

Retaining State Corrections in Recurrent Control

Dario Fumarola
Amazon Web Services
New York, New York

Abstract

Local optimization can improve a recurrent controller’s action while leaving the memory that generated it unchanged. We examine what carries over to later decisions when a frozen controller retains a bounded Gauss–Newton correction to its state. Paired simulator branches execute the same corrected action, then differ in the state and plan they store. The retained correction improves proposal loss over the next few inputs and reduces subsequent optimization work. With the corrected plan cached in both branches, state retention lowers eight-input realized cost by a further 0.7% in navigation and 1.1% in constrained chain control. Full-episode navigation success rises from 82.7% with discarded corrections to 85.4% with retention; warm-started model predictive control reaches 85.8% and has substantially fewer contacts. Fitting an action-space solution back into memory yields similar success, while rolling back the latest correction on a target change or stale observation helps during sensing delays. The results suggest a short-lived complement to plan reuse, with the clearest benefit under tight solve budgets.

1 Introduction

A local planner often repairs a neural controller’s proposed action, only to face much the same proposal at the next observation. In a recurrent controller, this can happen because the repair never reaches the memory passed between inputs. We study the consequences of writing a bounded correction into that memory.

Consider a frozen recurrent model that decodes its hidden state into a short control sequence. After optimizing the state, the controller executes the improved first action and must decide what to store. It can save either the original proposal state or the corrected one. The immediate action is identical in both cases; the next recurrent update is where the two choices begin to matter.

We initially expected a cached action sequence to explain most of the benefit. An action-space optimizer can shift its previous solution, warm-start the next solve, and search beyond the trajectories accessible through a learned decoder. Warm-started model predictive control (MPC) is therefore a demanding comparison [1, 2]. There is also a reason to be cautious about state retention: a correction for an obstacle that remains nearby may help at the next input, whereas a correction aimed at a superseded target may persist too long.

We use two complementary evaluations. Full episodes compare control quality at matched latency. At selected checkpoints, we clone the simulator and controller, execute one common corrected action, and vary state storage and plan caching independently. These branches let us follow the memory intervention through identical recorded inputs, through physical closed-loop continuation, and through a shared action-space solver.

The experiments cover navigation and a redundant articulated chain moving among obstacles. Both use frozen recurrent and decoder weights, a damped Gauss–Newton update, and fixed limits on displacement and candidate evaluations.

We also fit action-space solutions back into the recurrent state and test a one-input rollback rule for changed targets or stale sensing. The episode comparisons report completion, contacts, and latency alongside the branching measurements.

2 Correcting a Recurrent Controller

2.1 A common objective in action coordinates

At input t , the controller receives observation x_t , including the action actually executed at $t - 1$. It proposes

$$z_t^0 = f_\theta(z_{t-1}, x_t), \quad U_t^0 = g_\phi(z_t^0; c_t), \quad (1)$$

where $z_t^0 \in \mathbb{R}^d$ is the complete persistent state, $U_t^0 = (u_{t,0}, \dots, u_{t,H-1})$ is an H -step control sequence, and c_t contains the current physical state estimate, target, and local geometry. The decoder includes projection onto $\mathcal{U}_t$, the task’s convex set of permitted control sequences. Only $u_{t,0}$ is executed.

Let $r_t(U; c_t)$ collect the weighted task residuals from a rollout under the local model. We use

$$C_t(U) = \frac{1}{2} \|r_t(U; c_t)\|_2^2, \quad (2)$$

$$F_t(U) = C_t(U) + \frac{\mu}{2m} \|S(U - U_t^0)\|_2^2, \quad (3)$$

where m is the number of scalar controls and the diagonal matrix S divides each control by its physical limit. All quantities in F_t are held fixed during the solve, including the anchor U_t^0 .

Action refinement minimizes $F_t(U)$ directly. State correction minimizes

$$L_t(z) = F_t(g_\phi(z; c_t)). \quad (4)$$

Both use the same residuals, dynamics, physical limits, proximity weight, and cumulative action-change limit

$$\|S(U - U_t^0)\|_2 / \sqrt{m} \leq \rho_U. \quad (5)$$

Algorithm 1 One state-correction call

Require: z_t^0, c_t ; trial limit K ; radii ρ_z, ρ_U

- 1: $z \leftarrow z_t^0$; evaluate $L_t(z)$; mark direction stale
- 2: **for** $j = 1, \dots, K$ **do**
- 3: **if** direction is stale **then**
- 4: Form b, P, v by eq. (6); mark direction fresh
- 5: Stop if non-finite or $-\eta b^\top v \leq \epsilon_t$
- 6: Set largest $\alpha \in [0, 1]$ within the state ball
- 7: **end if**
- 8: $z' \leftarrow z + \alpha v$; decode and evaluate candidate
- 9: Stop if the candidate is non-finite or $\alpha = 0$
- 10: **if** candidate passes eqs. (5) and (7) **then**
- 11: $z \leftarrow z'$; mark direction stale
- 12: Stop if the accepted decrease is below tolerance
- 13: **else**
- 14: $\alpha \leftarrow \alpha/2$; retain the current direction
- 15: **end if**
- 16: **end for**
- 17: **return** $\tilde{z}_t = z$ and $\tilde{U}_t = g_\phi(z; c_t)$

State correction also imposes $\|z - z_t^0\|_2 \leq \rho_z$ to limit the change written into memory. The proximity penalty remains entirely in action coordinates. Because the state ball and learned decoder restrict the available trajectories, we compare immediate objective reduction for both search spaces.

The state coordinates are standardized once using training-set statistics. The recurrence and decoder incorporate the inverse transformation, preserving the trained controller’s function. Projection, rollout, and residual evaluation are shared by all correction methods. Their derivatives use the active branch at a projection boundary.

2.2 A bounded local update

Write $L_t(z) = \|R_t(z)\|_2^2/2$, including the action-proximity residual. At the current accepted state z , form

$$b = J^\top R_t(z), \quad P = J^\top J + \lambda I, \quad v = -P^{-1}b, \quad (6)$$

where $J = \partial R_t(z)/\partial z$ and $\lambda > 0$. We solve the current linearization exactly using an output-space factorization whose inner system has the decoder’s output dimension (section A).

Starting with $\alpha \leq 1$, choose a candidate $z' = z + \alpha v$ within the cumulative state ball. Accept it only if its decoded sequence satisfies eq. (5) and

$$L_t(z) - L_t(z') \geq -\eta \alpha b^\top v > \epsilon_t. \quad (7)$$

A rejected candidate consumes a trial. The next trial halves α and reuses the direction. After acceptance, a remaining trial may compute a new direction. Non-finite quantities end the solve. The reference configuration allows three trials per input.

The update combines damped Gauss–Newton with an Armijo decrease test [3]. Accepted steps decrease the evaluated loss and stay inside the total displacement limit; section A.3 gives the proof and evaluation count.

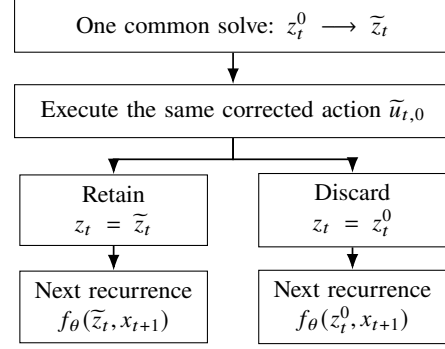


Figure 1. The memory intervention. Both branches begin with the same physical state and execute the same action. Their first subsequent observation is therefore also the same. Later actions can differ because only the stored recurrent state differs.

2.3 State storage

The two state-correction controllers execute the same rule, differing only in the stored state:

$$\begin{aligned} \text{discard: } u_t &= \tilde{u}_{t,0}, \quad z_t = z_t^0, \\ \text{retain: } u_t &= \tilde{u}_{t,0}, \quad z_t = \tilde{z}_t. \end{aligned} \quad (8)$$

Starting from a common proposal, the two controllers produce identical actions. Across a full episode, their proposals can diverge as the stored states pass through the recurrence. Both receive the action actually executed as part of the next observation.

We also test *action refinement with state fitting*. First solve directly for U_t^* . Then fit a nearby state to that solution:

$$\min_{\|z - z_t^0\| \leq \rho_z} \frac{1}{2m} \|S(g_\phi(z; c_t) - U_t^*)\|_2^2. \quad (9)$$

We allow one damped Gauss–Newton direction and at most two candidate evaluations. The controller executes the first action of U_t^* and uses the fitted state at the next input, with fitting charged to the same total latency budget as action refinement. This provides a second way to write an optimized solution into memory.

2.4 Rolling back an outdated correction

The retain controller always passes $\tilde{z}_t$ to the next recurrence. A second variant keeps z_t^0 for one additional input, so that the latest correction can be undone before it is used. On receiving x_{t+1} , it sets

$$\hat{z}_t = z_t^0 + a_{t+1}(\tilde{z}_t - z_t^0), \quad a_{t+1} \in \{0, 1\}, \quad (10)$$

and computes $z_{t+1}^0 = f_\theta(\hat{z}_t, x_{t+1})$. The switch is zero if the active target identifier changes or the relevant sensor record is more than one input old; otherwise it is one. The action already executed at t is unchanged.

Target identifiers and sensor timestamps are available to every controller. The rule, which we call *context rollback*, adds one saved state vector and acts on the latest correction. Earlier recurrent history is carried forward. We evaluate it separately from unconditional retention.

3 Related Work

MPC repeatedly optimizes a finite control horizon and applies its first input. Real-time iteration exploits the similarity of successive problems through warm starts and bounded work per update [1]. Schwenkel et al. [2] additionally store earlier optimization data to construct improved warm starts, with stability and asymptotic learning results under their assumptions. Differentiable MPC combines optimization with learned models and objectives [4]. Our cache experiment uses a single shifted plan and compares it with storage inside the recurrent policy after a common solve.

Safety filters intervene when a proposed control conflicts with constraints. Control-barrier-function filters express this intervention as a quadratic program [5], while predictive safety filters use a model and backup policy to reason over future constraint satisfaction [6]. Multi-step filtering can reduce repeated interventions and chattering while leaving the nominal controller’s memory unchanged [7]. We include a multi-step predictive filter for this reason and allow action refinement and MPC to reuse their optimized control sequences.

Inference-time activation updates have also been used to steer frozen models. Plug and Play Language Models differentiate an auxiliary objective through hidden activations [8]. In Test-Time Training layers, a learned model serves as the recurrent hidden state, and its update procedure is trained along with the surrounding system [9]. We work with an already trained recurrent architecture and follow a bounded activation update through subsequent control decisions, using common-input replay and physical continuations to measure its persistence.

4 Experimental Design

4.1 Tasks

Navigation. A radius-0.12 m agent moves through 10×10 m mazes with narrow doorways and six to twelve moving circles. It observes a local wall-distance grid, visible obstacle positions and estimated velocities, its velocity, the relative goal, and its previous action. A common waypoint process supplies local route information to every controller. The policy predicts 12 two-dimensional accelerations at 0.2 s intervals. The local objective penalizes waypoint error, small clearance, control effort, and abrupt control changes.

Success means reaching within 0.35 m of the goal in at most 160 inputs. We count contacts from swept physical motion and report contact-containing steps per 10^3 executed steps. Time to goal is capped at 160 for failed episodes. A separate stress test delays obstacle observations by two inputs while retaining current agent proprioception.

Constrained chain control. A planar fixed-base chain has 12 links of length 0.18 m. Its input contains current joint angles, the previous executed joint velocities, four circular obstacles, and a two-dimensional end-effector target. The decoder predicts eight joint-velocity vectors at 0.1 s intervals.

Joint angles and velocities are bounded, large velocity

changes are penalized, and motion must avoid obstacles and nonadjacent links. We generate targets from collision-free configurations and supply only their endpoints to the controller. The redundancy leaves several possible inverse-kinematic solutions, so reaching a target requires choosing a configuration together with a feasible path from the current chain.

An episode contains three targets, each held for 40 inputs. Success requires reaching within 0.04 m of every target for five consecutive inputs before it changes, with no contact during the episode. Endpoint error is averaged over the final five inputs of each target interval. Geometry, costs, and generation rules are specified in section B.

4.2 Controllers and compute matching

All learned-proposal methods share a frozen 256-state recurrent model and a two-layer decoder within each training seed. Action refinement keeps the best of the current neural proposal and the projected, shifted solution from the previous input as its initial candidate. The extra shifted-plan check consumes a candidate slot; the neural-proposal evaluation is already required by both correction methods. The shifted solution is projected into the same action-change region around U_t^0 .

The MPC baseline shares the horizon, observations, dynamics, and task costs. It optimizes without the correction-specific proximity penalty or action-change radius, imposes predicted clearance constraints, and warm-starts both the control sequence and constraint multipliers. The current neural proposal supplies an alternative initial trajectory.

Navigation also includes a control-barrier-function (CBF) filter on the first action and a multi-step predictive filter. The latter minimizes deviation from the full nominal sequence subject to predicted clearance, actuator limits, and a terminal braking condition. These filters use the same observations and local model as the other controllers. Their formulations and infeasibility handling are given in section C.

We select configurations on validation episodes under limits of 4 ms mean controller time and 8 ms p99, tuning solver iteration limits, damping, and stopping tolerances for each method. Retain and discard share every numerical setting. The latency sweep adds other envelopes, each with a p99 target twice the mean target.

Neural inference and its derivatives run in FP32 on an NVIDIA A100-SXM4 80 GB. MPC and predictive filtering use compiled CPU solvers on the same workstation, an AMD EPYC 7543 with one solver thread. Timing includes controller-side transfers, warm-start checks, derivatives, failed candidates, state fitting, and output synchronization. Common sensor preprocessing and simulator advancement are excluded. This compares elapsed controller time on one CPU/GPU system; solver work counts are reported separately.

4.3 Sampling and paired comparisons

Navigation uses 40,000 training and 4,000 validation episodes; chain control uses 30,000 and 3,000. We test five independently trained recurrent models on 2,000 navigation and 1,500

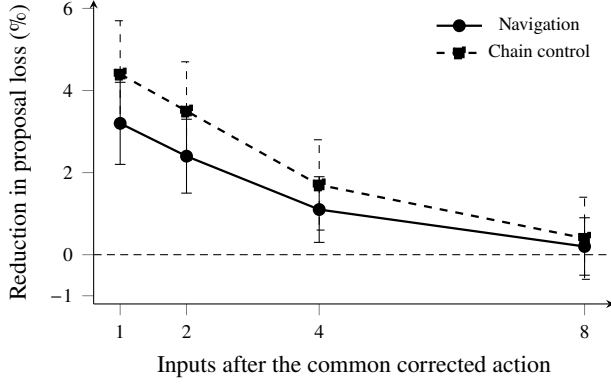


Figure 2. The direct memory effect decays over a few inputs. Both branches receive identical recorded observations, including previous-action features, and neither performs further correction. Error bars are paired 95% intervals.

chain episodes each. Within a seed, all methods share initial conditions and disturbance streams. Fitting and model selection use the training and validation splits.

Intervals use 10,000 paired hierarchical bootstrap replicates, resampling trained models and then their episodes. Entries $a \pm b$ show the estimate and the larger rounded half-width of the 95% interval. Differences central to the argument are reported with their paired intervals and per-model effects in table 5. Timing uses five full-stream repetitions after 1,000 warm-up inputs; the reported value is the median repetition statistic.

5 Results

5.1 Persistence after a shared action

For each trained model, we select 800 held-out episodes and one predetermined checkpoint per episode. At that checkpoint, a single common state-correction call supplies the action for two cloned controller-simulator branches. One stores the corrected state and the other stores the proposal (fig. 1). The sample includes checkpoints where the solve accepts no correction, giving 4,000 paired checkpoints per task across the five models.

We first replay the same next eight observations and task contexts to both branches with correction disabled. This holds sensing fixed while the two recurrent states evolve. The task-loss reduction at horizon h is

$$B_h = 100 \frac{\bar{C}_h^{\text{discard}} - \bar{C}_h^{\text{retain}}}{\max\{\bar{C}_h^{\text{discard}}, 10^{-6}\}}, \quad (11)$$

where C_h excludes the proximity penalty and the bars average checkpoints before taking the ratio.

The next proposal’s loss is 3.2% lower in navigation and 4.4% lower in chain control after state retention (fig. 2). Much of the effect fades over the first four inputs, when the reductions have fallen to 1.1% and 1.7%. By input eight, both estimates are close to zero and their intervals span zero.

We then let the two physical simulators advance independently for eight inputs, still with correction disabled. Once their actions diverge, each controller receives its own observations; exogenous disturbances remain paired by absolute time. Retention lowers realized stage cost, including contact and terminal-distance penalties, by 2.0% in navigation, [0.8, 3.2], and 3.0% in chain control, [1.4, 4.6].

When both branches resume the same correction rule, the realized-cost reductions are 1.0% in navigation, [0.2, 1.8], and 1.6% in chain control, [0.4, 2.8]. The larger change is in solver work. Over the next eight inputs, navigation uses 9.3 residual-Jacobian evaluations after retention and 12.1 after discard; the chain uses 10.8 and 14.2. These correspond to paired reductions of 23.1%, [17.0, 29.0], and 23.9%, [17.5, 30.0].

5.2 State retention with a matched plan cache

To measure carryover beyond a warm start, we cross two choices at the same checkpoints: retaining or discarding $\tilde{z}_t$, and caching or discarding $\tilde{U}_t$. All four branches execute the common solve’s first action. The plan-only branch stores z_t^0 together with the full corrected plan.

Over the next eight inputs, all four branches use the same action-space Gauss-Newton solver, with state correction and fitting disabled. Where a cache is present, the solver shifts the plan, projects it into the current actuator set, and compares it with the fresh neural proposal. The better candidate initializes the solve, and the optimized plan replaces the cache. Each cache check consumes one of three candidate slots and is included in timing.

The downstream solver minimizes C_t under actuator constraints, with the proposal-centered penalty and action-change radius removed. For a given physical context, the branches share the objective, feasible set, and numerical settings; their stored state and plan affect initialization.

A plan cache alone lowers cost by 1.1% in navigation and 1.7% in chain control (table 1). With that corrected plan in both caches, retaining the state gives a further reduction of 0.7%, [0.2, 1.2], and 1.1%, [0.3, 1.9], respectively, measured relative to the plan-only branch. The navigation effect ranges from −0.1% to 1.5% across trained models, with four of five positive; all five chain effects are positive (table 5).

5.3 Navigation performance and solve budget

In full navigation episodes, retention raises success from 82.7% to 85.4%, a paired gain of 2.7 percentage points, [1.7, 3.7] (table 2). The current optimization rule is shared. Contacts fall from 18.2 to 14.2 per thousand steps, and capped time falls by 5.6 inputs.

Action refinement reaches 83.9% success and also makes the stronger immediate correction. At 4,000 common checkpoints, three candidate trials reduce the current objective by $30.8 \pm 1.0\%$ in action space and $27.6 \pm 0.9\%$ in state space. Both searches start from the neural proposal with their plan caches cleared, and the retain and discard variants coincide at these checkpoints.

We also tried fitting the action-space solution back into

Table 1. Eight-input continuation after a common corrected action. Cost reduction is relative to storing z_t^0 with no plan cache; positive values indicate lower physical cost. Every branch uses the same downstream action-space solver. Latency is per subsequent input, including cache handling, and a cached plan is refreshed after each solve.

Stored state	Plan cache	Navigation		Chain control	
		Cost reduction (%)	Mean / p99 ms	Cost reduction (%)	Mean / p99 ms
z_t^0	Absent	0	3.86 / 7.53	0	3.95 / 7.82
$\tilde{z}_t$	Absent	0.8 ± 0.7	3.72 / 7.39	1.4 ± 1.0	3.83 / 7.64
z_t^0	Present	1.1 ± 0.7	3.64 / 7.25	1.7 ± 1.1	3.79 / 7.56
$\tilde{z}_t$	Present	1.8 ± 0.8	3.55 / 7.18	2.8 ± 1.2	3.70 / 7.43

Table 2. Navigation under the 4 ms mean, 8 ms p99 validation envelope. Controllers are grouped by method family. Each receives the same observations and 12-step model; retain and discard also share solver settings. Capped time is measured in inputs, with failures assigned 160. Lower contacts, time, and latency are better.

Controller	Success (%)	Contacts/ 10^3	Capped time	Mean ms	p99 ms
Frozen recurrent policy	68.4 ± 1.9	37.1 ± 2.3	115.3 ± 2.8	2.10	2.70
<i>Filtering and constrained planning</i>					
CBF filter	79.2 ± 1.6	6.4 ± 0.8	100.5 ± 2.4	2.58	3.24
Multi-step predictive filter	82.1 ± 1.5	4.1 ± 0.6	95.0 ± 2.3	3.92	7.62
Warm-started MPC	85.8 ± 1.3	6.0 ± 0.7	87.0 ± 2.1	3.93	7.71
<i>Local correction of the frozen policy</i>					
State correction, discard	82.7 ± 1.5	18.2 ± 1.2	92.0 ± 2.3	3.81	7.43
State correction, retain	85.4 ± 1.3	14.2 ± 1.0	86.4 ± 2.1	3.79	7.28
Action refinement	83.9 ± 1.4	15.9 ± 1.1	89.6 ± 2.2	3.82	7.31
Action refinement + state fitting	85.2 ± 1.3	14.4 ± 1.1	87.2 ± 2.1	3.91	7.64

memory. Success reaches 85.2%, a paired gain of 1.3 points over action refinement, [0.5, 2.1], after charging fitting to the same latency budget. Direct state retention is ahead by 0.2 points, with interval $[-0.5, 0.9]$, so this comparison gives little basis for preferring either route into memory.

MPC reaches 85.8% success, 0.4 points above retained state correction, $[-0.4, 1.2]$. Its contact rate is 6.0 per thousand steps, compared with 14.2 for retention, a reduction of about 58%. The predictive filter has the fewest contacts at 4.1, although its capped time is 95.0 inputs and success is 82.1%. Retention has the lowest capped-time estimate among these three controllers, at 86.4 inputs, narrowly ahead of MPC’s 87.0.

The ordering changes with the solve budget (fig. 3). At about 3 ms, retained state correction reaches 81.8%, action refinement 80.6%, and MPC 80.8%. The retention–action-refinement gap grows to 1.5 points near 4 ms, then narrows to 0.6 points near 8 ms. At that last operating point, MPC reaches 89.0% and retention reaches 87.0%.

5.4 Constrained chain control

Retention raises chain success from 74.2% to 78.6%, a paired gain of 4.4 points, [3.1, 5.7] (table 3). State fitting reaches 78.1% with endpoint error of 4.5 cm, close to retention’s 4.4 cm. MPC has the highest success estimate at 79.4% and the lowest contact rate, though its mean and p99 latencies are also the highest.

The immediate-objective test again favors action search: at

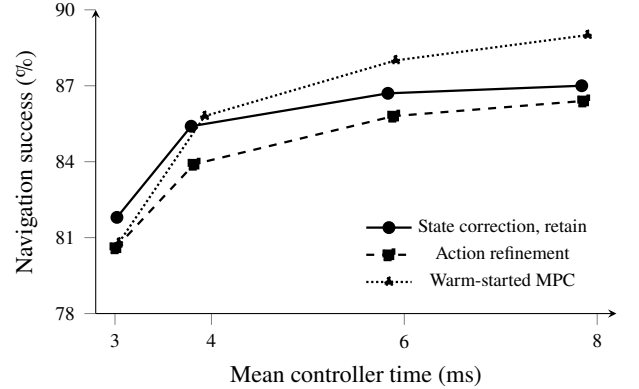


Figure 3. Navigation across latency envelopes. Each point is a separately validation-selected configuration; lines connect operating points. The retention–action-refinement gap is largest around 4 ms, while MPC gains more from additional solve time. Main operating-point intervals are in table 2.

common checkpoints it reduces the objective by $34.9 \pm 1.2\%$, compared with $29.7 \pm 1.1\%$ for state search. Across full episodes, action refinement has lower success than either method that writes an optimized solution into memory.

5.5 Target changes and stale sensing

Replacing the target immediately after the common action changes the sign of the estimated retention benefit (table 4, left). The navigation interval includes zero; for the chain,

Table 3. Constrained chain control under the same latency envelope. Success requires reaching all three targets without contact. Endpoint error averages the final five inputs of every target interval, including unsuccessful intervals. Lower endpoint error, contacts, and latency are better.

Controller	Success (%)	Endpoint error (cm)	Contacts/ 10^3	Mean ms	p99 ms
Frozen recurrent policy	58.6 ± 2.0	9.8 ± 0.5	24.6 ± 1.8	1.93	2.62
Warm-started MPC	79.4 ± 1.5	4.1 ± 0.3	4.8 ± 0.7	3.97	7.85
<i>Local correction of the frozen policy</i>					
State correction, discard	74.2 ± 1.7	5.6 ± 0.3	11.1 ± 1.1	3.91	7.73
State correction, retain	78.6 ± 1.5	4.4 ± 0.3	8.1 ± 0.9	3.88	7.58
Action refinement	76.0 ± 1.6	5.1 ± 0.3	9.7 ± 1.0	3.86	7.51
Action refinement + state fitting	78.1 ± 1.5	4.5 ± 0.3	8.3 ± 0.9	3.96	7.83

Table 4. Changed context and selective rollback. Left: effect of one retained correction on eight-input physical cost, with further optimization disabled; positive means lower cost than discard. Right: full-episode success when sensing is intermittently delayed. Context rollback uses only the target identifier and record age, leaving the current action solver unchanged.

One-shot intervention: cost reduction (%)			Delay bursts: episode success (%)		
Continuation	Navigation	Chain	Memory policy	Navigation	Chain
Current sensing, fixed target	2.0 ± 1.2	3.0 ± 1.6	Discard	81.9 ± 1.5	70.9 ± 1.8
Target replaced at next input	-0.8 ± 0.9	-1.9 ± 1.1	Always retain	82.1 ± 1.5	71.8 ± 1.7
Records delayed by two inputs	-0.5 ± 1.0	-1.4 ± 1.2	Context rollback	83.4 ± 1.4	73.0 ± 1.7

retention increases subsequent cost by 1.9%, with interval [0.8, 3.0]. Both branches receive the new target at the next input.

Under continuous two-input obstacle-observation delay, navigation success falls to 70.4% for retention and 70.9% for discard, from nominal values of 85.4% and 82.7%. Candidate acceptance remains high, moving from 88% to 86%. Thus a solve can continue to reduce its supplied objective while the outdated geometry leads to worse physical outcomes.

Context rollback (eq. (10)) withholds the latest correction when a target identifier changes or the relevant sensor record becomes stale. We test full episodes with two eight-input bursts of two-input delay: navigation bursts begin at inputs 48 and 96, and chain bursts at 48 and 88. Targets, success criteria, and the per-input state-correction solver are shared by all three memory policies.

During these bursts, rollback raises navigation success from 82.1% to 83.4%, a paired gain of 1.3 points, [0.5, 2.1]. The chain gain is 1.2 points, [0.4, 2.0] (table 4, right). It adds at most 0.02 ms to mean time, and both timing limits remain satisfied. On nominal episodes, rollback yields 85.3% navigation success and 79.0% chain success; unconditional retention yields 85.4% and 78.6%.

6 Discussion

The matched-cache experiment places state correction in a fairly narrow role: a short-lived complement to plan reuse. Retaining a correction changes the next neural proposal before any further solver work, and the remaining benefit with both caches enabled is small. It may still be useful when each input permits only a few candidate evaluations. The replay curves put its timescale at several inputs in these two tasks.

Action search reduces the current objective more effectively in both environments. The state-fitting results suggest that a controller can obtain similar downstream performance by solving in action coordinates and then fitting that solution into memory. This adds a decoding fit to the budget, so the balance will depend on how expensive fitting is relative to another action-space step. At the reference envelope, direct retention and fitting have closely matched success estimates.

Rollback addresses a specific failure mode. Saving the pre-correction state for one input lets the controller withdraw its most recent update when a target changes or a timestamp flags stale sensing. Older corrections have already entered the recurrent history, and the current action still depends on the quality of the local model. The gain during delay bursts therefore leaves open how to handle longer periods of model error or gradual changes that the metadata misses.

Contact avoidance also remains a separate concern. MPC and predictive filtering have much lower navigation contact rates, with the filter trading away some completion speed. Our state-update rule guarantees local objective decrease and bounded displacement; collision avoidance still depends on the model and controller design. The experiments cover one gated recurrent architecture and two simulators with differentiable geometry. Other backbones, or training that exposes the policy to state corrections, could change both persistence and the cost of fitting.

7 Conclusion

Retaining state corrections raises nominal success by 2.7 percentage points in navigation and 4.4 points in chain control. With a corrected plan already cached, the additional reduction in eight-input physical cost is 0.7% and 1.1%, respectively.

Most of the proposal-level effect fades over four inputs and is close to zero by eight. The advantage over action refinement is greatest around the 4 ms operating point; by about 8 ms, MPC has the higher navigation success rate. Context rollback improves delay-burst success by 1.3 points in navigation and 1.2 points on the chain.

References

- [1] M. Diehl, H. G. Bock, and J. P. Schlöder. A real-time iteration scheme for nonlinear optimization in optimal feedback control. *SIAM Journal on Control and Optimization*, 43(5):1714–1736, 2005. doi:10.1137/S0363012902400713.
- [2] L. Schwenkel, M. Gharbi, S. Trimpe, and C. Ebenbauer. Online learning with stability guarantees: A memory-based warm starting for real-time MPC. *Automatica*, 122:109247, 2020. doi:10.1016/j.automatica.2020.109247.
- [3] J. Nocedal and S. J. Wright. *Numerical Optimization*. Second edition. Springer, 2006. doi:10.1007/978-0-387-40065-5.
- [4] B. Amos, I. D. Jimenez Rodriguez, J. Sacks, B. Boots, and J. Z. Kolter. Differentiable MPC for end-to-end planning and control. *Advances in Neural Information Processing Systems*, 31, 2018. arXiv:1810.13400.
- [5] A. D. Ames, X. Xu, J. W. Grizzle, and P. Tabuada. Control barrier function based quadratic programs for safety critical systems. *IEEE Transactions on Automatic Control*, 62(8):3861–3876, 2017. doi:10.1109/TAC.2016.2638961.
- [6] K. P. Wabersich and M. N. Zeilinger. A predictive safety filter for learning-based control of constrained nonlinear dynamical systems. *Automatica*, 129:109597, 2021. doi:10.1016/j.automatica.2021.109597.
- [7] F. Pizarro Bejarano, L. Brunke, and A. P. Schoellig. Multi-step model predictive safety filters: Reducing chattering by increasing the prediction horizon. In *IEEE Conference on Decision and Control*, 2023. arXiv:2309.11453.
- [8] S. Dathathri, A. Madotto, J. Lan, J. Hung, E. Frank, P. Molino, J. Yosinski, and R. Liu. Plug and play language models: A simple approach to controlled text generation. In *International Conference on Learning Representations*, 2020. arXiv:1912.02164.
- [9] Y. Sun, X. Li, K. Dalal, J. Xu, A. Vikram, G. Zhang, Y. Dubois, X. Chen, X. Wang, S. Koyejo, T. Hashimoto, and C. Guestrin. Learning to (learn at test time): RNNs with expressive hidden states. In *International Conference on Machine Learning*, PMLR 267:57503–57522, 2025. PMLR 267, sun25h.
- [10] R. Verschueren, G. Frison, D. Kouzoupis, J. Frey, N. van Duijkeren, A. Zanelli, B. Novoselnik, T. Albin, R. Quirynen, and M. Diehl. acados: A modular open-source framework for fast embedded optimal control. *Mathematical Programming Computation*, 14:147–183, 2022. doi:10.1007/s12532-021-00208-8.

A Solver Details and Local Properties

A.1 Exact Gauss–Newton through the output space

At a fixed input, let $D = \partial g_\phi(z; c_t)/\partial z$ and let Q be the residual Jacobian with respect to the physical control sequence, including the action-proximity residual. Then $J = QD$. Set

$$A = (Q^\top Q)^{1/2} D \in \mathbb{R}^{m \times d}. \quad (12)$$

Then $P = \lambda I + A^\top A$, and the inverse-vector product is

$$P^{-1}b = \lambda^{-1} [b - A^\top (\lambda I + AA^\top)^{-1} Ab]. \quad (13)$$

The square root uses a symmetric eigendecomposition, clipping negative round-off eigenvalues to zero, and Cholesky solves the inner system. Here $d = 256$ and $m = 24$ or 96 . Timing includes the derivatives and factorization at every new linearization.

Action refinement solves its constrained quadratic in physical coordinates. Line-search candidates are projected into the actuator set and action-change ball; the decrease prediction uses the actual projected displacement.

A.2 Stopping and cumulative limits

The reference state solver uses $K = 3$, $\lambda = 0.02$, $\rho_z = 0.35$, $\rho_U = 0.20$, $\mu = 0.05$, and $\eta = 0.1$. We set $\epsilon_t = 10^{-6} \max\{1, L_t(z_t^0)\}$ and stop after an accepted decrease below $10^{-3} \max\{1, L_t(z_t^0)\}$.

For displacement $e = z - z_t^0$ and a nonzero direction v , the largest nonnegative step within the state ball is the positive root of

$$(v^\top v)\alpha^2 + 2(e^\top v)\alpha + e^\top e - \rho_z^2 = 0, \quad (14)$$

clipped to $[0, 1]$. We check the action cap on the decoded candidate. A violation consumes a trial and halves the step; a zero step or non-finite candidate ends the solve. The last accepted state is preserved throughout.

State fitting uses the same state ball and damping. A candidate must reduce the fitting loss; failed fitting still counts toward the budget.

A.3 Local decrease and evaluation bounds

Proposition 1. *Suppose objective evaluations are deterministic and finite. Let $\mathcal{A}$ index the accepted steps of Algorithm 1, with directions v_j and step lengths α_j . Then*

$$L_t(\tilde{z}_t) \leq L_t(z_t^0) - \sum_{j \in \mathcal{A}} \eta \alpha_j b_j^\top P_j^{-1} b_j, \quad (15)$$

$$\|\tilde{z}_t - z_t^0\|_2 \leq \rho_z. \quad (16)$$

The call performs at most K candidate evaluations and K residual-Jacobian evaluations, in addition to its initial objective evaluation.

Proof. Positive damping gives $b_j^\top v_j = -b_j^\top P_j^{-1} b_j \leq 0$. Summing the accepted decrease tests proves the first statement. Every accepted state is checked against the same ball around

z_t^0 , proving the second. Each loop slot evaluates at most one candidate and forms at most one Jacobian; a retry reuses the previous Jacobian. Early stopping can only reduce these counts. $\square$

The bound applies to the evaluated loss for one fixed local objective, including at projection boundaries. In implementation, the finite-value checks and ϵ_t threshold handle invalid quantities and round-off.

B Task Specification

B.1 Navigation

A 10×10 m region is split recursively into three to six rectangular rooms, each at least 2 m wide. Split positions are uniform between 40% and 60% of the parent’s longer side. Walls are 0.15 m thick; openings are uniform between 0.65 and 1.05 m. Start and goal lie in different connected rooms at least 5 m apart.

Every controller receives waypoints from the same shortest-path process on the known static room graph. It ignores moving obstacles and supplies the next doorway center, or the goal once the agent enters the final room. Moving-obstacle prediction uses the sensed records described below.

Circle radii are uniform in $[0.12, 0.25]$ m. Each circle remains in its assigned room. With $\Delta t = 0.2$ s, its velocity evolves as

$$w_{i,t+1} = \text{proj}_{\|w\| \leq 0.25} (0.95 w_{i,t} + 0.08 \sqrt{\Delta t} \xi_{i,t}), \quad (17)$$

where $\xi_{i,t} \sim \mathcal{N}(0, I_2)$. The simulator advances positions with this velocity and reflects motion at room boundaries. The controller predicts constant-velocity obstacle motion over its horizon.

The observation includes a 17×17 static wall-distance grid over a 6 m square, up to twelve circle records within 3 m, the relative waypoint and goal, current velocity, and the previous action. Each circle record contains relative position, velocity, radius, and a validity mask. Velocity estimates have independent noise of standard deviation 0.02 m/s per component. Unobserved slots contribute zero residual.

Agent dynamics are

$$v_{h+1} = (1 - \gamma \Delta t) v_h + \Delta t u_h, \quad p_{h+1} = p_h + \Delta t v_{h+1}. \quad (18)$$

The control limits are $|u_h| \leq 1$ m/s² componentwise, and $\gamma = 5/6$ s^{−1}. This drag keeps each velocity component within 1.2 m/s from an initially resting state. Control changes are penalized relative to the actual previous action u_{-1} .

For $H = 12$, define $b_\sigma(a) = \sigma \log(1 + e^{a/\sigma})$ and clearance margin $a_c = 0.15$ m. Let d_{hj} be predicted clearance from the agent’s boundary to object slot j . There are fourteen slots: twelve circles, walls, and the boundary of the sensed square.

With fixed padded normalization,

$$\begin{aligned}
C_t^{\text{nav}}(U) = & \frac{1}{2H} \sum_{h=0}^{H-1} \|(p_{h+1} - p_t^{\text{wp}})/3\|_2^2 \\
& + \frac{10}{28H} \sum_{h=0}^{H-1} \sum_{j=1}^{14} \left[\frac{b_{0.03}(a_c - d_{hj})}{a_c} \right]^2 \\
& + \frac{0.05}{2H} \sum_h \|u_h\|_2^2 + \frac{0.10}{2H} \sum_h \|u_h - u_{h-1}\|_2^2.
\end{aligned} \tag{19}$$

The static grid is bilinearly interpolated. In the physical simulator, swept contact removes motion into the obstacle and keeps the last nonpenetrating position; the episode then continues. The next controller input contains the resulting measured velocity and executed action.

B.2 Constrained chain

Let $q \in \mathbb{R}^{12}$ contain joint angles. With $X_0 = (0, 0)$ and $\ell = 0.18$ m,

$$X_j(q) = \ell \sum_{i=1}^j \begin{bmatrix} \cos(\sum_{k=1}^i q_k) \\ \sin(\sum_{k=1}^i q_k) \end{bmatrix}, \quad j = 1, \dots, 12. \tag{20}$$

Controls are joint velocities: $q_{h+1} = q_h + 0.1u_h$. The convex actuator set imposes $|q_h| \leq 2.6$ rad and $|u_h| \leq 1.2$ rad/s over the horizon. Velocity changes are penalized using 0.3 rad/s as their scale. Links have radius 0.01 m. Contact occurs when a link capsule intersects an obstacle or a nonadjacent link. In delayed-sensing tests, a commanded motion that would cross a physical joint limit is stopped at that limit and counted as a constraint contact.

Four static obstacle centers are sampled in the reachable disk, with radii in $[0.08, 0.16]$ m. Initial and target configurations are drawn from smooth angle sequences in $[-1.8, 1.8]^{12}$ and rejected on collision. The three target endpoints must be at least 0.3 m apart. Before splitting the generated data, an offline planner screens for a collision-free, actuator-feasible transition that reaches each target within 40 inputs. At inference, the controller receives each endpoint; target joint configurations and intermediate paths are withheld.

For $H = 8$, let $e_h = (X_{12}(q_{h+1}) - p_t^*)/0.5$, and let d_{hj} range over 48 link-obstacle and 55 nonadjacent link-pair clearances. The task cost is

$$\begin{aligned}
C_t^{\text{chain}}(U) = & \frac{1}{2H} \sum_h \|e_h\|_2^2 + \frac{1}{2} \|e_{H-1}\|_2^2 \\
& + \frac{8}{206H} \sum_h \sum_{j=1}^{103} \left[\frac{b_{0.01}(0.03 - d_{hj})}{0.03} \right]^2 \\
& + \frac{0.02}{24H} \sum_h \|u_h/1.2\|_2^2 \\
& + \frac{0.05}{24H} \sum_h \|(u_h - u_{h-1})/0.3\|_2^2.
\end{aligned} \tag{21}$$

The minimum segment distance is differentiated on its active closest-point branch. Swept contact checking subdivides each input interval into twenty joint-space interpolation steps; the task’s reported contact definition is this fixed-resolution check. On contact, motion stops at the last clear substep and that episode is marked unsuccessful. Evaluation continues to collect endpoint error and contact counts.

C Baselines, Training, and Evaluation

C.1 Control baselines

Action refinement retains its previous optimized plan and stores the original recurrent proposal state. The retain and discard controllers begin each solve from the current neural proposal, without a separate plan cache. The immediate-loss checkpoint test clears warm starts for every method; section 5.2 describes the four-branch cache intervention.

MPC minimizes C_t subject to the actuator limits and predicted clearance $d_{hj} \geq a_c$ in navigation or $d_{hj} \geq 0.03$ m for the chain. It uses acados [10] with Gauss–Newton cost Hessians and warm-started primal and dual variables. The reference envelope permits three sequential quadratic programming (SQP) iterations and up to three line-search evaluations per iteration. Both the neural and shifted plans are checked before solving.

Predictive filtering uses the same SQP implementation and clearance constraints, with objective

$$\frac{1}{2H} \sum_{h=0}^{H-1} \|u_h - u_{t,h}^0\|_2^2 + \frac{0.05}{2H} \sum_h \|u_h - u_{h-1}\|_2^2, \tag{22}$$

plus terminal $\|v_H\|_\infty \leq 0.03$ m/s. Penalizing deviations over the full sequence follows the multi-step filtering rationale of [7]. Our finite-horizon implementation uses estimated obstacle motion, a braking endpoint, and slack restoration. The robust backup-set construction and guarantees of [6] are outside this implementation.

For the CBF filter, let r and v_r be the relative position and velocity to a visible circle, and let $h = \|r\|_2^2 - R^2$, with R the summed radii plus the margin. The QP minimizes squared deviation from the nominal first acceleration subject to actuator limits and

$$2\|v_r\|_2^2 + 2r^\top(u - \gamma v_r) + 4\omega r^\top v_r + \omega^2 h \geq -\xi, \tag{23}$$

using constant obstacle velocity, $\omega = 2 \text{ s}^{-1}$, and $\xi \geq 0$. Walls use the corresponding linear-distance constraint. The slack penalty is $10^4 \|\xi\|_1$. No unobserved obstacles enter the QP.

MPC and predictive filtering apply the same 10^4 penalty to slack on predicted clearance or terminal constraints, while keeping actuator constraints hard. On a numerical failure, the controller checks the shifted plan and a maximum admissible braking plan, then executes the candidate with lower slack. Initialization, restoration, and fallback evaluations count toward the time and evaluation limits. Slack permits predicted constraint violations; we assess physical outcomes through the simulator’s contact metric.

C.2 Backbone and training

The backbone retains four 64-dimensional blocks, concatenated as a 256-dimensional state. A width-256 feature stream passes through four gated feedforward blocks of width 512. At block b ,

$$\tilde{a}_b = a_b + V_b [\text{SiLU}(W_b \text{LN}(a_b)) \odot W'_b \text{LN}(a_b)], \quad (24)$$

$$\gamma_b = 0.05 + 0.94 \text{sigmoid}(A_b \tilde{a}_b + \alpha_b), \quad (25)$$

$$h_{b,t} = \gamma_b \odot h_{b,t-1} + (1 - \gamma_b) \odot \tanh(B_b \tilde{a}_b + \beta_b). \quad (26)$$

For $b < 4$, set $a_{b+1} = \tilde{a}_b + C_b h_{b,t}$. The first feature vector is a learned projection of x_t . The decoder is a $256 \rightarrow 512 \rightarrow m$ multilayer perceptron with a SiLU hidden activation, followed by actuator projection. The corrected state contains all four recurrent blocks.

The teacher uses the student’s sensed context, 30 SQP iterations, and eight initial control sequences. Training minimizes squared error on projected actions with AdamW: learning rate 3×10^{-4} , cosine decay, weight decay 10^{-4} , clipping at one, and 64 subsequences of length 64 per batch. Each subsequence has a 32-input burn-in. Navigation uses 100,000 updates and the chain 80,000. Checkpoint selection maximizes validation success, breaking ties by contact rate.

Training-rollout means and standard deviations define $z = D^{-1}(h - \bar{h})$, with standard deviations floored at 10^{-2} . Recurrence and decoder incorporate the inverse transform. Weights and scaling statistics are frozen before evaluation.

C.3 Selection and timing

Validation searches damping values 0.005, 0.02, and 0.08; candidate limits of one, two, three, five, or eight; and relative stopping thresholds of 10^{-4} , 10^{-3} , or 10^{-2} . All correction methods use $\mu = 0.05$ and $\rho_U = 0.20$. State radii are selected from 0.20, 0.35, and 0.50; both tasks select 0.35. Retain and discard share the retain-selected settings. MPC and predictive filtering search the same five choices of SQP iteration limit. Each QP has at most twenty interior-point iterations. Every configuration uses the same validation episodes.

Selection maximizes validation success within both latency targets, then minimizes contact rate and mean time. Settings are fixed before test evaluation. Retain and discard share three trial slots. Action refinement with fitting also has three candidate slots at the reference envelope: at most two for action search, including the shifted-plan check, and one for fitting.

Neural inference uses PyTorch 2.2 and CUDA 12.1; SQP models and derivatives are compiled before timing. We pin the controller process to one physical CPU core, run it without concurrent jobs, and measure from preprocessed host observation to completed host control, including transfers and fallback work. After warm-up, five complete-stream sweeps use independently permuted episode orders. Their mean times vary by at most 0.09 ms and their p99 values by at most 0.23 ms.

C.4 Branch interventions and physical cost

Before each donor episode, we choose one checkpoint from inputs 20 through 120 in navigation or inputs 8 through 24 of the chain’s second target interval. Navigation donors run for 160 inputs and hold the goal after arrival. Half use action refinement and half use state correction with discard. At the checkpoint, we save the plant, controller, two-input observation history, and exogenous continuation draws.

The correction and first action are computed once and copied. Empty corrections remain as zero interventions; 61% of navigation and 68% of chain checkpoints have an accepted update. Replay observations come from a separately continued discard branch with correction disabled, including its previous-action features. The active waypoint or endpoint is held fixed for eight inputs, except in the target-replacement test.

Physical branches generate their own observations, with noise and obstacle innovations indexed by absolute input and sensor or object slot. This pairs exogenous draws as the trajectories diverge. Inherited optimizer workspaces and plan caches are cleared in every branch before the common solve.

For navigation, the realized cost over $n = 8$ future inputs is

$$J_n^{\text{nav}} = \frac{1}{n} \sum_{h=1}^n [\|(p_{t+h} - p_{t+h}^{\text{wp}})/3\|_2^2 + 0.05 \|u_{t+h}\|_2^2 + 10 \mathbf{1}\{\text{contact at } t+h\}] + \|(p_{t+n+1} - p_{t+n+1}^{\text{wp}})/3\|_2^2. \quad (27)$$

The indexing starts with the first decision after the common action; p_{t+h} is the measured state before that decision. The chain uses endpoint error normalized by 0.5 m, joint-velocity cost $0.02 \|u/1.2\|_2^2/12$, the same contact penalty, and a terminal endpoint term. Reported percentage reductions use ratios of mean physical costs, as in eq. (11). Active-correction continuations use the same stopping threshold in both branches; residual-Jacobian counts include evaluations followed by rejection or immediate termination.

For target replacement, navigation receives a different reachable waypoint in the current room; the chain receives a feasible endpoint at least 0.3 m from the old target, screened by the same generator. Both new references are announced at the next input. Delayed-sensing tests hold navigation obstacle records or chain joint-angle measurements two inputs behind. Navigation keeps current proprioception, and the chain keeps the previous executed command. Both use delayed quantities without extrapolation, with the correction rule unchanged.

C.5 Plan-cache continuation

The four branches in table 1 start from the same checkpoint and common solve as the other interventions. The cached sequence is exactly $\tilde{U}_t$, including at checkpoints without an accepted state update. Each cached plan is shifted by one input and padded by repeating its last control, then projected onto the current actuator set. The action-space solver evaluates the projected plan against the fresh neural proposal and chooses

Table 5. Paired effects by independently trained model. Columns report full-episode retain-minus-discard success, the eight-input physical-cost reduction from state retention with both plan caches enabled, and delay-burst success gain from rollback over unconditional retention. Success differences are percentage points (pp); cost reductions are relative percentages.

Model	Retain vs. discard (pp)		State given plan cache (%)		Rollback vs. retain (pp)	
	Navigation	Chain	Navigation	Chain	Navigation	Chain
1	2.50	5.60	0.4	2.2	1.80	0.73
2	4.25	2.60	-0.1	1.0	0.30	2.20
3	1.40	4.20	1.5	0.6	2.40	0.40
4	3.35	6.20	0.6	0.2	0.80	1.07
5	2.00	3.40	1.1	1.5	1.20	1.60

the lower task cost, breaking ties in favor of the neural proposal.

A cache check uses a trial even when the plan is rejected.

All four branches have three candidate slots, damping 0.02, Armijo coefficient 0.1, and the same relative stopping threshold. Only the task residuals enter this diagnostic solve: $\mu = 0$, with no proposal-relative radius. A direction is accepted using its actual projected displacement. Cache maintenance and every candidate check count toward latency. The initial state intervention occurs once; subsequent recurrence is uncorrected in all four branches, while a plan cache is refreshed wherever that factor is enabled.

Cost reductions are ratios of mean physical costs across checkpoints. To measure the state contribution given a cache, we compare the state-plus-plan branch directly with the plan-only branch. This conditional ratio differs slightly from subtracting their reductions against the no-cache reference. Per-model effects use the same ratio within each model’s 800 checkpoints.

C.6 Context rollback and per-model effects

Each active navigation waypoint and commanded chain endpoint has an identifier that changes on ordinary transitions and replacement commands. Sensor age comes from the obstacle-record timestamp in navigation and the joint-angle timestamp in chain control. Every method can access this metadata, separately from the frozen network’s input features. We fix the one-input age threshold before evaluation.

Rollback is applied before the next recurrent update. Afterward, the old proposal/correction pair is replaced by the new pair, so the saved state permits reversal of just the latest correction. During delay-burst evaluation, all methods share targets, delayed records, and disturbance streams, with current sensing outside the two specified bursts. Nominal tests retain ordinary target transitions, and solver settings stay fixed for both rollback and delayed sensing. Rollback takes 3.87/7.50 ms (mean/p99) in navigation and 3.94/7.74 ms in chain control during delay bursts.