Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

From Pixels to Registers: A Multi-Modal Testing Framework for Chained Perception, Control and Firmware

Brent Hartshorn

brenthartshorn@proton.me

none https://orcid.org/0009-0004-2853-655X

Research Article

Keywords:

Posted Date: September 15th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11020853/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: The authors declare no competing interests.

From Pixels to Registers: A Multi-Modal Testing Framework for Chained Perception, Control and Firmware

B.S. Hartshorn (https://github.com/crustos/robotsim)

September 12, 2026

Abstract

Robotic vision policies trained on photorealistic synthetic imagery fail when the real world presents a texture, a shadow or a reflection the renderer never produced. We argue the fault is the representation rather than the realism: a policy that consumes photographs must learn to ignore appearance, when it could instead be handed a representation from which appearance has already been removed. We describe a chained architecture in which a perception stage converts camera input into a non-photorealistic line drawing paired with a semantic map, and a control policy consumes only that abstraction. Decoupling perception from control allows each to be trained independently [4], and abstract intermediate representations such as sketches are known to preserve spatial grounding while discarding visual distraction [5].

The contribution is an open simulator that makes the architecture testable end to end: it generates the aligned multi-modal corpus the interface requires — photorealistic, line art, object index and metric depth from one viewpoint — carries a physically grounded robot with an optional rigid-body contact backend, imports externally authored scenes and assets [3], and executes the real control firmware against the simulated plant at two distinct fidelities: a value-level seam fast enough for the training loop, and an offline register-level gate in which the same commands are driven through emulated I^2C peripherals, H-bridges and a serial lidar protocol [11]. The second catches a class of defect the first cannot represent, and we quantify the divergence between them. We report a perception result on 479 samples, a multi-label scene-description head trained from labels the renderer already knew, and a control policy cloned from a privileged expert that reaches its goal in 8/8 held-out scenes while seeing only the abstraction. Every number is reported beside the score of doing nothing, and we record the two hypotheses about the control bottleneck that measurement destroyed, together with two measurement choices the register-level gate itself invalidated.

1 Introduction

A robot that has learned to drive from photographs has learned, among other things, what its training floor looked like. Change the floor and the policy degrades, not because the geometry changed but because the pixels did. The standard response is more realism and more variety: better renderers, larger domain randomisation, more data. That response treats the symptom.

The alternative is to change what the policy is shown. If perception emits a line drawing — contours, creases, silhouettes — then texture, colour, shadow and specular highlight are not filtered by the policy, they are simply absent from its input. The policy reasons about shape, which is what it needed all along, and the burden of coping with appearance falls on a perception stage that can be trained separately and reused across tasks, morphologies and environments.

This paper makes three contributions. It sets out the chained architecture and the argument for it (Sections 2–4). It describes an open implementation that generates the aligned corpus the architecture requires, closes the loop from policy to bare-metal firmware and onward to emulated peripheral registers, and admits an alternative physics backend and externally authored scenes at the same interfaces (Section 5). And it reports a first perception result, with the negative findings that accompany it (Section 6). We are explicit throughout about what is measured and what remains proposed.

2 Related Work

Abstraction as interface. RT-Sketch conditions manipulation policies on hand-drawn sketches, showing that a deliberately impoverished representation can retain the spatial content a policy needs while discarding what confuses it [5]. Latent-prediction approaches decouple the feature extractor from the control policy so each can be trained independently [4]. Our contribution is to make the intermediate representation explicit and renderable rather than latent, so that it can be supervised directly from a simulator and inspected by a human.

Geometry as a better prior than appearance. E-RayZer learns 3D-aware representations by self-supervised reconstruction with explicit geometry, and finds that the resulting representations transfer to downstream 3D tasks better than leading appearance-based pre-training [2]. That result is about pre-training rather than control, but it supports the same premise: what survives into the representation should be structure.

Staged decomposition. SEIG reconstructs a scene as an executable Blender program by refining geometry, materials, composition and lighting in separate stages, and reports that staging substantially improves fidelity over solving the factors jointly [1]. This is independent evidence for chaining: decomposing a visual problem into scoped stages outperforms asking one model to do everything at once.

Simulators. MuBlE couples MuJoCo physics with Blender rendering for long-horizon manipulation, arguing that existing simulators trade physical accuracy against visual fidelity [3]. Our simulator makes the opposite trade by default: contact is kinematic rather than dynamic, and the effort goes into the modalities and into executing the real control firmware. The two are therefore complementary rather than competing, and we have since built the join in both directions (Sections 5.4–5.5): a MuJoCo backend behind our contact interface, and an importer that renders MuBlE's scenes and assets through our four passes. Where a trade is deliberate, the useful move is to make the alternative available at the same seam rather than to argue for one side of it.

Blender has been used for synthetic training data at scale [12, 13], and NPR and sketch representations have a long history in learned vision [16, 17, 14, 15].

3 Biological Motivation

Human perception does not require photorealism. A line drawing of a chair is recognised instantly, despite carrying no texture, no colour and no lighting. The abstraction is not merely sufficient; it is often faster to interpret than the photograph, because the contours that define shape have already been separated from the surface detail that does not. A representation that presents edges directly is doing work the visual system would otherwise have to do, and it is doing it in a form that cannot be perturbed by a new floor finish.

4 The Chained Architecture

The pipeline has two stages with a fixed, inspectable interface between them (Figure 1).

Stage 1: perception. Camera input → line drawing plus semantic map. Trained on aligned synthetic pairs, this stage absorbs all appearance variation.

Stage 2: control. Line drawing plus semantic map → actions. Trained only on the abstraction, this stage never sees a texture and cannot overfit to one. It is cloned from an expert with privileged access to the true scene, so what is being measured is whether the abstraction carries enough to recover the expert's decision — not whether the expert's decision was wise (Section 7).

The interface is an image, not a latent vector. This matters for three practical reasons: a human can look at it and see what the robot sees; Stage 2 can be trained on rendered line art before Stage 1 exists, which is how the two stages here were in fact developed; and either stage can be replaced without retraining the other.

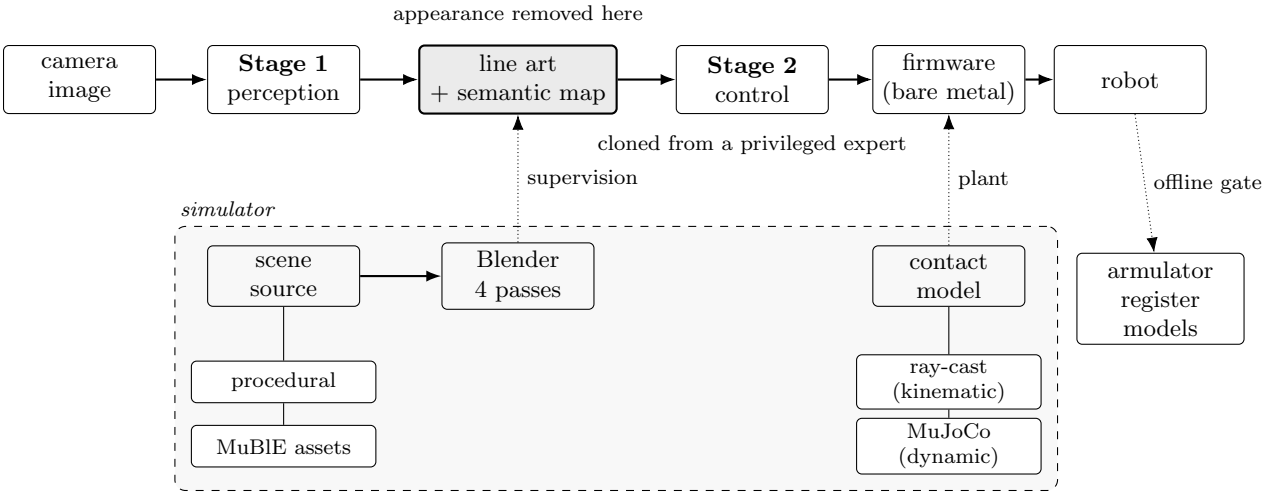


Figure 1: The chain and the two seams that feed it. Everything to the right of the shaded interface sees only line art and a semantic map, so no texture, colour or shadow reaches the policy. The simulator supplies that interface as supervision, and drives the same firmware the robot will run. Scene source and contact model are interfaces with interchangeable implementations: the procedural generator and ray-cast contact are the defaults, MuBlE assets [3] and MuJoCo the alternatives added at those seams. The register models hanging below the robot are not in the tick loop: they re-drive the same commands through emulated peripherals offline, and Section 5.7 reports what that disagrees with.

4.1 The Semantic Bridge

The semantic map is supervised from the renderer's object-index pass, which assigns each pixel the integer identifier of the object it belongs to. The translation from index to token is recorded per sample rather than once per corpus, so a corpus whose labelling changed midway remains readable rather than silently mislabelled. Because the labels are integers written to a float buffer, they survive exactly: recovered values are the assigned indices, with no inference and no quantisation.

5 Implementation

The architecture requires a corpus that does not exist and a simulator that can produce it. Both are released under permissive licences [9, 10].

5.1 Aligned Multi-Modal Capture

Each sample is one scene rendered four ways from a single viewpoint: photorealistic colour, line art, object index and metric depth (Figure 2). They are aligned by construction — the camera does not move between them — and verification re-checks that alignment after the fact, because modalities that disagree on resolution train a network to a systematic offset it can never recover from.

The four are not equally expensive. Depth and object index are render *passes*: the renderer resolves them while shading, so requesting both costs roughly 14% over the colour image alone. Line art is not a pass — it needs its own render with materials flattened — and so costs a second render.

Two engine constraints are worth recording because neither is documented prominently and both silently determine whether the corpus is usable. Of the engines tested, only path-traced Cycles implements the object-index pass; a pipeline built on the real-time rasteriser produces depth but no semantic map. Conversely, on the Blender build used here no light source illuminates a diffuse surface under Cycles, so the photorealistic pass renders black while depth and segmentation remain perfectly correct. The corpus writer therefore checks that the colour pass has tonal range, because an unlit render is the corruption that looks like success: the file exists, the resolution matches, the manifest is complete, and every pixel is black.

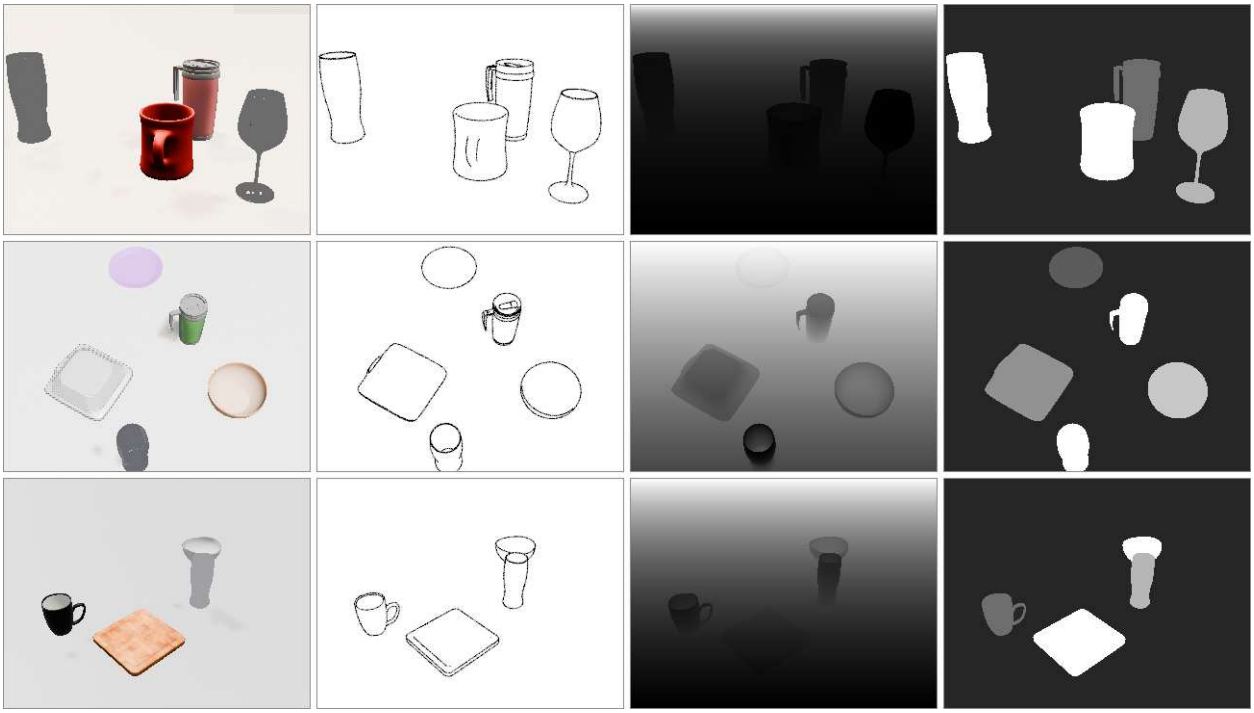


Figure 2: Three samples, each rendered four ways from one viewpoint: colour, line art, metric depth, object index. Alignment is by construction — the camera does not move between them. The geometry here is imported (Section 5.5), so the line art traces a mug's handle and a wine glass's stem rather than the bounding boxes around them. Depth is stored as metric range and shown here as inverse depth, which spends the tonal range on the objects rather than on ground receding to the horizon; the object-index pass assigns each object a distinct integer, shown as distinct grey levels.

5.2 Randomisation and Reproducibility

Layout, materials, lighting and viewpoint are randomised from a recorded seed, so any sample can be regenerated alone. Scale and gravity direction are deliberately held fixed: a policy that cannot trust size or which way is up is worse served than one given no geometry at all. Generation is embarrassingly parallel and measured 2.15 s per sample on one core at 128×96. The implementation moves quickly; the repository is the reference [9].

5.3 The Physical Robot

The same simulator carries a physically grounded robot: ground contact, terrain following, collision with sliding, momentum via acceleration limits, and multi-channel ray-cast lidar with per-beam semantic labels at 3.7 μs per ray. Lidar is cast rather than read from the depth pass for a specific reason: the depth buffer holds planar depth, not range, so a flat wall reads the same value at the centre of frame as at its edge, where the true slant distance is 41% greater at 45°.

5.4 Dynamic Contact

Contact is reached through an interface with a no-op default: drive models command a velocity and ask what pose they actually get, and neither side needs to know how the answer was produced. The kinematic ray-cast model is one implementation. A rigid-body model built on MuJoCo is another, and selecting it is a one-word change at robot construction.

Under the dynamic backend the robot is a mass with wheels. Gravity, collision and momentum come from the solver; traction does not, because a sphere sliding on a plane is not a wheel. Each wheel is instead asked

for the surface speed the commanded twist implies at its own position, and the force that would erase the resulting slip is computed and then clipped to the friction circle μN. That clip is the substantive difference from the kinematic model, which reports slip and then moves the robot the commanded distance regardless. Commanding $1.5\,\mathrm{m\,s^{-1}}$ for two seconds from rest:

μ	**distance**	**final speed**	**grip used**
1.00 (tarmac)	2.856 m	$1.495\,\mathrm{m\,s^{-1}}$	0.02
0.30	2.595 m	$1.495\,\mathrm{m\,s^{-1}}$	0.07
0.05 (ice)	0.786 m	$0.785\,\mathrm{m\,s^{-1}}$	1.00

Table 1: Same command, three surfaces. Where grip is available the dynamic backend tracks the commanded speed and agrees with the kinematic one; where it is not, the shortfall *is* the slip. Peak acceleration is μg whatever the motors are asked for: commanded $50\,\mathrm{m\,s^{-1}}$ from rest at $\mu = 0.3$ produces $0.0465\,\mathrm{m\,s^{-1}}$ in one $1/60\,\mathrm{s}$ tick against a $\mu g \Delta t$ bound of 0.0491.

One implementation detail is worth recording because it is not obvious and it silently defeats the model. MuJoCo combines contact friction between two geoms by taking the *maximum* of the two, not the minimum or the product. A nominally frictionless wheel on a high-friction floor is therefore not frictionless — the floor wins — and the solver's own tangential force ends up opposing the explicit tyre model almost exactly. The symptom is a robot that is grounded, carries full load, reports saturated grip, and still crawls: every individual reading plausible. Every geom is consequently built with a low solver-side friction so that the solver supplies normal force and essentially nothing tangential, leaving μ as the only friction in the model rather than one of two competing sources.

5.5 Importing MuBlE Scenes

The same integration runs in the other direction. MuBlE ships authored Blender assets and a convex-hull decomposition for each object, so its tabletop scenes can be rendered through our four passes rather than approximated. Rendering appends the authored `.blend` per object, giving the geometry and materials MuBlE itself renders; physics loads the shipped hulls into MuJoCo, so the robot collides with a mug's handle rather than with the box around it. Across the five demonstration scenes, all 22 objects resolved shipped geometry, at a mean of 16.0 collision hulls each. Both paths fall back to the bounding box per object, so an incomplete asset checkout degrades rather than fails.

Lighting, world and viewpoint remain randomised over the imported geometry, since appearance variation over fixed structure is the point of the corpus. Imported objects take object-index values disjoint from those the procedural generator uses, and the index-to-token map is recorded per sample rather than once per corpus — different scenes contain different objects, so index 17 genuinely means something different between samples, and a corpus-level map would be wrong for most of them. Recovered index values on imported scenes are exactly the assigned integers, as for procedurally generated ones.

Imported scenes cost roughly 2.3× a procedural sample to render at the same resolution on the same machine, which is the price of real meshes: a single mug carries more geometry than an entire procedural scene.

A caveat on what this does and does not establish. These are measurements of the simulator, not of transfer. They say the dynamic backend behaves as a rigid-body model should and that imported scenes produce a corpus that passes the same alignment and exposure checks as a procedural one. Whether either changes a sim-to-real outcome is the experiment in Section 9, and it has not been run.

5.6 Firmware in the Loop

The control policy is intended to run on ARM or RISC-V microcontrollers under a bare-metal system [8]. That claim is testable before hardware exists: the control application is compiled by crust's host simulator and executed as real machine code against the simulated plant, reading an encoder and writing a motor command while the simulator supplies the physical consequences [10].

This surfaces failures a policy validated only in Python cannot encounter, and they bear directly on any sim-to-real transfer metric. When the encoder reports wheel rotation — as a physical shaft encoder does — the wheels reach commanded speed before the body does and keep turning when the robot is obstructed. Running identical firmware to an identical setpoint, the simulated robot reached 3.10 m while its own dead reckoning reported 4.01 m; when obstructed it reached 1.27 m while still reporting 4.01 m. A stuck-sensor fault produced an open-loop runaway to 28.95 m against a 4.00 m setpoint.

The implication is that a single transfer error conflates two independent sources: the perception error this architecture is designed to eliminate, and the proprioceptive error it does not address at all. They should be reported separately.

5.7 Register-Level Hardware Models

The seam described above is *values*: the firmware calls `sim_motor_write(duty)` and `sim_encoder_read()`, and the simulator supplies the consequences. That is the right shape for asking whether the system behaves, and the wrong shape for asking whether a driver is correct. Firmware that programs a PWM controller's registers incorrectly — wrong auto-increment mode, wrong channel, direction pins swapped — passes a value-level test perfectly, because the seam sits above the registers where the mistake lives.

We therefore added a third fidelity between the value seam and instruction-level emulation. armulator [11] models the peripherals rather than the processor: an I^2C PWM controller at register granularity, the H-bridge truth table it drives, the motor that integrates the result, and a spinning lidar that streams measurement packets over a UART. No instructions are executed at this fidelity, and that is what makes it affordable.

The cost is not why it is offline. The often-quoted figure that instruction emulation runs some 10^4 times slower than real time attaches to stepping AArch64 opcodes through an MMU, not to the peripheral models. Measured against the 16.7 ms budget of a 60 Hz tick, two DC motors driven through the full I^2C-to-shaft path cost 1.8 µs per tick and the lidar at 1800 samples per second costs 43 µs — respectively 0.01% and 0.26% of the budget, and roughly an order of magnitude cheaper than the ray-cast lidar the simulator already runs every tick. The models could run in the loop. They are kept out of it for a different reason: a gate measures the agreement of two independent implementations of the same physical claim, and splicing them together would destroy the only quantity being measured.

The gate. The same commanded profile is run twice through robotsim's own differential-drive integrator, with identical parameters and identical commands. The reference path feeds commanded wheel speeds straight in, as the value seam does. The register path converts each command into a duty cycle and a direction, writes them as controller registers, advances the motor model, and reads back what the shafts actually did. Holding the integrator constant is what makes the comparison attributable: every divergence belongs to the register layer, because nothing else differs.

Over a three-second straight-line profile the register path lags the value seam by 0.144 m, settling to a steady-state wheel-speed disagreement of 0.0006 m/s. Under a 0.6 rad/s turn the two headings diverge by 0.086 rad. At a commanded 0.02 m/s the register path does not move at all.

A metric this destroyed. Our first gate thresholded the peak wheel-speed disagreement, which is the obvious quantity to bound. It reports 0.89 m/s against a 1.0 m/s command and fails every implementation, correct or not: at a step change the register path starts from rest while the value path is already at speed, so the peak error approaches the commanded speed by construction. The only way to make that threshold pass is to delete the spin-up model, which is the one piece of physics worth having. The gate now bounds the steady-state error and reports the peak as a diagnostic.

The sensor half. The lidar is exercised the same way: firmware spins the head by writing PWM registers, issues a scan command, and the model streams five-byte measurement packets back over the UART. The gate decodes those packets off the wire rather than reading the model's internal state, so the comparison covers the duty routing, the spin, the angle and distance encodings and the framing bits. The world behind the sensor is an injected ray-casting callable, so the simulator's own scene can supply it under Blender while the gate runs against exactly-known geometry offline.

A second metric this destroyed. Budgeting the lidar gate against the distance quantisation alone — the obvious choice, since distance is what is being compared — fails a correct implementation by a factor of seventeen. Checking each return against the *bracket* of ranges the world takes across the angular uncertainty

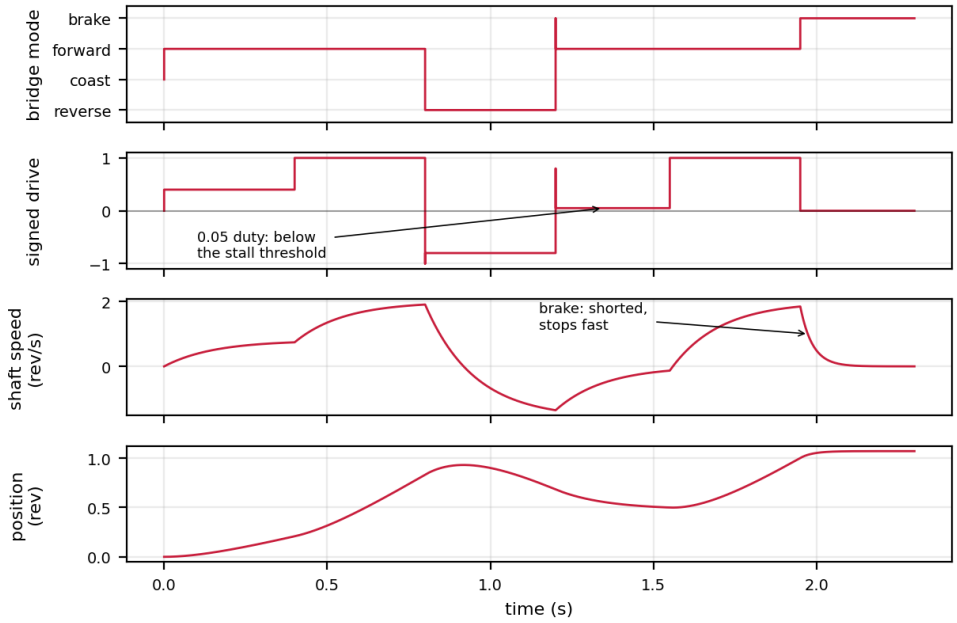


Figure 3: One motor driven through a profile written as PWM register values, recorded at the H-bridge and at the shaft. The bridge mode is a decoded truth table rather than a number: *coast* and *brake* are distinct states, and firmware that means to stop but only coasts is a common defect that a value-level seam cannot express. Two behaviours are annotated because a reader is likely to mistake them for artefacts: a commanded duty below the stall threshold produces drive but no rotation, and braking short-circuits the motor so it stops far faster than it coasts.

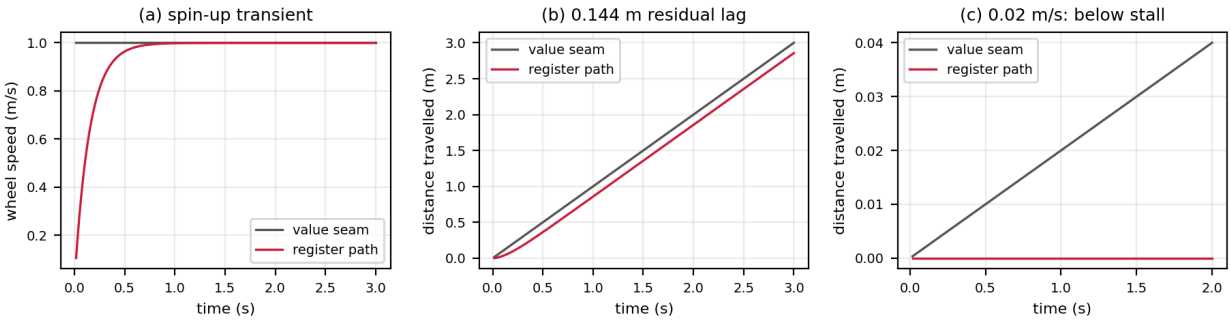


Figure 4: The same commands through both seams. (a) The value seam is at speed instantly; a real motor approaches it. (b) The transient integrates into a permanent displacement offset of 0.144 m over three seconds, which does not recover because the lag is spent at the start. (c) At 0.02 m/s the commanded duty falls below the motor's stall threshold: the value seam crawls forward 0.040 m while the register path does not move at all. This is the divergence that matters most, because it is not a transient and no amount of settling time removes it.

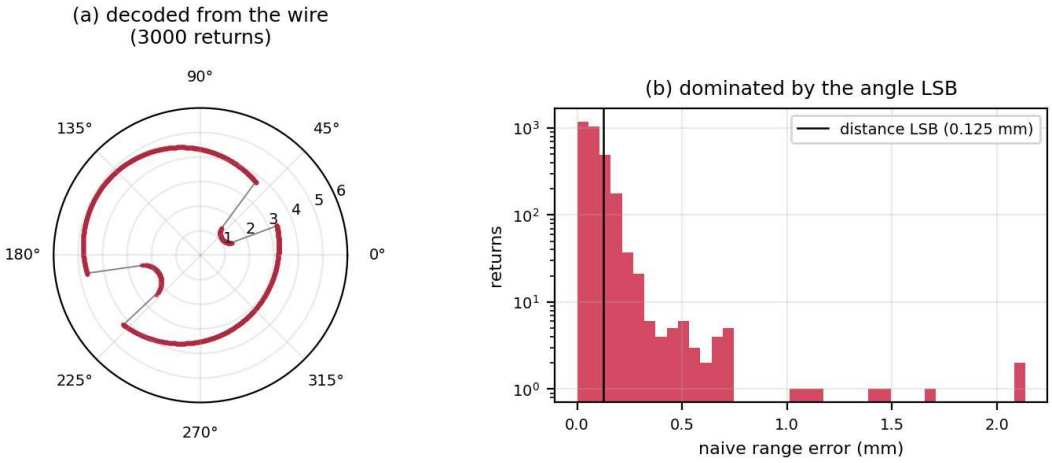


Figure 5: (a) A scan decoded from the serial stream, over the geometry that produced it, with the sensor deliberately off-centre so wall range varies with bearing. (b) Why the tolerance is angular. Ranges travel as quarter-millimetre fixed point, but bearings travel on a 1/64 degree grid, and the range consequence of that is $|\mathrm{d}r/\mathrm{d}\theta|\,\delta\theta$, which is unbounded at grazing incidence. The naive error reaches 2.1 mm, seventeen times the distance resolution, entirely from the angle term.

instead, every one of 3000 decoded returns is consistent with the geometry, with zero residual. The lesson generalises past this sensor: when a quantity is reconstructed from two quantised channels, a tolerance derived from one of them is not a tolerance.

What this is not. No processor is emulated here and no firmware image is executed, so nothing in this section verifies code generation, boot, or the correctness of a compiled binary. It verifies the layer between: that a driver's register writes produce the intended physical effect, and that a sensor's wire format survives the round trip. Instruction-level verification of a deployed image remains future work (Section 9), and this does not substitute for it.

6 Stage 1: Perception

We trained the perception stage on 479 generated samples at 64×48 for 48 epochs, holding out 96 samples for validation. The target is one-pixel-wide line art, which is approximately 99% background.

Accuracy is not usable here. A network that outputs a blank page scores 0.988 accuracy and has learned nothing. We therefore weight ink pixels in the loss and report F1 over ink, always beside what the blank page achieves.

	F1	**precision**	**recall**	**accuracy**
blank page	0.000	—	0.000	0.988
trained, strict	0.308	0.196	0.720	0.970
trained, within 1 px	**0.641**	0.497	0.905	—
trained, within 2 px	0.734	0.604	0.935	—

Table 2: Validation scores, 96 held-out samples. The trained model's *accuracy is worse than the blank page's* while its F1 rises from nothing: one line that settles the choice of metric.

Strict scoring measures localisation, not detection. Strokes are one pixel wide, so a prediction tracing a contour perfectly but one pixel to the left scores zero on both precision and recall for that contour. Allowing the small matching tolerance conventional in boundary detection separates the two, and the jump from 0.308 to 0.641 says the network is finding the edges and placing them within a pixel — visible in Figure 6, where the predictions recover silhouettes, the horizon and the robot outline. We report both

numbers together, and verify that the tolerant metric cannot be gamed: predicting ink everywhere yields recall 1.000 but precision 0.164.

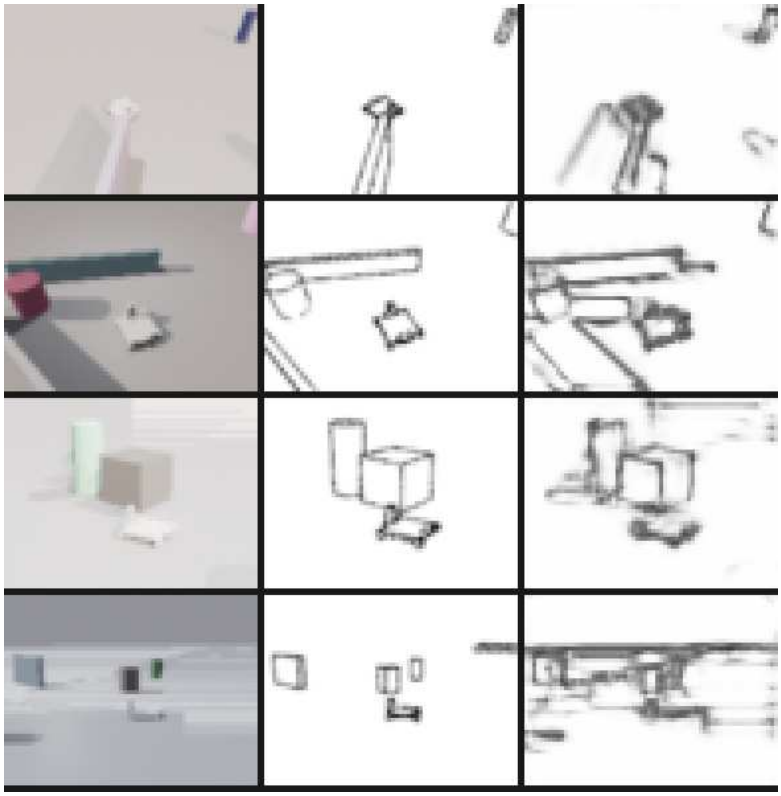


Figure 6: Validation samples: photorealistic input, target line art, and the network's output. The model is under-confident rather than wrong — the soft output traces the structure, and thresholding at 0.5 is what discards it.

6.1 What the Perception Bottleneck Is Not

Three hypotheses were tested and two of ours were wrong. We record them so they are not retried.

Corpus size is not the limit. Going from 140 to 479 samples did not raise the score. Training F1 reached 0.367 against validation 0.305 — a gap of 0.06 — meaning the model cannot fit the training set either. That is underfitting, and more data does not address it.

Receptive field is not the limit. Five 3×3 layers see 11 pixels, which we expected to be too local to judge an object boundary. Dilations of $1, 2, 4, 8$ widen that to 63 pixels and scored *worse* at matched epochs (0.256 vs 0.279 at 12 epochs; 0.307 vs 0.326 at 24).

Threshold tuning does not help. Selected on training data it gave 0.325 on validation against 0.341 at the default. Selecting it on the validation set would have improved the reported number, which is precisely how a tuned threshold becomes an inflated score.

What remains is capacity and the loss. An exact-pixel target punishes a one-pixel miss as hard as a total one, and the tolerance analysis says that is the error the model is making.

6.2 A Third Target: Scene Descriptions

The simulator knows what every object is, where it is and which way up it is. The corpus throws that away: it keeps pixels and an integer per pixel, and the word *mug* never appears. Keeping it costs nothing at generation time and yields a third supervision target beside line art and the semantic map — propositions about the scene, and the sentences that render them.

One rule governs what may be asserted: only what is visible. A caption is supervision for a network whose sole input is the image, so a proposition about an object behind a cupboard, or one pixel wide at the frame edge, is one the network cannot recover. Training on it does not teach occlusion; it teaches guessing, because guessing is the only behaviour that lowers the loss. Every per-object fact is therefore gated on the object's *rendered* pixel count in the object-index pass, which is the one measurement that knows about occlusion, clipping, framing and scale at once. For the same reason image-space facts are read from the mask rather than projected from the camera matrix: projection says where an object would be, the mask says where it is.

Propositions are existential over category rather than per instance. `board:tipped` means some visible board is tipped, so a scene containing two can carry both `board:tipped` and `board:upright` without contradiction. Per-instance targets would need stable identity across randomised scenes, which does not exist; the generated sentences disambiguate in words where the proposition cannot.

Generating the phenomenon is a precondition for supervising a description of it. Every imported scene is upright, so a corpus made from them contains no positive example of `tipped`, and a proposition that never fires cannot be learned — it contributes a constant to the loss and a zero to the score. The generator therefore knocks objects over and re-seats them on the surface.

Trained as a second head on the shared trunk over 180 samples and a 61-item vocabulary, the description head reaches micro F1 0.346 against 0.191 for predicting each proposition's majority class, with macro F1 0.296. The vocabulary is built from the training split alone: a proposition appearing only in validation is a class the model could not have seen, and scoring it is a quiet form of leakage.

7 Stage 2: Closing the Loop

Stage 2 consumes the abstraction and emits a drive command. Its labels come from behaviour cloning: an expert with privileged access to the true pose of the robot, the goal and every obstacle computes a command from geometry, and the student is trained to reproduce it from the rendered abstraction alone. The expert is a potential field and is not a good driver in any general sense, but it is deterministic and inspectable, and it need not be right for the comparison to mean something. What is measured is whether the abstraction carries enough information to recover the decision.

Frame error is not a control result. A cloned policy compounds its own small errors into states the expert never visited and the training set therefore never contained. It can match the expert on every held-out frame and fail the moment it drives. We report both, and the gap between them is the substance of this section rather than an aside: at the point where the policy reached the goal in 0 of 8 scenes, its frame-level error was already comfortably better than the constant predictor.

driver	reached	mean final	mean closest
expert (privileged)	7/8	0.46 m	0.46 m
policy (abstraction only)	8/8	0.29 m	0.29 m
drive straight ahead	0/8	3.22 m	1.85 m

Table 3: Closed loop over eight held-out scenes, every driver on the identical scenes. The policy renders its own observation from wherever it has driven itself. Driving straight is not a silly floor: a goal ahead of the robot is sometimes reached by accident, and a policy that cannot beat it has learned nothing about steering however respectable its frame error.

The policy's 8/8 against the expert's 7/8 is one episode and should not be read as beating the teacher. A plausible mechanism exists — a potential field gets stuck, and a cloned policy can smooth that pathology away — but eight scenes cannot distinguish it from noise.

7.1 What the Control Bottleneck Was Not

Two hypotheses were tested and both were wrong. We record them because both are the intuitive first guess.

The corpus was not too small. At 0/8 reached, the obvious reading was too little data. Doubling the corpus from 270 to 623 frames made the policy *worse* — validation error 0.197 against 0.152 for predicting

the training mean, so worse than ignoring the image. A scaling curve against a fixed held-out set is flat from 13 training episodes to 52. Data was never the constraint, before or after the fix below.

The bottleneck was the pooling. The diagnostic was in the training column: the model could barely fit data it had already seen, while a unit test showed the same architecture memorising four frames to an error of 0.002. That is not underfitting from scarcity, it is an architecture discarding the signal. The policy pooled its feature map to a global mean before the output layer. A mean over the width of a frame says how much goal is visible and not which side it is on — and steering is entirely a question of which side the goal is on. On this corpus the goal's horizontal centroid correlates -0.765 with the expert's commanded yaw rate; the channel mean the policy was given correlates -0.047.

Replacing the global mean with a pooling that keeps horizontal position — mean over height, into bands across the width — took training error from 0.141 to 0.057, steering error from worse-than-constant to 0.169 against 0.228, and closed-loop success from 0/8 to 8/8. Finer banding continued to help, which is the signature of a spatial bottleneck rather than a statistical one.

8 Limitations

The chain closes, but at a scale that should temper every number in this paper. Stage 1 is a small convolutional network trained on a few hundred samples on one CPU core, and Stage 2 on 623 frames from 70 expert rollouts; both should be read as evidence that the apparatus produces learnable data, not as competitive results. The closed-loop evaluation is eight scenes, which is enough to separate a working policy from a stationary one and not enough to separate two working policies.

The end-to-end claim the architecture is ultimately about — that a chained policy degrades more gracefully than a photorealistic one under appearance shift — is still not demonstrated. Both stages exist and both are measured, but they have not been run against a photorealistic baseline on held-out appearance, and that ablation is the experiment the harness was built for.

Stage 2 is trained and evaluated on *rendered* line art rather than on Stage 1's predictions, so the two stages have not yet been composed. The cost of that composition is exactly the quantity the architecture's critics would ask about, and we have not paid it.

The cloned policy inherits the expert's ceiling and the expert is a potential field, which gets stuck; a better teacher would likely move both numbers. The default contact model remains kinematic, and the corpora reported here were generated with it: the rigid-body backend is available but opt-in, so nothing here shows that dynamics changes what a policy learns. The scene importer reconstructs authored meshes and materials but not the source simulator's lighting or camera treatment, so an imported sample resembles its geometry rather than its renders.

The register-level gate is narrow in a way worth stating plainly. It covers one PWM controller, one H-bridge topology and one lidar protocol, and it compares a motor model against a kinematic integrator — two models, not a model against hardware. Neither path has been checked against a physical robot, so the gate establishes internal consistency and the size of the register layer's contribution, not accuracy. The motor dynamics are first order with no current, no torque limit and no thermal behaviour, and the lidar has no noise, no dropout and no reflectivity, so the sensor half tests the encoding rather than the sensing. A gate built on two models can only find disagreements between them; it cannot find an assumption they share.

9 Future Work

The immediate experiment is the ablation the harness now supports end to end: train a baseline policy on photorealistic input and a chained policy on the abstraction, then compare their degradation under lighting, texture and geometry never seen in training. The scene importer sharpens it — evaluating on imported scenes varies asset provenance rather than only lighting — and Stage 2 makes it measurable as task success rather than as pixel error.

Then: composing the two stages, so that Stage 2 is driven by Stage 1's predicted abstraction rather than the rendered one, which is the only way to price the perception stage. Larger control corpora with a better teacher than a potential field. Trajectory corpora generated under the rigid-body backend, where slip along a path has consequences.

Two things follow from the register-level gate. The first is closing the remaining fidelity step: instruction-level verification of the deployed firmware image, which neither the value-level seam nor the peripheral models provide — the models establish that correct register writes produce the intended effect, not that the compiled image emits those writes. The second is running a real driver through the gate rather than the reference driver written for it, which is the point at which the channel-routing and stall defects it is built to catch could actually be caught rather than merely demonstrated.

10 Conclusion

Photorealism asks a policy to learn what to ignore. Supplying structure directly removes the question. We have described a chained architecture built on that principle and released the simulator that makes it testable: aligned multi-modal capture, an optional rigid-body contact backend, externally authored scenes, and the real control firmware executed against the simulated plant at two fidelities — a value-level seam fast enough to train against, and an offline register-level gate that re-drives the same commands through emulated peripherals and reports what the fast seam cannot represent. Both stages are now implemented and measured. A perception network recovers line art and scene propositions from photographs, and a control policy that has never seen a photograph drives to its goal in every held-out scene.

The result we most want to report — that this degrades more gracefully under appearance shift than a photorealistic policy — is still not in hand, and the scales here are small. What has changed is that the question is now an experiment rather than a proposal. Along the way two confident diagnoses of why the control policy was failing were destroyed by measurement, one of them by the data that was supposed to confirm it; both are recorded, because the intuitions behind them are the ones a reader is most likely to share. The register-level gate added two more of the same kind, both measurement choices rather than diagnoses: a speed threshold that no correct implementation could pass, and a sensor tolerance derived from one of the two quantisations that produce the error.

A Appendix: Open Release and Reproducibility

The simulator, the firmware toolchain and the instruction-level emulator are released under permissive licences [9, 10, 11], as are the corpus generator, the perception model and the evaluation harness. Nothing in the pipeline is withheld. Both parts of the control toolchain descend from earlier third-party projects and retain their original copyright notices.

Reproducing a corpus requires the seed and the worker count. Scene content depends on the sample index alone, so the photorealistic, depth and segmentation passes are bit-identical across worker counts; the line pass is not, because Blender's stroke renderer carries state between renders within a process, so changing how samples are divided shifts strokes by up to one pixel. Both values are recorded in the corpus metadata.

The correctness of the perception implementation rests on a numerical gradient check, which agrees with the analytic gradients to 3×10^{-7} in double precision. The check is run in double precision deliberately: a numerical derivative is a difference of two nearly equal numbers, and in single precision the cancellation swamps the result, so the check fails on arithmetic rather than on a wrong gradient.

References

[1] G. He, R. Luo, W.-C. Ma, and H. Averbuch-Elor (2026). *Thinking in Blender: Staged Executable Inverse Graphics with Vision-Language Models. arXiv:2606.02580.*
`https://arxiv.org/abs/2606.02580`

[2] Q. Zhao, H. Tan, Q. Wang, S. Bi, K. Zhang, K. Sunkavalli, S. Tulsiani, and H. Jiang (2026). *E-RayZer: Self-supervised 3D Reconstruction as Spatial Visual Pre-training. CVPR 2026*; *arXiv:2512.10950.*
`https://openaccess.thecvf.com/content/CVPR2026/html/Zhao_E-RayZer_Self-supervised_3D_Reconstruction_as_Spatial_Visual_Pre-training_CVPR_2026_paper.html`

[3] M. Nazarczuk, K. Stepanova, J. K. Behrens, M. Hoffmann, and K. Mikolajczyk (2025). *MuBlE: MuJoCo and Blender simulation Environment and Benchmark for Task Planning in Robot Manipulation. arXiv:2503.02834.*
https://arxiv.org/abs/2503.02834

[4] C. Rizzardo, F. Chen, and D. Caldwell. (2023). Sim-to-real via latent prediction: Transferring visual non-prehensile manipulation policies. *Frontiers in Robotics and AI*, *9*.
https://doi.org/10.3389/frobt.2022.1067502

[5] P. Sundaresan, Q. Vuong, J. Gu, et al. (2024). *RT-Sketch: Goal-Conditioned Imitation Learning from Hand-Drawn Sketches. arXiv.* https://doi.org/10.48550/arxiv.2403.02709

[6] Ekta Rathi (2025) *Impact of Visual Representations (Realistic Vs. Cartoon Vs. No Visuals) On Reading Comprehension in 9–11-Year-Olds* ShodhSamajik: Journal of Social Studies, 2(2), 60–68. https://doi.org/10.29121/ShodhSamajik.v2.i2.2025.42

[7] Rebecca Zhu, et al. (2025) *Cross-Contextual Variability in Children's Early Understanding of Visual Media* https://cogtoolslab.github.io/pdf/zhu_topics_2025.pdf

[8] B.S. Hartshorn (2026) *Full-stackless Computing: A Multi-language Compiler That Builds and Boots its Own Operating System* https://dx.doi.org/10.2139/ssrn.7226482

[9] B.S. Hartshorn. *robotsim: a Blender-based robotics simulator.*
https://github.com/crustos/robotsim

[10] B.S. Hartshorn. *Crust: a multi-language compiler and host simulator.*
https://github.com/brentharts/crust

[11] M. Perelman (2024) *armulator: an AArch64 instruction-level emulator.*
https://github.com/crustos/armulator

[12] E. Wood (2020) *Microsoft using Blender for AI training*
https://develop3d.com/visualisation/microsoft-using-blender-for-ai-training/

[13] E. Wood (2021) *Fake It Till You Make It: Face analysis in the wild using synthetic data alone*
https://arxiv.org/abs/2109.15102v2

[14] Hui Tang, Kui Jia (2023) *A New Benchmark: On the Utility of Synthetic Data with Blender for Bare Supervised Learning and Downstream Domain Adaptation* https://arxiv.org/abs/2303.09165v4

[15] Wilhelm Johannes Kilian, et al. (2024) *A synthetic segmentation dataset generator using a 3D modeling framework and raycaster: a mining industry application* https://doi.org/10.3389/frai.2024.1453931

[16] Seunghoon Hong, Dingdong Yang, Jongwook Choi, Honglak Lee *Inferring Semantic Layout for Hierarchical Text-to-Image Synthesis* https://arxiv.org/abs/1801.05091v2

[17] Yonggang Qi, Guoyao Su, Pinaki Nath Chowdhury, Mingkang Li, Yi-Zhe Song (2021) *SketchLattice: Latticed Representation for Sketch Manipulation* https://arxiv.org/abs/2108.11636v1

Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- generatepaperfigures.py
- generatepaperfigures.py
- paper.tex
- paper.tex