d_{n,1}), \ldots, (g_{n,H}, d_{n,H})\big) \sim \pi_{\text{ctrl}}(\cdot \mid s_n, A_n). \tag{3}$$

Gains and durations lie within prescribed bounds, with strictly positive durations. All pairs are selected before the chunk is executed; they may be shared across actions according to the controller interface. The controller executes the unchanged commands in order, applying gain $g_{n,j}$ for duration $d_{n,j}$ to action $a_{n,j}$ under Equation 1. Low-level feedback remains active throughout execution. At the next chunk boundary, both policies observe the resulting state and predict the next actions and execution settings. This retains closed-loop adaptation while keeping the base-policy parameters frozen.

Variable-duration decisions. The adapter induces an SMDP with decision state $x_n = (s_n, A_n)$ and action u_n. Executing this choice produces $x_{n+1} = (s_{n+1}, A_{n+1})$ after elapsed time Δ_n. This duration includes execution up to termination and any overhead charged by the deployment protocol. Using the same elapsed time for reward and discount allows the adapter to compare execution choices with different physical durations.

Physical-time reward. To encourage earlier success, we charge a constant cost rate $c > 0$ for time spent before completion. With variable-duration decisions, the penalty must reflect elapsed time rather than assign the same cost to every adapter query. Divide an interval Δ_n into M steps of length $h = \Delta_n/M$, each with reward $-ch$ and discount $\gamma_h = e^{-\beta h}$. The continuous-time limit gives

$$R_n = \lim_{M\to\infty} \left[-ch \sum_{k=0}^{M-1} \gamma_h^k\right] = -c \int_0^{\Delta_n} e^{-\beta t}\, dt. \tag{4}$$

Combining this within-interval discount with the SMDP return in Equation 2 yields a single time-cost integral. For success after N intervals at time $T = t_N$, measured from $t_0 = 0$,

$$G_0 = \sum_{n=0}^{N-1} e^{-\beta t_n} R_n = -c \int_0^T e^{-\beta t}\, dt. \tag{5}$$

The resulting cost depends on elapsed physical time rather than the number of adapter queries. It is linear in time for $\beta = 0$ and exponentially discounted for $\beta > 0$. Earlier completion increases return, while unsuccessful execution continues to accrue cost. This supplies a direct learning signal for acceleration at the controller level.

Learning the execution policy. We use soft actor–critic (SAC) (Haarnoja et al., 2018) to optimize the continuous execution decisions, reusing collected transitions through off-policy updates. Its critic $Q_\theta(x, u)$ estimates the entropy-regularized return under π_{ctrl}. Given transitions from replay buffer $\mathcal{D}$, we minimize

$$\mathcal{L}_Q(\theta) = \mathbb{E}_{\mathcal{D}}\Big[\big(Q_\theta(x_n, u_n) - \big[R_n + e^{-\beta \Delta_n} \bar{V}(x_{n+1})\big]\big)^2\Big]. \tag{6}$$

The continuation value accounts for the next state and its newly predicted chunk. For nonterminal x', a slowly updated target critic $\bar{Q}$ and entropy temperature $\alpha \geq 0$ give

$$\bar{V}(x') = \mathbb{E}_{u' \sim \pi_{\text{ctrl}}(\cdot|x')}\left[\bar{Q}(x', u') - \alpha \log \pi_{\text{ctrl}}(u' \mid x')\right]. \tag{7}$$

The actor learns to select high-value execution controls by minimizing

$$\mathcal{L}_\pi(\phi) = \mathbb{E}_{x \sim \mathcal{D},\, u \sim \pi_{\text{ctrl}}(\cdot|x)}\left[\alpha \log \pi_{\text{ctrl}}(u \mid x) - Q_\theta(x, u)\right]. \tag{8}$$

Entropy encourages exploration at decision boundaries and is separate from the physical-time task cost. Training alternates between execution and replay updates, without differentiating through the base policy or robot dynamics.

5 Experiments

We test whether learning controller gain and action execution duration accelerates pretrained base policies, including those already improved by policy-level RL. We evaluate CGDO across policy architectures and action labels, with asynchronous execution, and on a real robot. Ablations and analysis of the learned execution then examine how gain and duration contribute to these speedups.

5.1 Experimental Setup

Tasks and base policies. We evaluate policies trained with both *reached-pose* and *commanded-pose* action labels, as defined in Section 2. For `Lift`, `Can`, and `Square` from `RoboMimic` (Mandlekar et al., 2021), and `Stack` and `Mug` from `MimicGen` (Mandlekar et al., 2023), we create reached-pose labels by replacing demonstration actions with the poses actually attained. For `Door`, `Hammer`, `Pen`, and `Relocate` from `Adroit` (Rajeswaran et al., 2018), we retain the original commanded-pose labels: the targets issued to the controller during data collection. The main simulation experiments use Diffusion Policy (DP) (Chi et al., 2023) trained on low-dimensional observations. Additional experiments use ACT (Zhao et al., 2023), DP trained on `RoboMimic`'s original commanded-pose labels (Appendix D.1), and a vision-based DP on a Piper arm.

Baselines. Our baselines include **DemoSpeedup** (Guo et al., 2025) and **SpeedAug** (Nam et al., 2026) for demonstration-based acceleration; **SAIL** (Arachchige et al., 2025) for high-gain tracking with adaptive timing; **RACE** (Kim et al., 2026) for constrained trajectory retiming; and **SpeedTuning** (Yuan et al., 2026) for interpolation between action targets. We also compare with **DSRL** (Wagenmaker et al., 2025) and **ResFiT** (Ankile et al., 2025), which use RL to improve the actions produced by the base policy. To test whether CGDO can further accelerate these policies, we freeze each resulting policy and train a CGDO execution policy, yielding **DSRL+CGDO** and **ResFiT+CGDO**.

Evaluation. Following SAIL (Arachchige et al., 2025), we use throughput-with-regret (TPR↑) to jointly assess execution speed and reliability: faster successes receive larger scores, and failures incur a fixed penalty. We also report success rate (SR↑), average time for successful rollouts (ATR↓),

Table 1: **Execution performance on `RoboMimic` and `MimicGen`.** We report SR, TPR (/s), ATR (s), and SoD. Best values are **bold** and second-best underlined, including ties at two decimal places.

	Lift				Can				Square				Stack				Mug			
Method	SR↑	TPR↑	ATR↓	SoD↑	SR↑	TPR↑	ATR↓	SoD↑	SR↑	TPR↑	ATR↓	SoD↑	SR↑	TPR↑	ATR↓	SoD↑	SR↑	TPR↑	ATR↓	SoD↑
Base policy	0.86	0.11	7.68	0.67	0.59	0.02	11.99	0.85	0.18	-0.03	16.07	0.79	0.91	0.11	8.86	0.61	0.32	-0.01	23.24	0.73
DemoSpeedup	0.99	0.28	3.82	1.34	0.97	0.13	7.64	1.33	0.53	0.02	11.84	1.07	0.99	0.37	2.80	1.93	0.88	0.10	9.06	1.87
SAIL	**1.00**	0.63	1.78	2.88	0.75	0.17	4.41	2.30	0.48	0.05	7.65	1.65	0.85	0.37	3.05	1.77	0.48	0.03	9.83	1.73
RACE	0.97	0.28	4.13	1.24	0.80	0.11	6.74	1.51	0.39	0.02	9.30	1.36	0.89	0.25	4.48	1.21	0.60	0.03	13.86	1.23
DSRL	**1.00**	0.52	1.91	2.68	**0.99**	0.20	5.09	2.00	**0.99**	0.19	5.14	2.46	0.98	0.29	3.44	1.57	0.94	0.11	8.61	1.97
SpeedTuning	**1.00**	0.26	4.19	1.23	0.96	0.12	8.05	1.26	0.26	-0.02	15.38	0.82	0.99	0.21	5.02	1.07	0.42	0.00	22.40	0.76
ResFiT	**1.00**	0.32	3.39	1.51	0.83	0.11	6.81	1.49	0.98	0.20	4.86	2.60	0.96	0.23	4.68	1.15	**0.98**	0.10	10.15	1.67
SpeedAug	**1.00**	**2.02**	**0.50**	**10.22**	**0.99**	0.40	2.52	4.03	0.78	0.11	6.97	1.81	**1.00**	0.62	1.97	2.73	0.92	0.14	6.83	2.49
CGDO	**1.00**	1.64	0.63	8.21	0.98	0.54	1.86	5.47	0.74	0.11	6.65	1.90	0.99	0.71	**1.43**	**3.77**	0.97	0.25	4.00	4.25
DSRL+CGDO	**1.00**	1.97	0.51	9.99	**0.99**	**0.59**	**1.67**	**6.08**	0.93	0.39	2.48	5.11	0.91	0.53	1.81	2.98	0.93	0.27	3.48	4.88
ResFiT+CGDO	**1.00**	1.41	0.74	6.97	0.94	0.48	1.99	5.11	0.94	**0.45**	**2.21**	**5.72**	**1.00**	**0.78**	1.45	3.73	0.97	**0.30**	**3.36**	**5.06**

Table 2: **Execution performance on `Adroit` hand manipulation tasks.** Metrics and ranking follow Table 1; actuator limits are unchanged. RACE treats commanded-pose targets as waypoints (Appendix A.3). Dashes indicate no successful rollouts.

	Door				Hammer				Pen				Relocate			
Method	SR↑	TPR↑	ATR↓	SoD↑	SR↑	TPR↑	ATR↓	SoD↑	SR↑	TPR↑	ATR↓	SoD↑	SR↑	TPR↑	ATR↓	SoD↑
Base policy	0.31	0.07	0.76	2.63	0.29	0.20	0.58	3.43	0.37	1.11	0.47	2.13	0.11	-0.28	0.82	2.45
DemoSpeedup	0.08	-0.34	0.81	2.48	0.06	-0.24	0.39	5.14	0.13	0.07	0.48	2.08	0.10	-0.25	0.67	3.00
SAIL	0.63	0.76	0.72	2.77	0.61	1.04	0.52	3.86	0.08	-0.03	0.69	1.47	0.32	0.20	0.76	2.62
RACE	0.00	-0.50	—	—	0.00	-0.50	—	—	0.34	1.21	0.47	2.13	0.00	-0.50	—	—
DSRL	0.97	1.87	0.58	3.45	0.98	3.98	0.25	8.03	**0.99**	4.94	0.24	4.28	**1.00**	2.96	0.35	5.65
SpeedTuning	0.56	0.55	0.75	2.65	0.57	0.90	0.57	3.49	0.64	2.03	0.48	2.10	0.51	0.49	0.84	2.39
ResFiT	0.99	1.70	0.62	3.23	0.95	2.47	0.42	4.76	0.85	3.08	0.46	2.19	0.97	2.21	0.48	4.13
SpeedAug	0.01	-0.49	0.80	2.50	0.44	0.25	1.02	1.96	0.36	0.76	0.56	1.80	0.12	-0.29	0.97	2.07
CGDO	0.99	2.31	0.47	4.27	0.96	3.81	0.26	7.67	0.93	3.89	0.33	3.03	0.88	1.82	0.53	3.80
DSRL+CGDO	**1.00**	**3.61**	**0.28**	**7.18**	**1.00**	**8.31**	**0.12**	**16.44**	**0.99**	**6.14**	**0.20**	**5.10**	0.97	**4.09**	**0.25**	**8.12**
ResFiT+CGDO	0.99	2.35	0.44	4.53	0.98	4.87	0.21	9.65	0.97	5.06	0.27	3.76	0.98	2.41	0.46	4.31

and speedup-over-demo (SoD↑), the mean demonstration duration divided by ATR. Simulation measures elapsed physical time; real-robot evaluation measures wall-clock time.

In the main simulation experiments, the CGDO execution policy selects controller gain g and action execution duration d for the predicted actions, as in Section 4. The damping gain follows the controller's existing rule. Section 5.4 also examines whether learning an additional damping parameter improves performance.

5.2 Accelerating Base Policies

CGDO increases SR and reduces ATR relative to the pretrained base policies on all nine tasks (Tables 1 and 2), yielding 1.42–12.29× faster successful execution. On `Lift`, ATR drops from 7.68 to 0.63 s while SR rises from 85.7% to 100%. On `Can`, `Stack`, and `Mug`, CGDO also completes successful rollouts faster than the demonstration-based, trajectory-tracking, and interpolation baselines while maintaining high success rates. On `Mug`, for example, it is 1.71× faster than SpeedAug, the fastest of these baselines, and improves SR from 91.7% to 96.7%. SpeedAug learns from demonstrations with intermediate actions skipped, which may explain its strong performance on `Lift`'s nearly straight vertical motion. Such skipping can distort curved motions and the response to commanded-pose targets, making acceleration harder on other tasks, especially `Adroit`.

On `Adroit`, where policies use commanded-pose targets, CGDO achieves higher success rates than RACE, DemoSpeedup, SAIL, and SpeedAug on all four tasks. On `Relocate`, it reaches 88.0% SR and 1.82 s^{-1} TPR, compared with 32.0% and 0.20 s^{-1} for SAIL, the strongest of these four baselines on both metrics. As discussed in Section 2, commanded-pose targets need not be reached before the next action starts. CGDO learns gain and duration from task feedback, so it can accelerate these policies without requiring the robot to reach every predicted target.

Table 3: **Results with ACT.** For each task, execution with fixed gain and duration and execution with CGDO use the same frozen ACT policy, actuator limits, and 300 initial states. Metrics follow Table 1.

	Can				Square			
Variant	SR↑	TPR↑	ATR↓	SoD↑	SR↑	TPR↑	ATR↓	SoD↑
ACT + fixed settings	0.35	-0.01	9.66	1.05	0.55	0.03	11.74	1.08
ACT + CGDO	**0.82**	**0.23**	**3.62**	**2.81**	**0.86**	**0.17**	**5.18**	**2.44**

Table 4: **CGDO with synchronous and RTC execution.** Each configuration uses 300 episodes per task. Metrics follow Table 1 and include total elapsed time.

	Lift				Can				Square				Stack				Mug			
Method	SR↑	TPR↑	ATR↓	SoD↑	SR↑	TPR↑	ATR↓	SoD↑	SR↑	TPR↑	ATR↓	SoD↑	SR↑	TPR↑	ATR↓	SoD↑	SR↑	TPR↑	ATR↓	SoD↑
Synchronous CGDO	1.00	1.65	0.62	8.23	0.97	0.43	2.29	4.44	0.51	0.08	5.51	2.30	0.99	0.68	1.56	3.45	0.86	0.18	5.02	3.38
CGDO with RTC	1.00	1.66	0.62	8.24	0.97	0.55	1.80	5.65	0.68	0.11	6.11	2.07	1.00	0.74	1.38	3.91	0.90	0.19	4.80	3.54

CGDO also provides further acceleration after policy-level RL. Adding it to a frozen DSRL or ResFiT policy reduces ATR on every task relative to executing that policy with fixed gain and duration (Appendix A.3). On `Lift`, ResFiT+CGDO reduces ATR from 3.39 to 0.74 s (4.60× faster), with both methods achieving 100% SR. Policy-level RL improves what actions the base policy predicts; CGDO learns how strongly the controller responds to each target and when to switch to the next action. The results show that improving the actions still leaves room to accelerate their execution.

Other policy architectures. We next evaluate CGDO with ACT, a transformer-based action-chunking policy. CGDO improves success and more than halves ATR on both `Can` and `Square` (Table 3). On `Can`, SR rises from 34.7% to 82.3% while successful execution becomes 2.67× faster. The improvement therefore extends beyond diffusion policies.

Asynchronous execution with RTC. With RTC (Black et al., 2025), the robot continues executing previously selected actions while the base policy predicts the next chunk and the execution policy selects its gains and durations. In our experiments, CGDO with RTC achieves higher TPR than synchronous CGDO on all five `RoboMimic` and `MimicGen` tasks (Table 4). On `Can`, ATR decreases from 2.29 to 1.80 s at the same success rate. These results show that CGDO also works with asynchronous execution, allowing the robot to further accelerate.

5.3 Real-Robot Execution

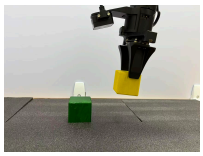
We deploy CGDO on a Piper arm for `Pick Bread`, `Nest Cups`, and `Stack Block` to test whether it accelerates execution when communication and visual inference also take time. Each task uses a vision-based DP with third-person and wrist-camera observations. We compare its original execution with CGDO while keeping the base-policy parameters frozen.



`Pick Bread`



`Nest Cups`



`Stack Block`

Task		SR↑	TPR↑	ATR↓	SoD↑
`Pick Bread`	Base Policy	**1.00**	0.06	16.88	0.82
	CGDO	**1.00**	**0.25**	**3.97**	**3.48**
`Nest Cups`	Base Policy	**0.80**	0.05	12.86	0.92
	CGDO	**0.80**	**0.17**	**4.49**	**2.63**
`Stack Block`	Base Policy	**1.00**	0.08	13.17	0.85
	CGDO	**1.00**	**0.28**	**3.54**	**3.17**

Table 5: **Real-robot performance.**

CGDO makes successful execution 2.86–4.25× faster across the three tasks, with unchanged observed success rates (Table 5). On `Nest Cups`, mean completion time falls from 12.86 to 4.49 s while both methods retain an 80% success rate. On `Stack Block`, CGDO completes all 20 trials in 3.54 s on average, compared with 13.17 s for the base policy. These gains include communication and inference overhead, demonstrating substantial acceleration in the deployed system.

5.4 Ablation Studies

On `Lift`, we compare CGDO, which learns gain and duration together, with three alternatives: keeping both quantities fixed, learning only gain, and learning only duration. When only one quantity is learned, the other remains fixed. All variants use the same frozen DP, observations, actuator limits, and held-out initial states. Fixed values are selected on development episodes, and each execution policy is trained separately.

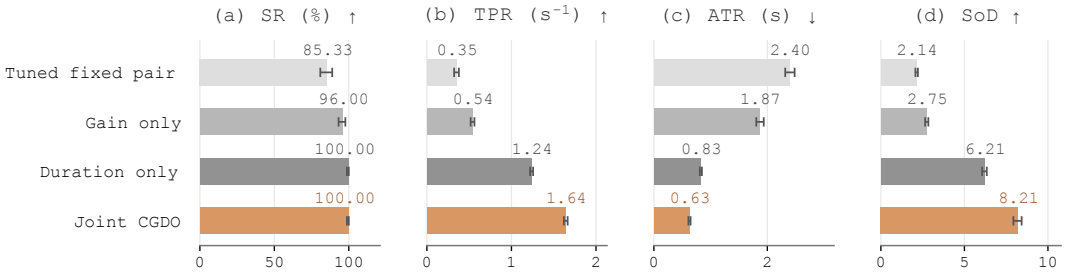


Figure 3: **Learning gain and duration together further accelerates `Lift`.**

CGDO increases SR and reduces ATR relative to the tuned fixed settings on `Lift` (Figure 3). Learning only duration already provides most of the speedup. Learning gain and duration together reduces ATR further, from 0.83 to 0.63 s, a 24.4% reduction, with both variants succeeding in every evaluation episode. Thus, even after learning when to switch actions, learning how strongly the controller responds to each target can further accelerate execution.

Learning damping. The gain–duration formulation above retains the damping rule $\mathbf{K}_d(g)$ from Section 2. We next test whether additionally learning a factor that scales this response to velocity improves execution. It reduces `Lift`'s ATR from 0.63 to 0.35 s at 100% SR, but lowers SR on the other four tasks, particularly `Square` and `Mug` (Table 6). Learning damping thus helps on individual tasks but offers no overall advantage across these tasks. We therefore retain the prescribed damping rule and learn only gain and duration.

Table 6: **Learning an additional damping parameter.** Each variant uses 300 episodes per task and a separately selected study checkpoint. Bold marks the better value per task/metric.

	`Lift`		`Can`		`Square`		`Stack`		`Mug`	
Learned parameters	SR (%)	ATR (s)	SR (%)	ATR (s)	SR (%)	ATR (s)	SR (%)	ATR (s)	SR (%)	ATR (s)
g, d	**100.00**	0.63	**98.33**	**1.86**	**73.67**	6.65	**98.67**	**1.43**	**94.67**	**4.29**
g, d + damping	**100.00**	**0.35**	91.67	2.11	40.67	**5.86**	97.00	1.47	54.33	5.68

5.5 Learned Execution Behavior

CGDO allocates execution time selectively. On `Lift`, only 4.1% of actions have an execution duration $d > 10$ ms, yet they account for 65.7% of execution time (Figure 4a). Execution time is concentrated on the last action in each chunk on `Lift`, whereas `Square` distributes time more evenly across actions (Figure 4b). Thus, the learned durations vary both within a chunk and across tasks.

To test whether fixed durations for each action index are sufficient, we replace the learned durations with their means at each index, estimated from separate episodes. This substantially reduces SR on both tasks (Figure 4c). The execution policy therefore benefits from adapting durations across chunks, beyond assigning a fixed duration to each action index.

The learned execution also tolerates error relative to intermediate action targets: on `Lift`, joint CGDO finishes sooner than fixed or gain-only execution despite larger tracking errors (Appendix C.2). Together with the response analysis in Section 3, these results support adapting execution to task outcomes rather than requiring every predicted target to be reached.

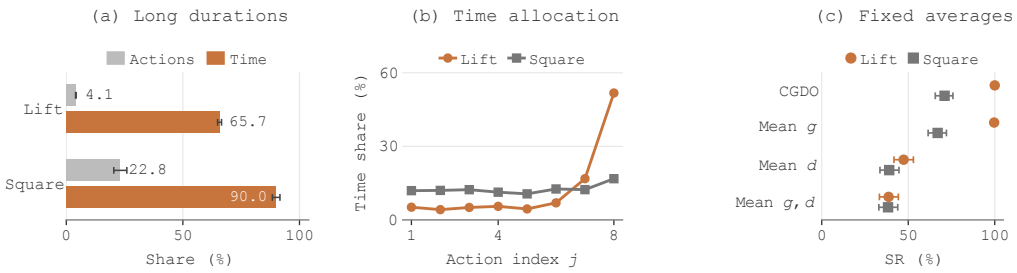


Figure 4: **Learned gain–duration adaptation.** (a) Shares of actions and execution time with $d > 10$ ms. (b) Time share by action index j within a chunk. (c) SR after fixing g, d, or both to their mean at each index, estimated from separate episodes, without retraining. Each task/condition uses 300 episodes; intervals are 95% episode-bootstrap in (a) and Wilson in (c).

6 Related Work

Robot policy acceleration. Data-centric approaches expose robot policies to faster trajectories during training. DemoSpeedup retrains policies on downsampled demonstrations (Guo et al., 2025), while SpeedAug augments demonstration tempos and further fine-tunes the policy with RL (Nam et al., 2026). During execution, SAIL combines high-gain tracking with adaptive action timing (Arachchige et al., 2025), and RACE retimes predicted trajectories under dynamic constraints (Kim et al., 2026). Both rely on reached-pose targets, limiting their direct compatibility with policies trained on commanded-pose actions. SpeedTuning and Speedup Patch instead accelerate predicted action chunks through temporal interpolation (Yuan et al., 2026) and adaptive downsampling (Wu et al., 2026), respectively. Such resampling changes the command sequence presented to the controller and therefore need not preserve the original closed-loop trajectory. CGDO takes a controller-level approach: it leaves the predicted actions unchanged and jointly learns controller gain and action duration from task feedback.

RL with Controller. Reinforcement learning has long been used below the task-policy level to adapt how robot actions are executed. Early work optimized impedance gains together with movement trajectories (Buchli et al., 2011), while later approaches made impedance state-dependent RL outputs jointly predicted with motion commands (Martín-Martín et al., 2019; Bogdanovic et al., 2020). The learned control space has since expanded to include discrete control modes, action repetition, and skill-specific controller parameters (Neunert et al., 2020; Berjaoui Tahmaz et al., 2025). More recently, OM-1 trains a dedicated high-frequency low-level control policy with RL beneath a general-purpose robot policy (Reward AI Team, 2026). CGDO similarly applies RL at the controller level, but retains the existing controller and frozen robot policy, jointly adapting only controller gain and action duration for the predicted actions.

7 Conclusion and Limitations

CGDO accelerates pretrained robot policies by jointly learning controller gain and action execution duration with RL, while keeping the base policy and its predicted actions unchanged. Our analysis shows that higher gain speeds up the response to each target, whereas shorter duration reduces slowing near intermediate targets. Their effects must be coordinated because faster execution can alter the realized trajectory, especially during contact. Learning this coordination from task feedback yields faster execution without requiring every predicted target to be reached. Experiments demonstrate faster execution across policies and tasks, including three real-robot tasks, with additional gains when combined with policy-level RL and asynchronous execution. These results identify controller-level adaptation as a complementary route to improving robot policies.

Limitations. CGDO still requires task-specific settings to train the execution policy; transfer to new tasks and robot dynamics remains untested. The time-based objective also does not explicitly constrain contact forces, so faster successful execution alone does not establish contact safety. Learning reusable execution policies and incorporating contact-aware constraints are important directions for future work.

AI USE STATEMENT

In this work, we used generative AI tools to assist with code implementation, running experiments, and revising the manuscript. The human authors conceived the research ideas, designed and iteratively refined the experiments, and wrote the initial manuscript, including the development and organization of the related work. We have reviewed all AI-assisted work. In particular, the main author carefully verified every experiment and its results. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Appendix A gives implementation and evaluation settings; Appendices B–D provide ablations, mechanism analyses, and deployment protocols. Appendices E and F detail physical-time learning and response dynamics. The project website provides simulation and real-world rollouts with visualizations, as well as code for reproduction.

REFERENCES

Lars Ankile, Zhenyu Jiang, Rocky Duan, Guanya Shi, Pieter Abbeel, and Anusha Nagabandi. Residual Off-Policy RL for Finetuning Behavior Cloning Policies, 2025. URL https://arxiv.org/abs/2509.19301.

Nadun Ranawaka Arachchige, Zhenyang Chen, Wonsuhk Jung, Woo Chul Shin, Rohan Bansal, Pierre Barroso, Yu Hang He, Yingyang Celine Lin, Benjamin Joffe, Shreyas Kousik, and Danfei Xu. SAIL: Faster-than-Demonstration Execution of Imitation Learning Policies, 2025. URL https://arxiv.org/abs/2506.11948.

Amin Berjaoui Tahmaz, Ravi Prakash, and Jens Kober. Impedance primitive-augmented hierarchical reinforcement learning for sequential tasks. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 10973–10979. IEEE, 2025. doi: 10.1109/ICRA55743.2025.11128462. URL https://doi.org/10.1109/ICRA55743.2025.11128462.

Kevin Black, Manuel Galliker, and Sergey Levine. Real-time execution of action chunking flow policies. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 33383–33407. Curran Associates, Inc., 2025. doi: 10.52202/085713-1122. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/300ccb2187dedd4edcc07f7e76d8e553-Paper-Conference.pdf.

Miroslav Bogdanovic, Majid Khadiv, and Ludovic Righetti. Learning variable impedance control for contact sensitive tasks. *IEEE Robotics and Automation Letters*, 5(4):6129–6136, 2020. doi: 10.1109/LRA.2020.3011379. URL https://doi.org/10.1109/LRA.2020.3011379.

Steven Bradtke and Michael Duff. Reinforcement learning methods for continuous-time markov decision problems. In G. Tesauro, D. Touretzky, and T. Leen (eds.), *Advances in Neural Information Processing Systems*, volume 7, pp. 393–400. MIT Press, 1994. URL https://proceedings.neurips.cc/paper_files/paper/1994/file/07871915a8107172b3b5dc15a6574ad3-Paper.pdf.

Antonia Bronars, Younghyo Park, and Pulkit Agrawal. Tune to learn: How controller gains shape robot policy learning, 2026. URL https://arxiv.org/abs/2604.02523.

Jonas Buchli, Freek Stulp, Evangelos Theodorou, and Stefan Schaal. Learning variable impedance control. *The International Journal of Robotics Research*, 30(7):820–833, 2011. doi: 10.1177/0278364911402527. URL https://doi.org/10.1177/0278364911402527.

Xinyue Chen, Che Wang, Zijian Zhou, and Keith Ross. Randomized Ensembled Double Q-Learning: Learning Fast Without a Model, 2021. URL https://arxiv.org/abs/2101.05982.

Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin CM Burchfiel, and Shuran Song. Diffusion Policy: Visuomotor Policy Learning via Action Diffusion. In *Proceedings of Robotics: Science and Systems*, Daegu, Republic of Korea, July 2023. doi: 10.15607/RSS.2023.XIX.026.

Lingxiao Guo, Zhengrong Xue, Zijing Xu, and Huazhe Xu. DemoSpeedup: Accelerating Visuomotor Policies via Entropy-Guided Demonstration Acceleration, 2025. URL https://arxiv.org/abs/2506.05064.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1861–1870. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/haarnoja18b.html.

Sunwoo Kim, Jeongjun Kim, and Joseph Lim. Time optimal execution of action chunk policies beyond demonstration speed. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 154823–154842, 2026. URL https://clvrai.github.io/RACE/.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative Q-Learning for Offline Reinforcement Learning, 2020. URL https://arxiv.org/abs/2006.04779.

Kevin M. Lynch and Frank C. Park. *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press, 2017. ISBN 9781316609842. doi: 10.1017/9781316661239. URL https://doi.org/10.1017/9781316661239.

Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. What Matters in Learning from Offline Human Demonstrations for Robot Manipulation, 2021. URL https://arxiv.org/abs/2108.03298.

Ajay Mandlekar, Soroush Nasiriany, Bowen Wen, Iretiayo Akinola, Yashraj Narang, Linxi Fan, Yuke Zhu, and Dieter Fox. MimicGen: A Data Generation System for Scalable Robot Learning using Human Demonstrations, 2023. URL https://arxiv.org/abs/2310.17596.

Roberto Martín-Martín, Michelle A. Lee, Rachel Gardner, Silvio Savarese, Jeannette Bohg, and Animesh Garg. Variable Impedance Control in End-Effector Space: An Action Space for Reinforcement Learning in Contact-Rich Tasks, 2019. URL https://arxiv.org/abs/1906.08880.

Taewook Nam, Junmo Cho, Youngsoo Jang, and Sung Ju Hwang. SpeedAug: Policy Acceleration via Tempo-Enriched Policy and RL Fine-Tuning, 2026. URL https://arxiv.org/abs/2512.00062.

Michael Neunert, Abbas Abdolmaleki, Markus Wulfmeier, Thomas Lampe, Tobias Springenberg, Roland Hafner, Francesco Romano, Jonas Buchli, Nicolas Heess, and Martin Riedmiller. Continuous-discrete reinforcement learning for hybrid control in robotics. In *Proceedings of the Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pp. 735–751. PMLR, 2020. URL https://proceedings.mlr.press/v100/neunert20a.html.

Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations, 2018. URL https://arxiv.org/abs/1709.10087.

Reward AI Team. OM-1: Frontier robot intelligence, learned firsthand from humans. *Reward AI Blog*, 2026. URL https://www.rewardai.com/blog/OM-1/.

Mark W. Spong, Seth Hutchinson, and M. Vidyasagar. *Robot Modeling and Control*. John Wiley & Sons, 2006. ISBN 9780471649908.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, August 1999. ISSN 0004-3702. doi: 10.1016/s0004-3702(99)00052-1. URL http://dx.doi.org/10.1016/S0004-3702(99)00052-1.

Andrew Wagenmaker, Mitsuhiko Nakamoto, Yunchu Zhang, Seohong Park, Waleed Yagoub, Anusha Nagabandi, Abhishek Gupta, and Sergey Levine. Steering Your Diffusion Policy with Latent Space Reinforcement Learning, 2025. URL https://arxiv.org/abs/2506.15799.

Zhichao Wu, Junyin Ye, Zhilong Zhang, Yihao Sun, Haoxin Lin, Jiaheng Luo, Haoxiang Ren, Lei Yuan, and Yang Yu. Speedup Patch: Learning a Plug-and-Play Policy to Accelerate Embodied Manipulation, 2026. URL https://arxiv.org/abs/2603.20658.

David D. Yuan, Tony Z. Zhao, Kaylee Burns, and Chelsea Finn. SpeedTuning: Speeding Up Policy Execution with Lightweight Reinforcement Learning, 2026. URL https://arxiv.org/abs/2608.09138.

Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware. In *Proceedings of Robotics: Science and Systems*, Daegu, Republic of Korea, July 2023. doi: 10.15607/RSS.2023.XIX.016.

Yuke Zhu, Josiah Wong, Ajay Mandlekar, Roberto Martín-Martín, Abhishek Joshi, Kevin Lin, Abhiram Maddukuri, Soroush Nasiriany, and Yifeng Zhu. robosuite: A Modular Simulation Framework and Benchmark for Robot Learning, 2025. URL https://arxiv.org/abs/2009.12293.

A Implementation and Evaluation

A.1 Execution and Learning Settings

Controller interfaces. The arm experiments use operational-space control in `robosuite` (Zhu et al., 2025), with scalar gain g, stiffness matrix gI_6, and damping matrix $2\sqrt{g}I_6$. Each query assigns one gain–duration pair to each of $H = 8$ actions. The execution policy observes the base-policy input, predicted chunk, joint positions and velocities, and gripper velocity; absolute rotations use a six-dimensional representation.

On `Adroit`, the base policy predicts 32 actions and executes the first eight. The execution policy observes these eight actions and selects one gain–duration pair shared across them. Gain scales the native actuator gain and position-bias coefficients together, preserving damping and control bounds. The observations also include arcsinh-transformed generalized velocities and remaining time.

A squashed Gaussian actor outputs normalized coordinates $z \in [-1, 1]^m$. With reference tick $\delta = 0.002\,\mathrm{s}$ and $\tilde{d} = d/\delta$, the decoder is

$$g = g_{\min} + \frac{z_g + 1}{2}(g_{\max} - g_{\min}), \qquad \tilde{d} = \exp\left(\log \tilde{d}_{\min} + \frac{z_d + 1}{2} \log \frac{\tilde{d}_{\max}}{\tilde{d}_{\min}}\right). \tag{9}$$

Arm bounds are $g \in [150, 3300]$, $\tilde{d} \in [0.1, 50]$; hand bounds are $g \in [0.8, 4]$, $\tilde{d} \in [0.5, 8]$. A final partial integration step implements fractional ticks. Success is checked after each integration call; reward and discount use actual elapsed time, including terminal truncation.

Table 7: Default CGDO settings. The control dimension is twice the number of independently controlled action positions.

Setting	Arm	`Adroit`
Executed actions / control dimension	8 / 16	8 / 2
Parallel environments / training seed	4 / 42	8 / 42
Actor and critic hidden layers (ReLU)	$512, 512, 512$	$256, 256, 256$
Batch size / replay capacity	256 / 10^6	256 / 3×10^5
Update-to-data ratio	4	1
Actor / critic learning rate	5×10^{-5} / 10^{-4}	5×10^{-5} / 10^{-4}
Critic ensemble / target subset	10 / 2	1 / 1
Target-network coefficient	0.005	0.005
Initial transitions / critic warm-up updates	32,768 / 20,000	8,192 / 2,000
Time penalty λ / discount γ_δ	0.04 / $0.997^{1/25}$	0.2 / 1
Entropy temperature	learned; init. 0.3, cap 0.7	fixed 0.001
Temperature learning rate / target entropy	10^{-5} / $0.1m \log 2$	—
Entropy weight	1	$\Delta_{\mathrm{req}}/(8 \cdot 5\delta)$
Critic target clipping / timeout bootstrap	$[-1000, 0]$ / yes	none / no

The arm ensemble uses the minimum of two sampled target critics and the ensemble mean for actor updates (Chen et al., 2021), with a conservative critic regularizer of weight 0.003 (Kumar et al., 2020). Actor densities, including the squashing correction, are evaluated in normalized coordinates. Initial means are $(g, \tilde{d}) = (1725, 5)$ on arms and $(1.1, 5)$ on hands. `Adroit` uses critic LayerNorm and an initial exploration range $g \in [0.8, 1.6]$, $\tilde{d} \in [2.5, 8]$, expanded after 20,000 transitions over 200,000 transitions. Its entropy weight uses requested duration; task reward and discount still use realized time.

For frozen DSRL/ResFiT policies, development selection compares the standard arm configuration with one initialized around the base policy's execution settings. Only DSRL on `Can` selects this alternative: actor learning rate 2×10^{-5}, 15% stochastic decisions, duration-weighted entropy, terminal timeouts, and $\tilde{d} \in [1, 50]$. `Adroit` compositions retain the hand configuration.

A.2 Metrics, Selection, and Uncertainty

Let z_i indicate success, t_i be first-success time, $T_{\max}$ the deadline, and $\overline{T}_{\text{demo}}$ the mean full-demonstration duration. We use

$$\text{SR} = \frac{\sum_i z_i}{N}, \quad \text{ATR} = \frac{\sum_{i:z_i=1} t_i}{\sum_i z_i}, \quad \text{SoD} = \frac{\overline{T}_{\text{demo}}}{\text{ATR}}, \quad \text{TPR} = \frac{1}{N}\sum_i \begin{cases} 1/t_i, & z_i = 1, \\ -1/T_{\max}, & z_i = 0. \end{cases} \tag{10}$$

ATR and SoD are undefined without successes. ATR averages each method's successful subset; TPR scores all episodes, including failures. Demonstration references include motion after first success.

Table 8: Evaluation horizons and demonstration references (seconds).

Task	Horizon	Demo	Task	Horizon	Demo
`Lift`	15	5.13	`Door`	2	2.00
`Can`	15	10.16	`Hammer`	2	2.00
`Square`	20	12.66	`Pen`	2	1.01
`Stack`	20	5.40	`Relocate`	2	2.00
`Mug`	25	16.98			

Controlled comparisons, ACT, commanded-action, training-seed, and composition studies select checkpoints on 100 development episodes and evaluate the selected checkpoint on 300 held-out episodes. Checkpoint selection ranks TPR, then SR, then lower ATR.

SR uncertainty uses 95% Wilson intervals; Figure 3 uses 10,000 episode-bootstrap resamples for other metrics. Appendix B.1 reports variation across independently trained adapters.

A.3 Baseline Settings and Policy Composition

CGDO uses 20× native arm torque limits on all `RoboMimic` tasks (`Lift`, `Can`, and `Square`) and on `Stack` and `Mug` from `MimicGen`. `Adroit` retains native actuator bounds. SAIL uses high-gain tracking. SpeedTuning learns an interpolation factor in $[0.5, 4]$; SpeedAug uses DPPO on `Hammer`/`Pen` and TempoBC on `Door`/`Relocate`.

RACE port. Our RACE implementation retimes cubic joint paths with TOPPRA and tracks them with a joint controller. It runs synchronously without TTS; reported simulation time excludes planning computation. We use ten development seeds and 300 held-out episodes. An infeasible plan reuses the remaining valid trajectory, or falls back to nominal joint PD when none is available. On `Adroit`, we directly use commanded-pose targets as waypoints without retraining the policy. These plans fail joint-limit or controllability checks, so all evaluated segments, including successful `Pen` episodes, use the 10 ms joint-PD fallback.

Composition with policy-level RL. CGDO reduces ATR for every frozen RL-adapted policy in Table 9, although some combinations reduce SR. Each comparison keeps the base-policy weights identical, measuring the additional acceleration from learning execution settings after policy-level RL.

B Additional Ablations

B.1 Independent Training Seeds

The three training seeds give similar results on `Lift`, while `Relocate` shows the largest variation (74–88% SR; Table 10). We train execution policies with seeds 42–44 and apply the common checkpoint-selection and evaluation protocol in Appendix A.2.

B.2 Control Space and Time-Cost Scale

Learning both gain and duration. On `Lift`, learning gain in addition to duration saves 0.17 s (95% paired bootstrap CI $[0.14, 0.20]$), with both variants succeeding in every episode (Table 12).

Task	Policy	Frozen policy SR↑	Frozen policy ATR↓	With CGDO SR↑	With CGDO ATR↓	ATR reduction (%)
Can	ResFiT	**98.67**	5.007	94.33	**1.987**	60.32
Square	ResFiT	**99.33**	4.930	94.33	**2.212**	55.13
Mug	ResFiT	97.00	10.253	**97.33**	**3.356**	67.27
Door	DSRL	**100.00**	0.441	99.67	**0.279**	36.82
Pen	DSRL	**99.00**	0.246	98.67	**0.197**	19.57
Relocate	DSRL	**100.00**	0.340	97.33	**0.246**	27.52

Table 9: **Further acceleration of RL-adapted policies.** Each pair evaluates the same frozen policy with and without CGDO (300 episodes per endpoint). SR is in percent and ATR in seconds; bold marks the better value per metric.

Table 10: Mean ± sample SD across three independently trained execution policies. SR is in percent and ATR in seconds.

Task	SR (%)↑	TPR↑	ATR↓	SoD↑
Lift	99.89 ± 0.19	1.61 ± 0.02	0.64 ± 0.02	7.99 ± 0.21
Square	69.22 ± 3.91	0.11 ± 0.01	6.45 ± 0.18	1.96 ± 0.05
Mug	95.78 ± 1.26	0.22 ± 0.03	4.46 ± 0.49	3.83 ± 0.42
Door	98.00 ± 0.88	2.15 ± 0.14	0.49 ± 0.02	4.08 ± 0.17
Relocate	82.33 ± 7.37	1.53 ± 0.39	0.60 ± 0.11	3.37 ± 0.54

The comparison uses the same frozen base policy, observations, actuator limits, and 300 paired initial states. Fixed values are tuned on 100 development episodes per setting; Table 11 collects the search ranges and selected values. Gain-only and duration-only policies are trained separately, with best/latest checkpoints selected on the development set.

Table 11: Fixed values for the Lift control-space ablation. Fixed execution searches all combinations of the listed gains and durations; $\tilde{d} = d/(2\,\mathrm{ms})$.

Variant	Fixed quantity	Candidate values	Selected
Fixed execution	g	$\{150, 500, 1000, 1725, 3000, 3300\}$	1000
	$\tilde{d}$	$\{0.1, 0.5, 1, 2, 5, 10, 25, 50\}$	5
Gain only	$\tilde{d}$	$\{1, 5, 25\}$	5
Duration only	g	$\{150, 1725, 3300\}$	1725

This appendix comparison uses a 5 s deadline. Figure 3 instead uses the main-table joint checkpoint (0.63 s ATR); all of its episodes also finish within 5 s.

Table 12: Restricted-control comparison on Lift (5 s deadline; 300 paired initial states). Metrics follow Table 1.

Variant	SR↑	TPR↑	ATR↓	SoD↑
Best fixed pair	0.85	0.35	2.40	2.14
Gain-only (selected duration)	0.96	0.54	1.87	2.75
Duration-only (selected gain)	**1.00**	1.24	0.83	6.21
Joint CGDO	**1.00**	**1.60**	**0.65**	**7.85**

Figure 5 shows how these speedups translate into earlier task completion across the full evaluation set. The cumulative success curve $F(t) = N^{-1} \sum_i \mathbb{I}[z_i = 1, t_i \le t]$ includes failures and reaches the overall SR at the deadline.

Sequential training. Both training orders yield near-perfect success and sub-second ATR on Lift (Table 13). In the second stage, both gain and duration remain trainable.

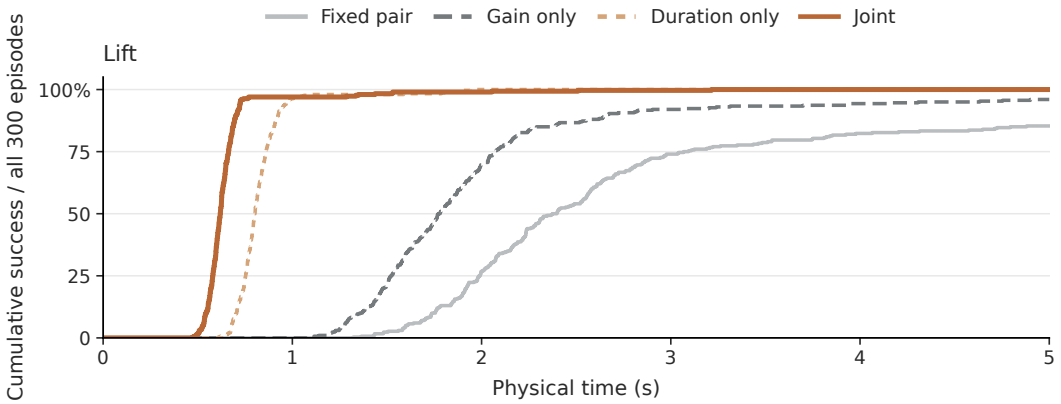


Figure 5: **Success by elapsed time.** All 300 episodes contribute to each curve, including failures. Controllers and initial states match Table 12.

Table 13: Sequential training on `Lift`, evaluated over 300 episodes per variant. Training counts are in millions.

First learned	Decisions before joint training	Total online physics steps	SR (%)	ATR (s)
Gain	5.00	221.080	99.00	0.699
Duration	5.25	90.910	99.67	0.625

Time penalty. Varying λ changes the time cost relative to entropy and optimization. Table 14 shows fast execution across the tested values, without a monotone reliability trend. The default $\lambda = 0.04$ is fixed before testing.

Table 14: Sensitivity to time penalty (300 episodes each). Bold marks the best displayed values per column; dashes indicate untested settings.

	`Lift`		`Square`	
λ	SR (%)	ATR (s)	SR (%)	ATR (s)
0.01	99.67	0.70	64.33	6.81
0.02	**100.00**	0.68	—	—
0.04	**100.00**	0.65	73.67	6.65
0.08	**100.00**	**0.61**	—	—
0.16	**100.00**	**0.61**	**74.00**	**5.73**

B.3 Learning the Damping Ratio

The damping experiment in Section 5.4 adds a learned ratio $\zeta \in [0.2, 2]$, giving damping matrix $2\zeta\sqrt{g I_6}$ for the arm tasks. Initialization at $\zeta = 1$ recovers the default rule; gain and duration bounds are unchanged. Table 6 uses 300 episodes per task and variant, with independently selected study checkpoints that can differ from the main task comparison.

C Understanding the Learned Execution

C.1 Where Is Execution Time Spent?

The time shares in Figure 4 use actual elapsed action durations, truncated at task termination. The execution policy selects all eight gain–duration pairs before the chunk starts, so the variation across actions reflects decisions made at the chunk boundary.

The gain–duration association differs across tasks. Comparing long and short durations within the same action index, the gain difference under deterministic execution is -6.4 on `Lift` (95% episode-

cluster CI $[-37.7, 29.4]$) and 113.4 on `Square` ($[82.6, 149.6]$). These results support learning the two quantities without imposing a fixed positive association.

C.2 Dependence on Learned Gain and Duration

We test dependence on learned variation by replacing gain, duration, or both with means for each action index, estimated from 100 separate calibration episodes. Evaluation uses 300 paired initial states with shared random draws and no retraining. The changes propagate through subsequent states and action predictions, so the results measure their effect on closed-loop execution.

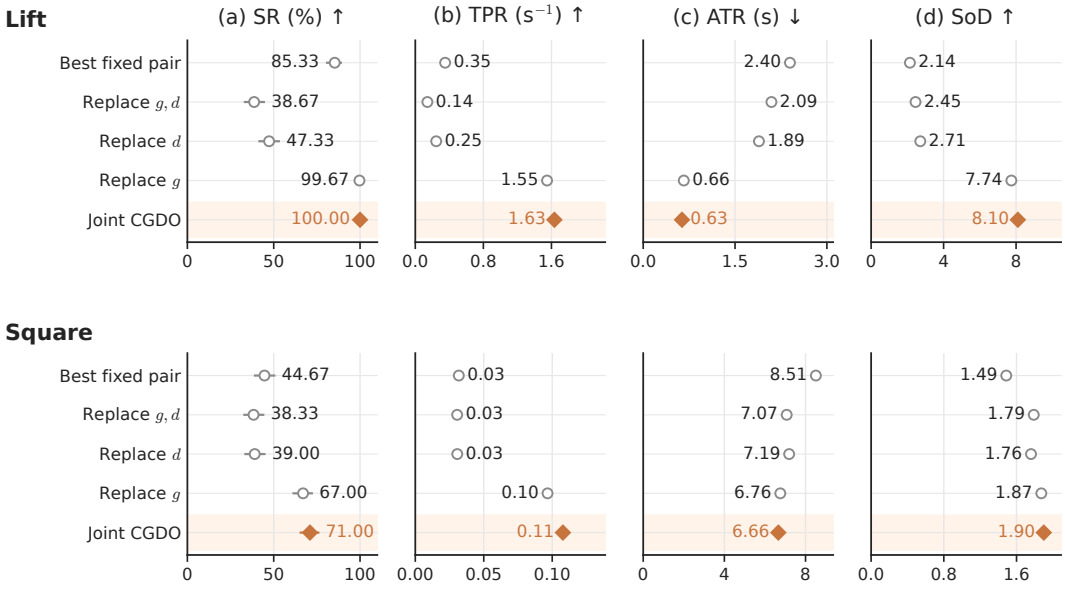


Figure 6: **Replacing learned gain and duration with fixed averages.** Duration replacement causes the larger losses. SR bars are 95% Wilson intervals. The unmodified study checkpoints achieve 100% SR/0.63 s ATR on `Lift` and 71%/6.66 s on `Square`.

Replacing gain reduces `Lift` TPR by 0.09 s^{-1} (Holm $p < 0.001$); its `Square` effect is unresolved ($p = 0.20$). Adaptive duration is the dominant contributor in these frozen-controller interventions.

Tracking and smoothness. On `Lift`, joint learning completes the task faster despite larger intermediate tracking errors than fixed or gain-only execution. SPARC and negative-LDLJ indicate smoother motion on `Lift`, with mixed outcomes on `Square`. We use 300 trajectories per condition, averaging tracking error over integration calls and measuring smoothness on positions resampled at 100 Hz.

D Additional Policy Interfaces and Real-Robot Protocol

D.1 Commanded-Pose Policies, ACT, and RTC

Commanded-pose policies. CGDO also accelerates DP trained on `RoboMimic`'s original commanded-pose labels (Table 15). These policies retain the original relative commands, normalization, and decoder, without reached-pose relabeling. On `Square`, SR improves by 14.33 percentage points (95% paired CI $[7.33, 21.33]$), and completion is 6.90 s faster ($[6.37, 7.43]$) on the 143 episodes where both variants succeed. The speedup therefore remains when comparing the same successful episodes.

Comparison settings. Commanded-pose DP predicts and executes eight actions; ACT predicts 32 and executes the first eight. Each comparison uses the same frozen base policy, $20\times$ actuator limits, and 300 paired initial states, with fixed execution at $(g, \tilde{d}) = (150, 25)$. Commanded-pose evaluation horizons are 5, 15, and 20 s for `Lift`, `Can`, and `Square`, respectively.

Table 15: **DP with commanded-pose labels.** Successful completion is 2.38–5.99× faster with CGDO. Metrics follow Table 1. Dashes indicate unavailable SoD because the demonstration-time references have not been verified.

	Lift				Can				Square			
Method	SR↑	TPR↑	ATR↓	SoD↑	SR↑	TPR↑	ATR↓	SoD↑	SR↑	TPR↑	ATR↓	SoD↑
Fixed settings	0.57	0.08	3.74	—	0.89	0.10	8.93	—	0.60	0.03	11.96	—
CGDO	**1.00**	**1.64**	**0.63**	—	**0.98**	**0.51**	**1.97**	—	**0.75**	**0.15**	**5.03**	—

RTC in simulation. RTC (Black et al., 2025) keeps two actions committed while the next policy and execution-policy predictions are computed, advancing by six actions per eight-action chunk. Committed actions retain their selected gain and duration. Pending controls start at $(g, \tilde{d}) = (3000, 5)$ and are included in the execution-policy observation. Evaluation measures elapsed time with overlapping inference and motion counted once. RTC checkpoints are selected on 100 development episodes by SR, lower ATR, then return, and evaluated on 300 held-out episodes with deterministic execution-policy outputs.

D.2 Real-Robot Protocol

All three tasks use a Piper arm, third-person and wrist cameras, and a frozen visual DP with eight-action chunks of absolute joint/gripper targets and eight denoising steps. Base Policy executes each action for 50 ms; CGDO selects durations within 1–50 ms. Table 16 collects the task-specific settings. `Pick Bread` and `Nest Cups` use convolutional encoders; `Stack Block` uses a frozen DINOv3 ViT-B/16 encoder.

Table 16: Real-robot evaluation settings. Trials are per method; the demo column is the verified mean full-demonstration duration used for SoD.

Task	Execution	Trials	Online training episodes	Demo (s)
`Pick Bread`	RTC	10	86	13.814
`Nest Cups`	Synchronous	20	60	11.839
`Stack Block`	Synchronous	20	20	11.232

Evaluation uses deterministic execution-policy outputs without learning updates. Success is operator-marked, and completion time runs from the first action to the accepted success event. This includes communication and inference, but excludes preparation and reset.

E Physical-Time Learning

A sufficient decision state includes robot/environment state, controller and policy memory, the proposed chunk, remaining time for terminal horizons, and pending work for asynchronous execution. The kernel jointly samples (x', Δ) given (x, u); early success shortens Δ and terminates the transition. The implementation approximates this state using the features in Appendix A.

Partitioning the time-cost integral in Equation 4 gives

$$R(\Delta_1 + \Delta_2) = R(\Delta_1) + e^{-\beta\Delta_1} R(\Delta_2), \qquad Q^\pi(x, u) = \mathbb{E}\left[R(\Delta) + be^{-\beta\Delta} V^\pi(x') \mid x, u\right]. \quad (11)$$

Success and terminal deadlines set $b = 0$; artificial continuing-task cutoffs retain $b = 1$. Standard arm training bootstraps at timeout, unlike `Adroit` and the alternative configuration for DSRL on `Can`. All evaluations count unfinished episodes as failures at the deadline. Entropy regularization occurs at decision boundaries and is separate from this additive physical-time cost.

For actual elapsed ticks $D = \Delta/\delta$, the implementation uses

$$R = -\lambda \frac{1 - \gamma_\delta^D}{1 - \gamma_\delta}, \qquad \beta = -\frac{\log \gamma_\delta}{\delta}, \qquad c = \frac{\lambda\beta}{1 - \gamma_\delta}, \qquad e^{-\beta\Delta} = \gamma_\delta^D. \quad (12)$$

These identities include fractional ticks. For $\gamma_\delta = 1$, $R = -\lambda D$ and $c = \lambda/\delta$. The undiscounted objective minimizes expected capped time to success; the discounted objective minimizes its exponentially weighted time cost.

F Response Dynamics and Gain–Duration Time Scales

F.1 Conditional Time-Scaling Relation

For unit-mass critically damped dynamics with held target r, write $\omega = \sqrt{g}$:

$$\ddot{x} + 2\omega\dot{x} = \omega^2(r - x), \qquad x(t) = r + [-e_0 + (v_0 - \omega e_0)t]e^{-\omega t}, \tag{13}$$

where $e_0 = r - x_0$. In normalized time $\vartheta = \omega t$, the dynamics depend on the ordered targets, initial position, normalized velocity v_0/ω, and holding duration ωd. Thus $g_b = b^2 g$, $d_b = d/b$, and $v_{0,b} = bv_0$ give $x_b(t) = x_1(bt)$ under this model, even for non-collinear targets. From rest, speed peaks at $1/\omega$; from a moving state, higher gain can instead produce stronger braking.

For the more general model $m\ddot{x} = g(r - x) - K_d\dot{x} + F_e(x, \dot{x})$, similarity additionally requires

$$K_{d,b} = bK_{d,1}, \qquad F_{e,b}(x, bv) = b^2 F_{e,1}(x, v), \tag{14}$$

and feasible actuator output. Unscaled environmental time constants, saturation, or unmatched initial velocity can therefore break the rule, motivating state-dependent gain–duration adaptation.

F.2 Evidence for Coordinated Time Scaling

We test the time-scaling relation in `Lift` using shared commanded-pose sequences and reset states (Table 17). Comparisons use 2 ms integration, native damping $K_d = 2\sqrt{g}$, and 20 resets per condition, starting from rest except in the moving-state tests. Main-figure examples use the smallest seed in each comparison.

Table 17: Settings for the response and time-scaling comparisons. Target sequences and reset states are shared across the compared execution settings.

Comparison	Commanded targets	Varied settings
Single-target response	2 cm Cartesian step	$g \in \{150, 600, 1725, 3300\}$
Earlier switching	Eight targets along a line or arc, spaced 1 cm apart	$d \in \{10, 50, 100\}$ ms
Coordinated scaling	Eight targets along a line or arc	$(g, d) = (150, 40\,\text{ms}), (600, 20\,\text{ms}), (2400, 10\,\text{ms})$
Contact time scale	Eight downward arc targets, radius 0.10 m	Same gain–duration pairs; no contact, fixed contact, or scaled contact; $\tau_c \in \{40, 80\}$ ms

Gain changes the response; duration sets the next switch. For motion from rest in $+x$, increasing gain from 150 to 3300 brings peak speed forward from 78 to 16 ms. Shorter duration instead switches to the next target before the robot slows, as confirmed by switch-versus-hold comparisons from the same state. Gain also affects braking: across 60 moving-state starts from 20 policy episodes, higher gain produces stronger mean deceleration.

Path preservation does not require reaching every target. Keeping $\sqrt{g}\,d$ fixed gives phase-aligned position RMS differences of 0.064–0.179 mm and mean peak-speed ratios of $1 : 2.011 : 4.028$. No intermediate target is reached within 2 mm while active; convergence occurs during the separate 1 s final-target hold. Thus, coordinated changes can preserve the path without reproducing a sequence of attained waypoints.

In Figure 2c, mean geometric RMS error is 1.706 mm for coordinated changes, versus 4.359 mm for duration alone and 5.048 mm for gain alone. Unlike phase-aligned error, this metric includes overshoot, using 101 normalized arc-length positions over the full path.

Contact introduces a separate time scale. An unchanged contact model breaks the gain–duration scaling relation. Scaling its time constant from τ_c to τ_c/b substantially restores agreement (Table 18), scaling the solver's reference damping and stiffness by b and b^2 as required by Equation 14.

Table 18: At $b = 4$, scaling the contact time constant restores approximate phase-aligned motion. Errors are hold-phase position RMS; recovery intervals resample 20 paired resets.

Base τ_c	Fixed contact (mm)	Scaled contact (mm)	Recovery [95% CI]
40 ms	6.860	0.483	6.377 [6.343, 6.411]
80 ms	5.783	0.481	5.302 [5.236, 5.358]

The improvement holds on all 20 resets for $b = 2, 4$ and persists at 1 ms integration on five paired resets. To isolate contact from actuator saturation, this simulator test uses frictionless soft contact and $20\times$ native torque bounds; normal forces reach 1314 N during the final-target hold.

Torque limits and initial velocity also constrain time scaling. At native torque bounds and $b = 4$, straight-path phase RMS rises from 0.368 to 20.124 mm (2 cm target spacing); the arc also deviates more over its shared traveled-length prefix. Mild clipping need not cause a measurable failure.

Keeping the same forward initial velocity across gains gives 1.161 mm command-phase RMS, versus 0.179 mm from rest. Scaling the arm's initial velocities reduces it to 0.171 mm. Transverse motion improves on average, but some trajectories do not improve. Execution settings therefore need to account for the current motion as well as the target.