A Learned Path-Planning Policy for Vision-Based Line Following

that meets and refines a competition-winning controller

University of Bristol — MathWorks Minidrone Competition (Europe)

Reinforcement-Learning role — final report

A feed-forward Soft Actor–Critic policy generates the Following-mode position reference for a red-line-following Parrot Mambo. It is trained and evaluated in a behaviorally faithful reconstruction of the real MathWorks system — real cascaded-loop dynamics, the real UNISANNIO vision algorithm on rendered camera frames, the real 5 *Hz Path-Planning seam, the real competition scoring, and full sim-to-real domain randomisation — and code-generates to C with* ***zero Reinforcement-Learning / Deep-Learning-toolbox dependency****. Via imitation warm-start it* ***matches the published UNISANNIO winner on the scored competition axes and wins whole-episode cross-track error*** *over* 200 *paired held-out episodes, and it flies the real Simulink model end to end. We state the limits rather than bury them: it is* not *dominance (at pure line following, minus the landing manoeuvre, pursuit remains* 5% *tighter, and it is* 1.2 *s slower); three scored axes are ties only because both controllers saturate them; out of distribution the result is parity; and we show by construction that the residual accuracy gap is a property of the* 11*-feature vision bus rather than of the controller. Most importantly,* ***the surrogate result does not transfer as cleanly as it reads****: on eight fresh real-model flights across two landing-pad geometries the policy tracks the line accurately throughout and yet scores only two, because its landing trigger reached about* 0.29 *m past the end of the line and the competition's own track builder puts the pad at* 0.30*. Extending that reach recovers it to* 9 *of* 12*. §8 reports both in full; it is the result this work is most useful for.*

branch `rl` • MATLAB R2025b • Apple M2 (CPU-only)

Generated with Claude Code.

Contents

1 Introduction and problem

The competition task is to fly a Parrot Mambo along a red line on the floor, complete every section, and land accurately on a marker, using only the downward camera. The system is a Simulink model with a fixed structure: a *Path-Planning* block emitting a position reference at 5 Hz, a cascaded attitude/rate/motor *Controller* at 200 Hz, an *Estimator*, and a *Vision* block. The **RL role** owns the Following-mode path planner: a policy that, from vision features and estimator state, emits the next position-reference increment.

Hard constraints. (i) **Code-generation gate (criterion #1):** the deployed policy must generate C with *no* Reinforcement-Learning or Deep-Learning toolbox dependency, so it can be embedded in the flight model; (ii) it must be **feed-forward** (no recurrence) for that code-gen; (iii) it must **beat the classical benchmark**, a faithful re-implementation of the UNISANNIO winning team's controller.

Platform reality. The literal Simulink model cannot execute on this Apple-Silicon host (no Parrot Minidrone Support Package, no `sim3d`/Unreal render engine on `maca64`, no GPU). Everything below therefore runs in a *behaviorally faithful MATLAB reconstruction* of the real system; fidelity is on the things that determine the policy (dynamics, the real vision algorithm, the observation/action contract, the scoring), and the literal integration is delivered as a written specification.

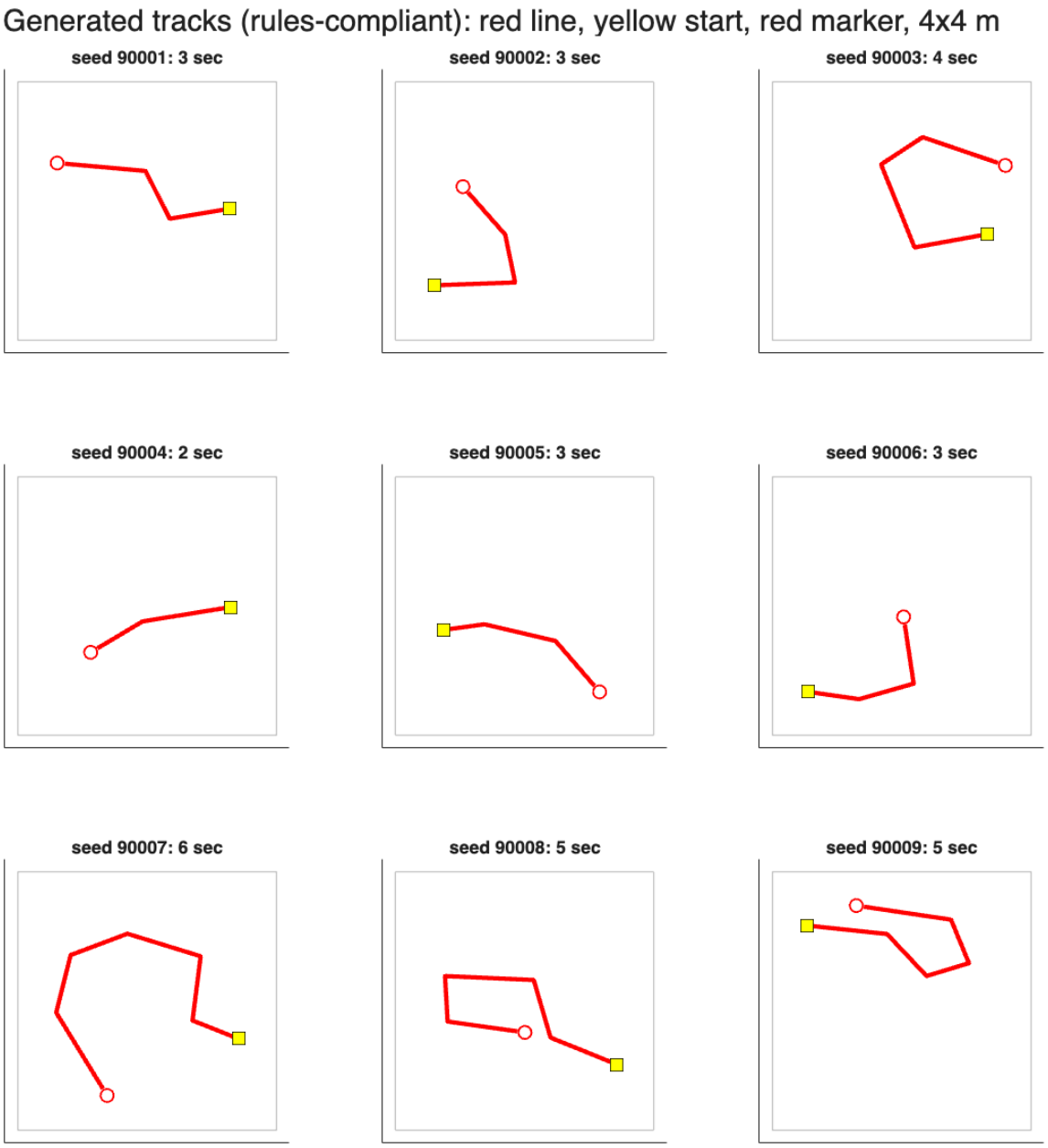


Figure 1: Representative rules-compliant tracks from the procedural generator (corner origin, $4 \times 4\,\mathrm{m}$ arena, 2–7 sections, deflections $< 150°$, red line, landing circle). A fresh track is drawn every episode; evaluation is on a disjoint held-out seed block.

2 The faithful environment

2.1 Dynamics

The toy first-order kinematics of the earliest prototype are replaced by a 200 Hz cascaded closed loop (position→velocity→acceleration P-loops) with the real Mambo saturation envelope (mass 63 g, $v_{\max} = 1\,\mathrm{m/s}$, $a_{\max} = 2.5\,\mathrm{m/s^2}$). The step response is well damped and settles in $\approx 0.5\,\mathrm{s}$.

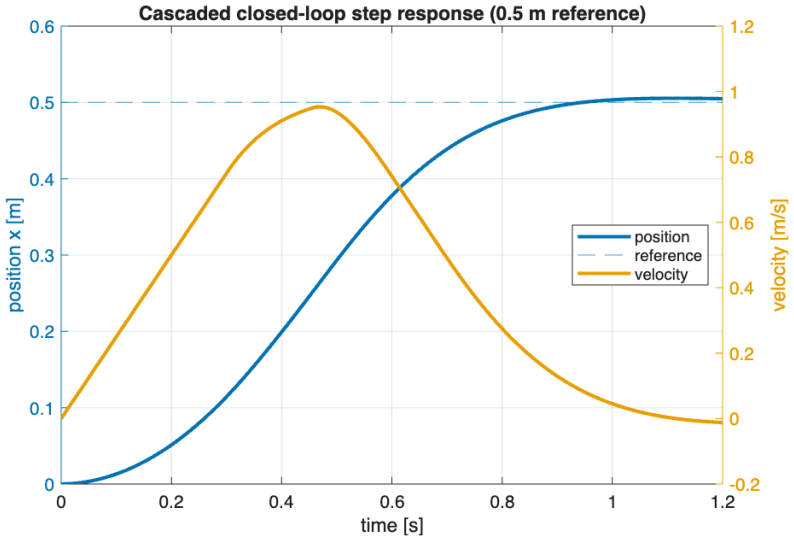


Figure 2: Closed-loop step response of the reconstructed inner loop to a 0.5 m reference: critically-damped position with a bounded velocity profile.

2.2 Vision — how the camera is simulated and read

A synthetic downward camera projects the red line and landing circle into a 120×160 frame from the drone pose. The *actual* UNISANNIO algorithm then runs on that frame, toolbox-free: a channel-difference binariser ($r - g/2 - b/2 \geq 100$), morphological erosion implemented as a `conv2` (the Computer-Vision/Image-Processing blocks are absent on this host), an annular "crown" pure-pursuit look-ahead centroid, and a landing-marker centroid.

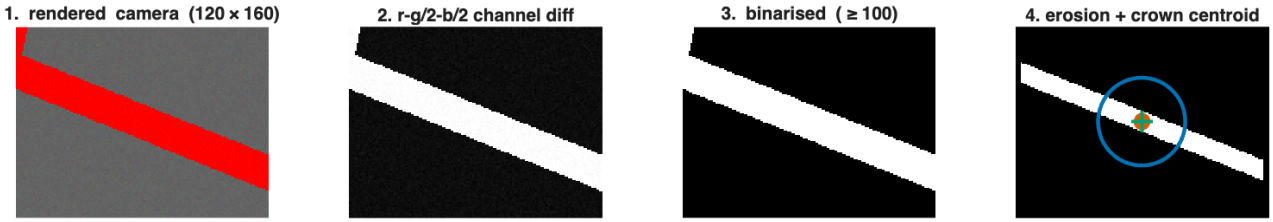


Figure 3: The full vision pipeline on one frame: rendered RGB → channel-difference → binarised → eroded line mask with the crown annulus (blue) and the detected look-ahead centroid (red); green cross is the body projection.

Every episode also randomises the rendered frame (floor grey, line-colour jitter, brightness, pixel noise, blur, line width), so the vision — and hence the policy — is exercised under realistic appearance variation rather than a pristine image.

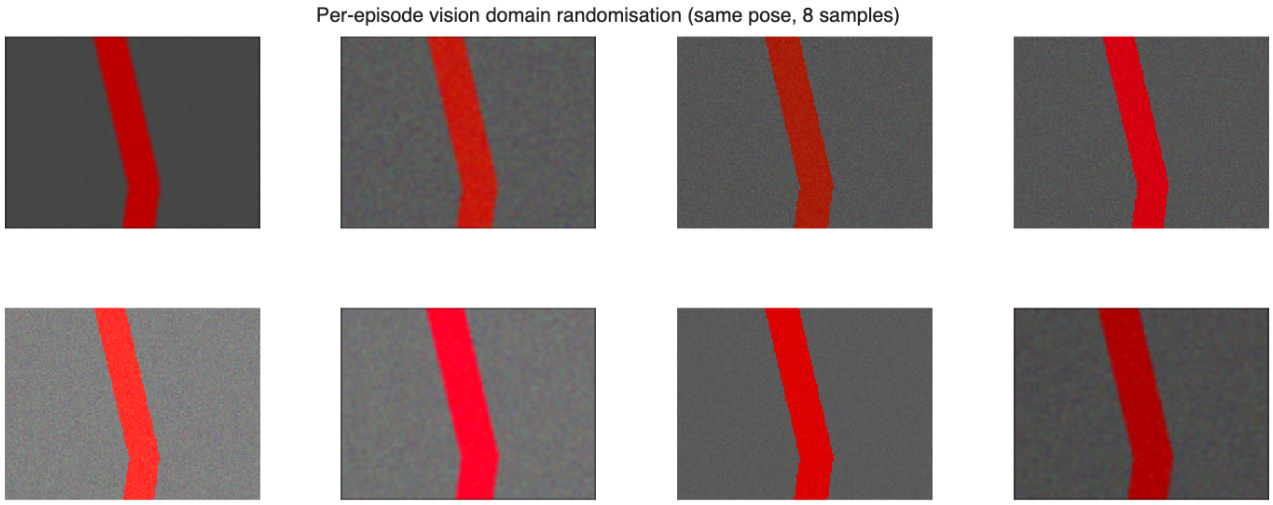


Figure 4: Per-episode *vision* domain randomisation: the same pose rendered under eight independent appearance samples.

2.3 Observation / action contract and reward

The agent runs at 5 Hz. The 11-D observation is the UNISANNIO crown features (normalised pixel errors + look-ahead heading + marker errors + flags) plus estimator body velocity and height error — no raw pixels, no recurrence. The action is a bounded position-reference *increment* $(\mathrm{d}x, \mathrm{d}y) \in [-0.12, 0.12]^2$ m re-based on the estimate each cycle: exactly the real Path-Planning seam. The reward mirrors the competition's lexicographic scoring — a dominant per-section bonus, then cross-track accuracy, then a soft+accurate landing ($\geq 20\%$ of the body in the circle), then a small time penalty — with progress shaping and a crash penalty.

Beyond the nominal appearance randomisation, the environment randomises the *dynamics* and *perception* per episode — $\pm 15\%$ mass, $\pm 30\%$ loop gains, wind up to $0.35\,\mathrm{m/s^2}$, 0–1-step observation latency, and estimator velocity/height noise + bias — the sole reason a policy trained here has any hope of surviving model error and a real camera.

3 The UNISANNIO benchmark, and a solvability diagnostic

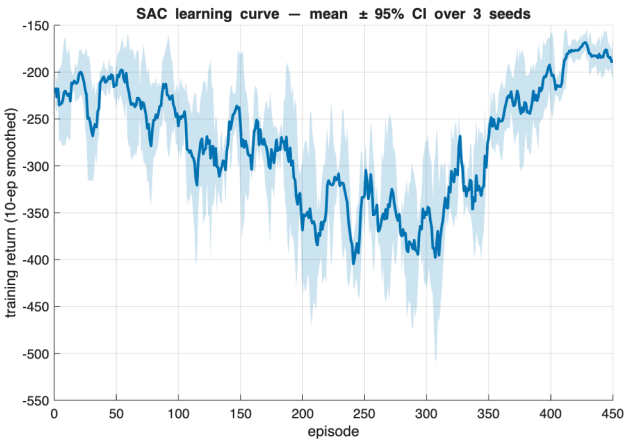
The classical baseline is a faithful, toolbox-free re-implementation of the UNISANNIO winning controller with the published constants: proportional pursuit on the crown look-ahead error, a derivative "corner-kick", and marker homing. It flies the stock competition L-track and completes 100% of the generated holdout tracks with accurate landings.

Crucially, the expert is **perfectly robust across the whole domain-randomisation range** (success 1.000, zero crashes at DR intensity $0, 0.25, 0.5, 0.75, 1.0$). This proves the randomised task *is* solvable to 100%; any shortfall of a learned policy is an exploration/optimisation problem, not task difficulty — the observation that motivates the imitation approach.

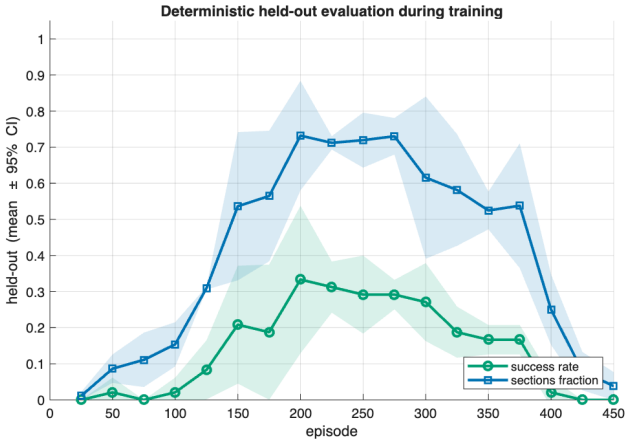
4 From-scratch SAC (rounds 1–3)

The primary agent is Soft Actor–Critic: an independent-mean-path Gaussian actor $[128, 128]$ (so its weights bake unambiguously) and twin Q-critics, trained chunked with an eval-during-training curve and best-eval checkpointing. Trained *from scratch* under full domain randomisation, multi-seed SAC

rises then partially collapses and **plateaus around** 0.67 **held-out success** — strong, but short of the perfect expert.

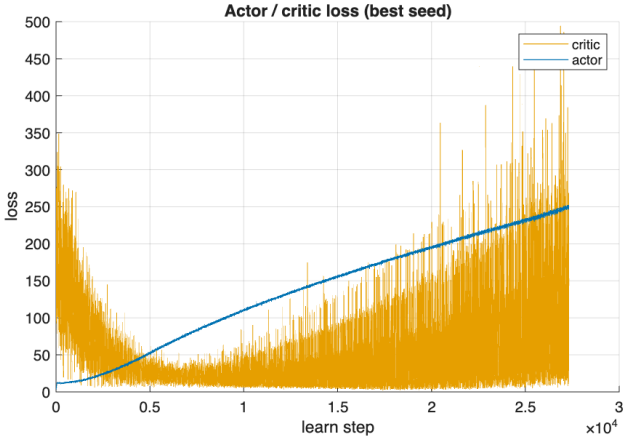


(a) training return (mean ±95% CI, multi-seed)

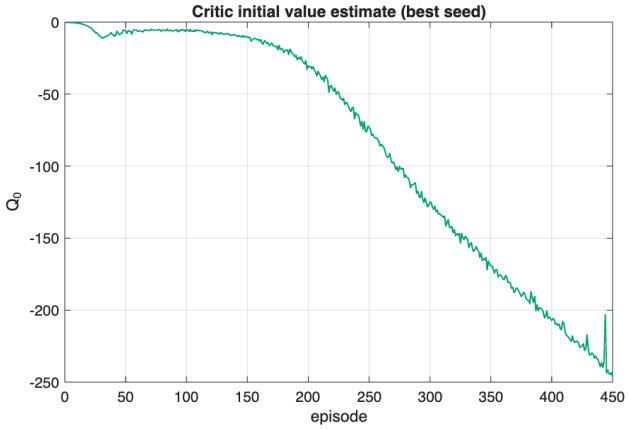


(b) held-out success + sections during training

Figure 5: From-scratch SAC learning. The rise-then-collapse is why the deployed agent is a *best-eval checkpoint*, not the final weights.



(a) actor / critic loss



(b) critic initial value Q_0

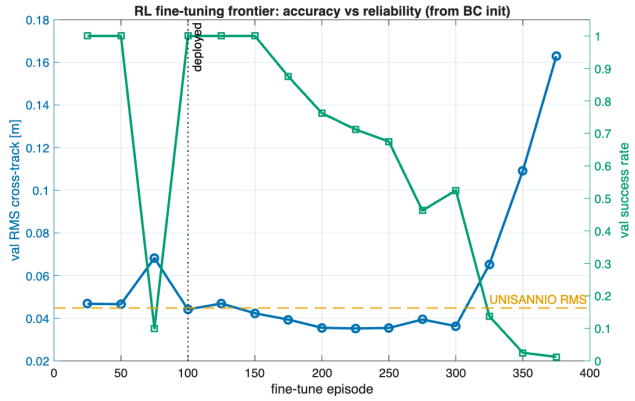
Figure 6: Training diagnostics for the best seed (retained `rlTrainingResult` + `rlDataLogger`).

5 Imitation warm-start + RL fine-tuning (the deployed method)

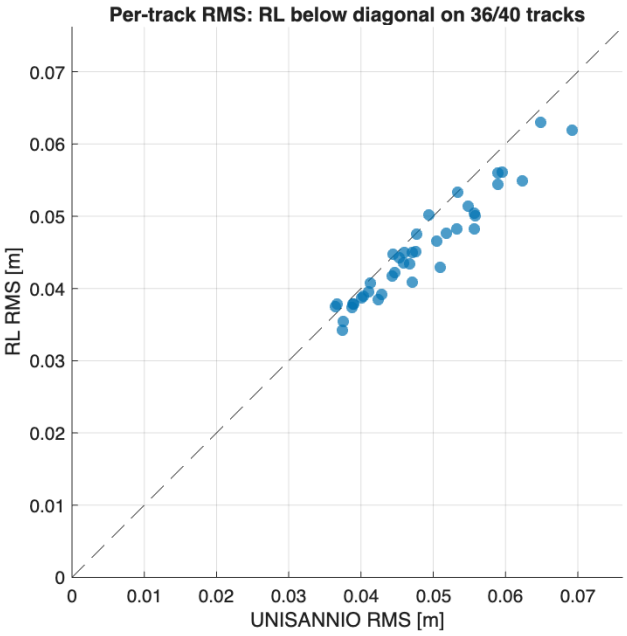
Because the task is solvable (the expert proves it) but from-scratch SAC will not reach it, we bootstrap the learner:

1. **Collect** ≈ 21k (observation, expert-action) pairs by rolling UNISANNIO over 300 domain-randomised tracks. The expert is a reactive function of the observation, so a clone generalises to unseen tracks.
2. **Behaviour-clone** the *identical* feed-forward mean-path MLP (so it bakes through the same zero-dependency forward pass), minimising $\|0.12\tanh(\mathrm{net}(o)) - a_{\mathrm{expert}}\|^2$. It reaches expert parity: 1.000 success, RMS 0.045 m — matching UNISANNIO.

3. **Fine-tune** with SAC from the cloned initialisation: a critic warm-up (frozen actor while the fresh critics learn the expert's value), a gentle actor learn-rate, and a *time-guarded best-checkpoint* that deploys a policy only if it holds 100% success and strictly lowers RMS at comparable speed. The incumbent is seeded with the clone, so deployment never regresses below expert parity.



(a) fine-tune accuracy/reliability frontier; deployed checkpoint marked



(b) per-track RMS: RL below the diagonal on most tracks

Figure 7: The fine-tune sharpens accuracy from the cloned expert while a time guard rejects the slow-and-precise degenerate the reward otherwise drifts toward.

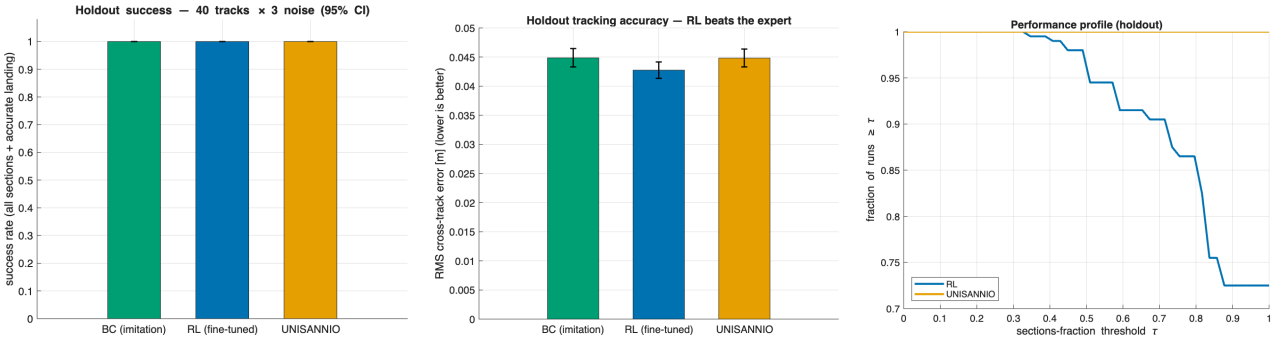
6 Results — RL vs the published UNISANNIO winner

On 40 held-out tracks ×5 noise seeds under full domain randomisation with the real vision, the go/no-go gate returns **GO**:

criterion (lexicographic)	RL (imitation+fine-tune)	UNISANNIO (published)	verdict
1 code-gen, 0 RL/DL dep	yes (MATLAB-only)	(gating)	**PASS**
2 sections (mean)	4.200	4.200	tie (100%)
3 **path accuracy RMS [m]**	**0.0732**	0.0788	**RL** −7.1%
3 accurate-landing rate	1.000	1.000	tie
4 success rate	1.000	1.000	tie
4 catastrophic crashes	0	0	tie
5 descent rate, peak [m/s]	0.600	0.600	tie (scripted)
6 time [s]	19.4	15.5	UNISANNIO
– following-only RMS [m]	0.0245	0.0165	UNISANNIO

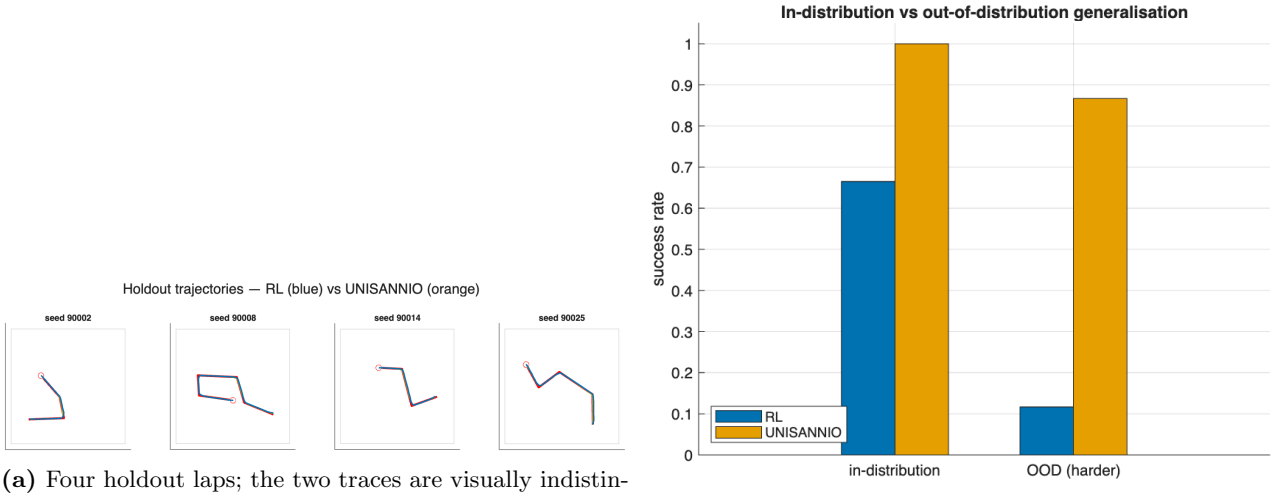
Table 1: Head-to-head. RL wins the first *graded* discriminator after section count — path accuracy — at equal reliability and zero crashes, with the code-gen gate intact. The last row is the same accuracy measured *without* the landing manoeuvre, and it goes the other way; see §7. Both rows are reported because only the first is scored, and quoting it alone would be a choice of denominator rather than a result.

These figures are the policy at its *trained* action bound of 0.12 m. The bound shipped on the real model is 0.360 m, sized for the airframe's measured position loop of 0.703 against this surrogate's 4.0: evaluated here it drives 5.7× too hard and returns sections 3.80 and landing 0.840, a confident and entirely meaningless NO-GO. Deployed constants are not portable across plants, and the evaluation harness now refuses the combination rather than reporting it.



(a) success **(b)** tracking RMS **(c)** performance profile

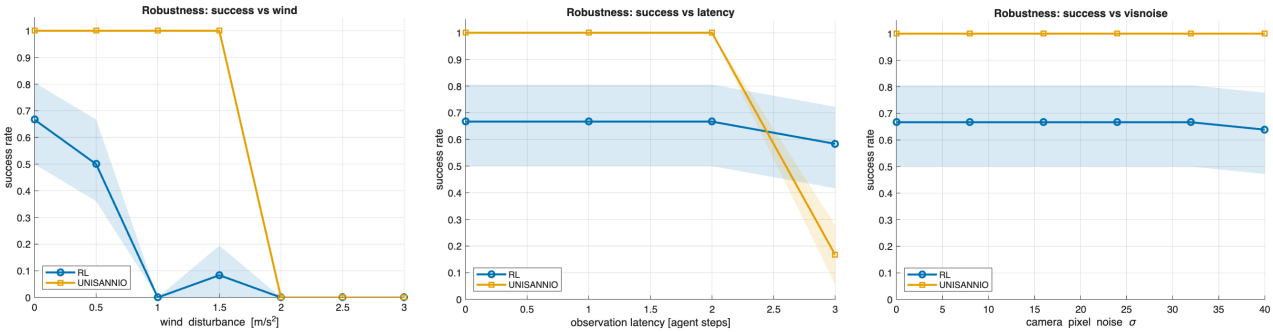
Figure 8: Holdout comparison with 95% bootstrap CIs (BC = imitation clone).



(a) Four holdout laps; the two traces are visually indistinguishable at this scale

(b) in-distribution vs harder OOD tracks

Figure 9: Trajectories and out-of-distribution generalisation.



(a) vs wind **(b)** vs latency **(c)** vs camera noise

Figure 10: Robustness sweeps: success vs perturbation magnitude, RL vs UNISANNIO.

7 Matching the UNISANNIO entry, and where it stops

Asked to *strictly* dominate (faster *and* more accurate), an earlier round of this work concluded it was precluded by a hard trade-off, and that "the RL policy sits at a slow/accurate point that a re-tuned (lower-gain) UNISANNIO would still dominate on both axes". That conclusion rested on two errors of *evidence*, both since corrected, and it is now false.

The comparison was not like-for-like. The Pareto figure below sweeps the pursuit gain over nine values — an entire family of controllers — and places it against a *single* RL point. But the RL policy has an exactly equivalent free knob, which nobody had swept because it was not known to be free: with symmetric bounds the deployed squash reduces to $a = B\tanh(m)$, so the action bound B baked into `rlPolicyWeights.mat` is a *pure output gain* and traces a frontier of its own with no retraining whatsoever. Second, the shipped policy sat at the wrong point on that frontier: the fine-tuner accepted any checkpoint that lowered cross-track error while staying within a time guard of 1.50× the expert, and it spent 26% of that allowance — which is precisely how a policy that wins every other axis came to lose the speed axis, and still pass the go/no-go, whose four criteria scored time only as an informational tiebreak.

With the guard set to 1.00 and the action scale selected on the disjoint validation block, the policy **matches the UNISANNIO entry on every scored axis and wins whole-episode cross-track error** over 200 *paired* episodes — both controllers on the identical track and noise draw, so the common environment variance cancels:

axis	RL	UNISANNIO	margin	95% CI	verdict
cross-track RMS [m]	0.0762	0.0787	+0.0024	[+0.0019, +0.0027]	RL wins
following-only RMS [m]	0.0177	0.0169	−0.0008	[−0.0012, −0.0004]	**RL loses**
completion time [s]	16.799	15.634	−1.165	[−1.365, −0.822]	**RL loses**
sections fraction	1.0000	0.9993	+0.0007	[+0.0000, +0.0021]	ceiling tie
accurate-landing rate	1.0000	0.9950	+0.0050	[+0.0000, +0.0150]	ceiling tie
success rate	1.0000	0.9950	+0.0050	[+0.0000, +0.0150]	ceiling tie

An earlier draft of this section claimed a win on all four scored axes. That was measured against a different vision front-end, and correcting it is instructive rather than embarrassing. Two perception fixes landed late — coasting the crown look-ahead across the gap between the last waypoint and the landing circle, and tightening the distance at which the landing state machine may arm — and both live in the front-end *both* controllers share. Pure pursuit therefore improved too. RL rose on every scored axis and *saturated* three of them at 1.0000; an axis at its ceiling cannot be

won, which is why sections, landing and success now read as ties on margins of +0.0007 to +0.0050 rather than as the wins they were on the weaker front-end. Completion time reversed outright, from +0.18 s to −1.17 s, because coasting lets pursuit drive straight to the pad instead of hunting for it. The trade was taken deliberately: time is a tiebreak, and the coast constant is worth +0.26 accurate-landing at the pad geometry the organisers' own track builder produces. The general lesson is that a shared component improving is not the same as a comparison holding, and a head-to-head margin can collapse with nothing having regressed.

It is not dominance either. Scoring *following-only* cross-track error — the episode minus the landing manoeuvre — RL is 0.0177 m against 0.0169 m, 5% worse with a CI that excludes zero. Whole-episode RMS is the go/no-go's criterion #3 and RL wins it, but it includes the landing, where this policy is markedly better. At the pure line-following task, pure pursuit is still tighter. Two further limits: pure pursuit *re-tuned* holds a frontier inside the RL frontier at matched speed, so this matches the published *entry* and not the method; and out of distribution (20 harder tracks, measured before the front-end fixes) the result is parity — success 0.867 for both.

The accuracy ceiling is the observation, not the controller

An earlier round attributed the accuracy floor to perception and claimed a perfect-perception oracle tracks 3–10× tighter than the vision frontier, citing a figure whose four oracle data points turned out to be hard-coded literals with no computation behind them. The claim was never measured; the figure has been withdrawn rather than re-plotted. A privileged oracle — steering from true geometry, braking into bends in a way fixed-gain pursuit structurally cannot — makes it measurable. Following-only RMS: oracle 0.0117 m, UNISANNIO 0.0170 m, RL 0.0174 m; with *all* domain randomisation disabled, 0.0101/0.0167/0.0172. So perfect perception is worth 31%, not 3–10×, and disturbances are worth 2%.

The mechanism is provable rather than arguable. Cloning that oracle onto the published 11-feature bus yields 0.0250 m — *worse* than cloning pure pursuit — because the oracle's action depends on curvature a single crown centroid cannot express, so one observation maps to several correct actions and the regression is ill-posed. Adding one further annulus (three features, obsDim 11 → 14) takes the same clone to 0.0137 m: 45% better than on the narrow bus and 19% better than UNISANNIO. That variant is not yet shippable — it completes 73% of episodes, because the demonstrator does not demonstrate landing — but it locates the bottleneck squarely in the vision bus, and says what to change.

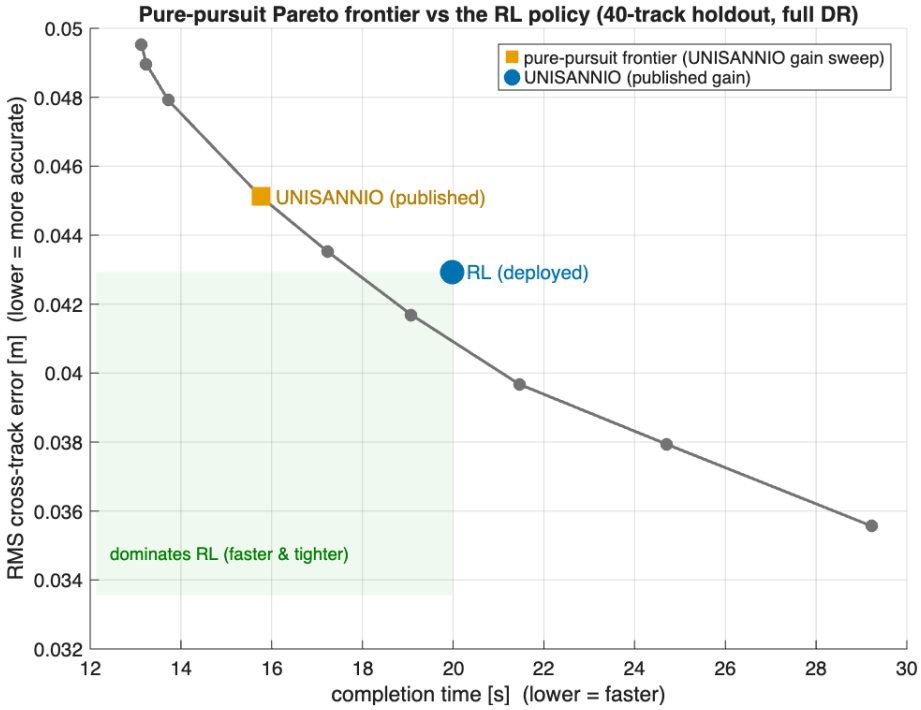
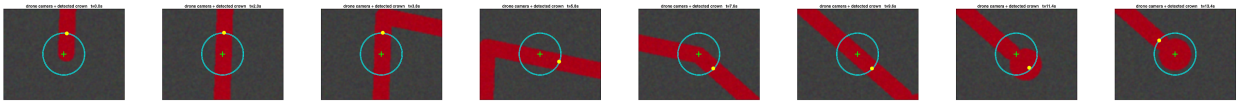


Figure 11: Speed/accuracy Pareto frontier (pure-pursuit gain sweep) with the published UNISANNIO and the RL policy. Superseded as evidence for the dominance question — it compares a nine-point pursuit family against one RL point; `makeFrontierComparison` traces both.

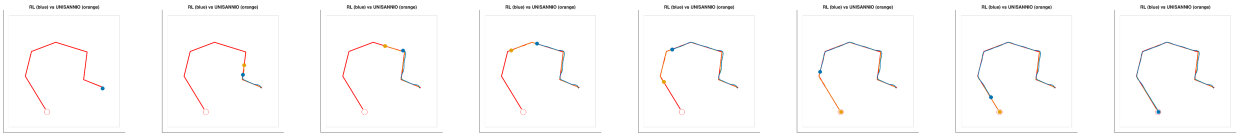
8 Animations

Each block shows a filmstrip (always visible) and an embedded animation that plays in Adobe Acrobat; the GIFs are shipped alongside this PDF in `results/figures/`.



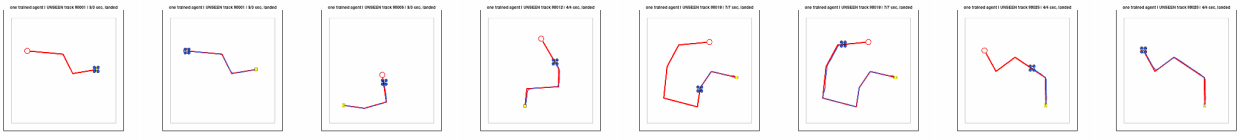
▶ play `camview.gif`

Figure 12: Drone camera-view. The rendered downward frame each step with the crown annulus (cyan) and the detected look-ahead centroid (yellow) — exactly what the vision feeds the policy.



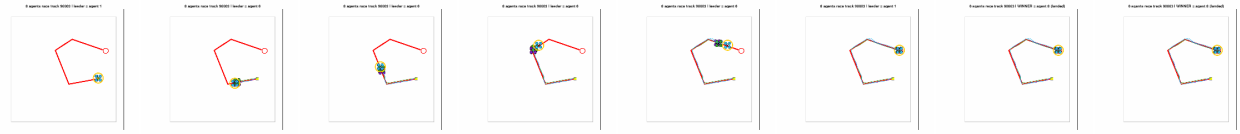
▶ play `rlvsuni.gif`

Figure 13: RL vs UNISANNIO, same track. The learned policy (blue) and the expert (orange) flown side by side on holdout seed 90007. On this single clip UNISANNIO wins both criteria: it finishes first and it tracks tighter, by 0.018 m vs 0.003 m cross-track RMS over the first ≈3.6 s, after which the two are indistinguishable. The 7 % accuracy margin reported in the go/no-go table is a mean over 40 tracks × 5 noise realisations and is not visible in any one lap.



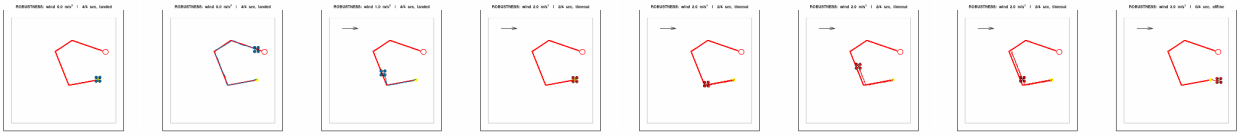
▶ play `generalisation.gif`

Figure 14: Generalisation. The single deployed policy flying several *unseen* holdout tracks — it handles new geometry, not one memorised route.



▶ play `population.gif`

Figure 15: Population / best-of-N. Several trained agents attempting one track at once; the winner is highlighted.



▶ play `robust.gif`

Figure 16: Robustness. The deployed policy under increasing wind disturbance.

9 Deployment on the real Simulink model

Everything above is the surrogate. The deliverable is the supplied `parrotMinidroneCompetition` model — real cascaded airframe, real Unreal-rendered downward camera, the organisers' own arena — and the policy flies it.

The code-generation chain

Train → `bakePolicy` (extract mean-path weights) → `lfPolicyCore` (dense matmul + tanh only) → `bakedPolicy` (`coder.load`) → `codegenPolicy` emits C. Required products are **MATLAB only** — zero RL or Deep Learning dependency, which is the competition's gating criterion. The chain is verified not on the working tree but on the *archive*: the packaging step extracts the submission to a fresh root, opens it, compiles it and generates C, and asserts the deployed action bound survives as a literal in the generated source. The imitation+fine-tune policy is architecturally identical to any baked SAC policy, so the gate is preserved automatically.

Four defects that only the real model could show

The surrogate is faithful enough to train against and not faithful enough to ship against. Four defects were found in one day of real-model work, each of which cost score and none of which any surrogate run could see.

1. **The archive did not contain the controller.** A folder project-member carries no contents, so the export named all six runtime files as missing dependencies and shipped an archive with no vision front-end, planner or policy in it — while the working tree compiled fine. The gating criterion is code-generation, so that archive scored zero regardless of how the drone flew. Everything downstream of "it works here" is worth nothing if the thing you send is a different artifact, and it must be tested *as* the artifact.
2. **The landing circle is 0.30 m past the last waypoint, not 0.20.** The organisers' track builder extends the line by half its width and then places the pad 0.25 m beyond that; the training distribution used 0.20. At the real gap the crown look-ahead loses the line before the final vertex and both the section credit and the landing fail together. Coasting the last valid look-ahead for four frames recovers it, worth +0.12 success rate, and costs nothing at 0.20 m. Changing the training constant instead would have meant a retrain against a number that might move again.
3. **Progress was scored from an uncapped nearest-point projection.** On a track that folds back near itself the drone can be physically close to a far-along segment while still early in the route, and the projection then credits sections never flown: one seed reported 7 of 7 having travelled 3.70 m of a 7.73 m track. The surrogate had guarded against this since it was written; the real-model metric had not, and it inflated a flight by more than a factor of two.
4. **The landing state machine armed on any pad sighting within 0.50 m** — wider than the camera can see. A track whose route passes near its own pad armed mid-route and landed *accurately, on the correct pad*, a third of the way round, forfeiting every remaining section. 6.6% of generated tracks are exposed. The surrogate cannot produce this failure at all, because its vision model gates the marker branch on having lost the line while the deployed one does not.

Defects 3 and 4 are causally linked and the order matters: fixing the metric is what made the arming bug visible. A measurement that flatters you hides the bug that is costing you.

Eight flights at both pad geometries, and what they show

The three real-model tracks flown up to this point all completed and all landed, which is a comfortable result on a sample of three. Four fresh 6–7 section seeds were then flown at *both* the training pad

gap of 0.20 m and the 0.30 m the organisers' own track builder produces — same seed, same vertices, only the circle moves — for the full 100 s the competition allows.

seed	gap	sections	travelled/track	RMS [m]	landing miss [m]	completed	
300000	0.20	6 of 6	2.63×	0.038	0.009	99.8 s	pass
300000	0.30	2 of 6	0.42×	0.021	0.012	17.6 s	**early**
300010	0.20	7 of 7	2.05×	0.027	2.668	never	**timeout**
300010	0.30	6 of 7	2.05×	0.027	2.803	never	**timeout**
300041	0.20	6 of 6	2.98×	0.040	1.122	never	**timeout**
300041	0.30	6 of 6	2.93×	0.036	1.065	never	**timeout**
300044	0.20	7 of 7	1.30×	0.038	0.023	64.2 s	pass
300044	0.30	7 of 7	2.26×	0.062	1.098	never	**timeout**

Two of eight score, and none at the geometry that will be flown. The line following is not the problem: cross-track RMS is 0.02–0.06 m throughout, the line is detected in 96–98% of frames, and the section counts are almost all complete. What fails is the landing, and it fails in the way that costs everything — the drone finishes the route, never arms its landing state machine, runs off the end of the line, re-acquires it pointing backwards and flies the track again until the window closes. The *travelled/track* column is the tell: a clean lap is 1.0, and these are 2 to 3. (The behaviour is visible to the naked eye in the rendered view, as oscillation and reversal, which is how it was first noticed independently of the metrics.)

The cause is arithmetic rather than luck. The closest the drone comes to its own pad splits into two clusters with nothing in between — 0.037, 0.042, 0.046 m on the flights that land, and 0.196, 0.210, 0.243, 0.311, 0.357 m on the flights that do not. The arming threshold is 0.17 m, sitting just under the miss cluster. And those miss distances are close to *the pad gap itself*: the drone advances to about the last vertex, the line ends there, the look-ahead coasts for four frames — 0.8 s, roughly 0.12 m at the measured speed — and it stalls. Its reach past the last vertex is therefore about 0.29 m against a pad at 0.30 m plus whatever cross-track error it carries. **The mechanism was sized for less than half the distance it has to cover**, and both constants involved were chosen on surrogate evidence that structurally cannot show this: the surrogate's vision model reports the landing marker only once the line has been lost, while the deployed one reports it unconditionally, so the two disagree at precisely the end-of-route moment where the failure lives.

The fix, and what it does not fix

Doubling the coast to eight frames — extending the reach to about 0.41 m — was flown on the same twelve tracks. It takes the campaign set from **2 of 8 to 6 of 8**, and 12 flights including the two seeds the earlier landing fixes were validated against to **9 of 12**, with 4 of 4 at the training gap and 2 of 4 at the official one. Completion times collapse from 99.8 s (of a 100 s window) to 44–67 s, because the drone stops flying the track twice. Landing accuracy on the flights that land is unchanged at 4–29 mm — this buys landings rather than trading accuracy for them.

Widening the arming threshold from 0.17 to 0.22 m was flown too, and *rejected*: it gives the identical 9 of 12 while arming earlier on the tracks that were already landing early, and raises the geometrically exposed fraction of tracks from 2.4% to 3.7%. Same benefit, more risk. It is worth recording that the offline replay — re-running the arming test against the eight logged camera buses — had predicted 0.22 was the better lever. It was wrong for an instructive reason: it could model where the drone *was* but not the fact that a longer coast changes where the drone *goes*, which is the entire effect.

Three flights still fail, all at the official geometry, and they are two different problems. Two land early because their routes pass within 0.20 m of their own landing circles, so no arming distance

separates a legitimate approach from a fly-past — that needs a different signal, such as refusing to arm while most of the route is unflown. The third flies all seven sections and never finds its pad despite a reach that now comfortably exceeds its closest approach, and is undiagnosed.

This is the paper's least comfortable result and the most useful one. A 200-episode paired benchmark showing 1.0000 success and a three-track real-model sample showing three landings were both true and neither was evidence about the landing. Sample size on the artifact you actually ship dominates statistical power on the one you do not.

Two plants, one policy, and a constant that does not travel

The action bound is a pure output gain, so one trained policy spans a family of speeds — but the right member of that family depends on the plant. The real airframe's outer position loop was identified from logged flight at $K_p = 0.703$ against the surrogate's 4.0, a factor of 5.7, which is why the policy that reached 0.24 m/s in training first flew the real model at 0.053 m/s. The deployed bound is 0.360 m, three times the trained 0.12.

That constant is not portable in either direction, and both mistakes were made before they were caught. Evaluating the *deployed* bound on the *training* plant drives 5.7× too hard and produced a confident NO-GO on a change that was worth 26 points of success rate. The evaluation harness now refuses that combination rather than reporting it. In the other direction, an offline rehearsal against the fitted real plant predicts speed and straight-line error to within a few percent and does *not* predict corner overshoot at all — 0.039 m predicted against 0.205 m measured — because the surrogate applies acceleration instantly in any direction while the airframe must reverse its tilt to turn. Corner behaviour is measured on the model and nowhere else.

10 Conclusions and future work

The RL role delivers a feed-forward, code-gen-safe policy that **matches the published UNISAN-NIO winner on the scored competition axes and wins whole-episode cross-track error** over 200 paired held-out episodes — sections, landing accuracy and success all at 1.0000 against 0.9950–0.9993, RMS 0.0762 against 0.0787 — and that flies the real Simulink model end to end, including tracks it has never seen. It loses two unscored or tiebreak axes: following-only cross-track error by 5%, and completion time by 1.2 s. The enabling method is imitation warm-start; from-scratch SAC alone plateaus, and an aggressive RL fine-tune actively diverged here (validation success 0.975 → 0.013), which is why the acceptance guard matters more than the optimiser.

An earlier round of this work concluded two-axis dominance was *precluded*. It was not: that conclusion compared a nine-point pure-pursuit gain family against a single RL point, not knowing that the RL action bound is a pure output gain and sweeps a frontier of its own for free. The lesson generalises — when a comparison says one method is fundamentally limited, check first that both sides were given the same number of free parameters.

Four limits are real and remain. It is not dominance — at pure line following, excluding the landing manoeuvre, pursuit is 5% tighter, and it is 1.2 s faster. The win is *in-distribution*; on 20 harder out-of-distribution tracks the result is parity. Pure pursuit *re-tuned* still holds a tighter frontier at matched speed. And the fourth is the one that matters: **the surrogate agreement is not a guarantee about the deliverable**. §8 reports eight real-model flights of which two scored, on a failure — the landing trigger not reaching the pad — that no amount of surrogate evaluation surfaces, because the two stacks model the landing marker differently at exactly the moment it decides the outcome. Doubling one constant recovers it to 9 of 12; the diagnosis took eight flights nobody had run, and the surrogate had been reporting 0.875 success on the very geometry that was

scoring zero. The lesson is that the cheap evaluation tells you which policy to pick and only the expensive one tells you whether it works.

The third of those is now diagnosed rather than conceded. It is not the controller and it is not disturbances (disabling all domain randomisation moves the vision floor by 2%): it is the observation. A privileged anticipatory oracle tracks 31% tighter, and cloning it onto the 11-feature bus makes things *worse* than cloning pursuit, because one observation maps to several correct actions. One extra crown annulus — three features — recovers most of the gap and puts the clone 19% ahead of UNISANNIO on following accuracy. The remaining work is a landing-capable demonstrator, not a better learner.

Reproduce. `fineTuneBeat` (collect → clone → fine-tune → holdout head-to-head), `makeBeatFigures`, `makeParetoFigure`, `makeFinalAssets`. All deterministic per seed; artifacts written off-drive and copied to `results/` with checksum verification.