Humanoid Robot Goalkeeper: Driving a Humanoid's Arms from Brain-Recording Encoding Models

Netholabs

29 May 2026

Abstract

We demonstrate a complete, real-time closed loop in which a physical humanoid robot (Unitree G1, 29-DOF) reaches an arm toward an object seen by its own camera, and the decision of *which* arm to raise is computed by a population of **encoding models fit to real extracellular brain recordings** from a behaving animal. A live camera frame is reduced to a single signed scalar—the horizontal offset of a detected ball—which is fed, as a stimulus, into a 20-neuron population model. Each "neuron" is not a hand-tuned unit: it is a Gaussian-of-lag temporal kernel whose parameters (α, τ_{ms}, σ_{ms}) were estimated by fitting that neuron's measured spiking to task variables in a large multi-session, multi-region neurophysiology dataset. The population's summed vote is thresholded into a discrete motor command, realised as a Cartesian arm waypoint solved by inverse kinematics and published to the robot's low-level motor bus. The loop runs end to end on hardware. This is, to our knowledge within the program, the first time these particular biological encoding fits have been used not to *predict* neural activity but to *act*—to close a perception → "neural decision" → motor loop on a real humanoid.

1 What this is

The demo (`ik/lift_arm_for_ball.py`) is a single Python process that wires together four components that were each built and debugged separately over the preceding sessions:

1. **Vision.** An MJPEG stream from the G1's front RealSense camera (served by our `camera/mjpeg_server`, which wraps the Unitree `VideoClient` API because the camera device is exclusively held by `videohub_pc4`).
2. **Object detection.** YOLOv8s detecting the COCO "sports ball" class. The ball's bounding-box centre `cx` is the only thing extracted.
3. **A biologically-derived controller.** The horizontal ball offset is mapped to a stimulus $s \in [-1, +1]$ and pushed through `NeuralController`, a 20-neuron population model whose parameters come directly from brain-recording encoding fits. The population produces a signed *vote*; a deadband turns the vote into `left` / `right` / `stand`.
4. **Embodiment.** The discrete command selects a Cartesian arm waypoint (raise the left arm, raise the right arm, or return to a ready pose). Dual-arm inverse kinematics (Pinocchio + CasADi) solves for joint angles each 20 ms, and these are streamed to the robot at ∼50 Hz.

The conceptual claim is narrow and concrete: **the rule that decides which arm the robot raises is inherited from how real neurons in a real brain responded to the side of a sensory stimulus and to the animal's own choices.** The robot is not running a network we designed to do the task; it is running a slice of a brain's measured stimulus-response geometry, repurposed as a controller.

2 System architecture

2.1 The neural model

2.1.1 Where the parameters come from

The neuron parameters are not invented. They are the output of a Phase-1 encoding analysis (internally codenamed "cat-sanity", `step1a`) over a large neurophysiology corpus. The provenance file (`figures/step1a_cat_sanity_summary.json`) summarises the scale of the fit (Table 1).

The 13 world variables—`stim_side`, `stim_contrast`, `choice`, `feedback`, `wheel_onset`, `wheel_velocity`, `paw_motion/velocity/acceleration`, `pupil_diameter/motion`, `tongue_motion`, `whisker_energy`—together with the brain-region acronyms attached to many units (`CA1`, `CP`, `MRN`,

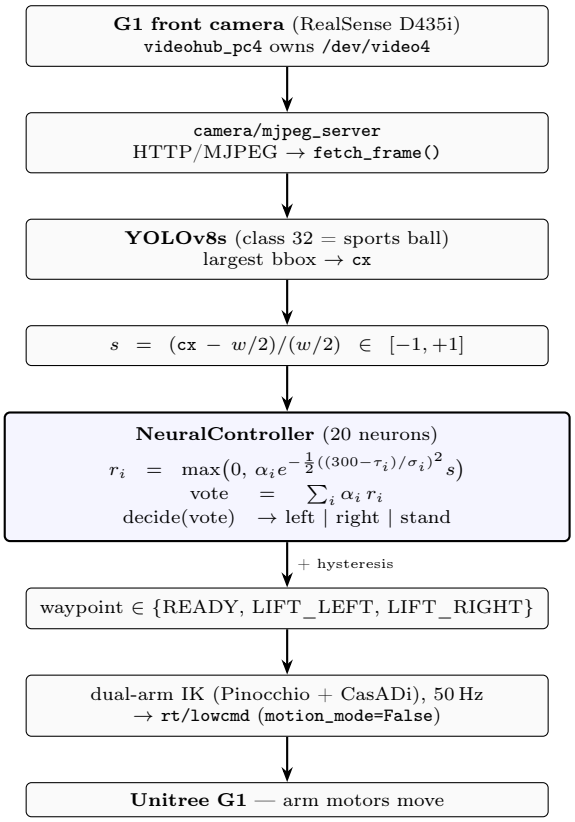

Figure 1: End-to-end pipeline. The vision/neural stages run in a background thread at 5 Hz with hysteresis (3 consecutive agreeing frames to commit a command, 5 empty frames to fall back to `stand`); the IK/publish loop runs in the foreground at 50 Hz and interpolates smoothly between waypoints over 2 s.

`PO`, `SSp-bfd6a`, ...) identify this as a multi-region recording during a **visual perceptual decision-making task**: an animal is shown a stimulus on the left or right, turns a wheel to report a choice, and is rewarded. For each neuron and each variable, a temporal **encoding** model was fit—i.e. a model that predicts the neuron's firing from the world variable—and scored.

Each fitted unit is described by a **Gaussian-of-lag** kernel with three numbers:

- `alpha` (α)—signed gain (the neuron's preferred polarity and strength),
- `tau_ms` (τ)—the lag at which the response is centred (peak latency),
- `sigma_ms` (σ)—the temporal width of the response.

Honesty about fit quality matters here: these are *weak* single-neuron encoders. The corpus-level median session $R^2 \approx 0.002$ and the median session correlation ≈ 0.09. Individual selected units are stronger (e.g. a `choice` unit in `MRN` with $R^2 \approx 0.17$, corr ≈ 0.68), but no one should read this as "the brain explains the behaviour." What the fits *do* capture reliably is **sign and latency structure**—which side a unit prefers, and when it responds—and that is exactly the structure the controller exploits.

Table 1: Scale of the Phase-1 encoding fit.

Quantity	Value
Sessions processed	459
Neurons processed	86,033
Neurons with non-zero α	80,054
Neurons with positive model R^2	53,269
"Strong" neurons (score ≥ 0.10, $R^2 > 0$)	17,663
World (task/behaviour) variables modelled	13

2.1.2 Neuron selection

From the 17,663 "strong" units we keep only the two variables relevant to a left/right action decision and take the cleanest ten of each:

- `world_variable` $\in$ {`stim_side`, `choice`},
- score ≥ 0.10 and model_r2 > 0,
- top 10 per variable by score.

The result—20 neurons, 10 sensory (`stim_side`) and 10 decision (`choice`)—is frozen in `neural_controller/config/selected_neurons.json`, each row carrying its session ID, probe, and cluster ID so the slice is traceable back to specific recorded units.

2.1.3 From neurons to a motor command

For a stimulus s and neuron i, the model evaluates the temporal kernel at a fixed lag of 300 ms (the visual-response window for these variables), projects the signed stimulus onto the neuron's polarity, and half-wave rectifies:

$$r_i = \max\left(0,\ \alpha_i \exp\left(-\tfrac{1}{2}\left(\frac{300-\tau_i}{\sigma_i}\right)^2\right) s\right). \quad (1)$$

The half-wave rectifier is the key biological move: a neuron only fires when the stimulus sign matches its preferred polarity ($\alpha_i\, s > 0$). The population vote sums the rectified rates, weighted again by polarity:

$$\text{vote} = \sum_i \alpha_i\, r_i, \quad (2)$$

and a ± 0.01 dead-band reads out a discrete command:

$$\text{cmd} = \begin{cases} \texttt{right} & \text{vote} > +0.01, \\ \texttt{left} & \text{vote} < -0.01, \\ \texttt{stand} & \text{otherwise.} \end{cases} \quad (3)$$

Because the units carry **mixed-sign** α (15 of the 20 selected neurons are negative-α), the population produces *signed* votes with no extra logic: positive stimuli wake the few positive-α units, negative stimuli wake the many negative-α units. This also produces a deliberate, documented **asymmetry**—the response to a left-side stimulus is $\sim$8.6$\times$ stronger in magnitude than to a right-side one ($|\text{vote}| \approx 2.59$ vs ≈ 0.30)—a direct fingerprint of the recorded population's composition rather than anything we coded. (In the earlier MuJoCo waist-yaw demo this asymmetry had to be normalised per side so the robot didn't slam to one side; in the arm demo it is harmless because the readout is discrete.)

3 Perception and embodiment

Detection. YOLOv8s at confidence 0.10 (the default 0.25 missed the specific ball, which detects at $\sim$0.22). Only the largest sports-ball box is used; its centre defines s. An annotated MJPEG stream (bbox + centreline + the live `stim`/`vote`/command) is served on `:8091` so an observer can watch the neural decision form in real time.

Inverse kinematics. Three torso-frame Cartesian targets per arm (`READY`, `LIFT_LEFT`, `LIFT_RIGHT`) are solved with the suite's `G1_29_ArmIK` (Pinocchio + CasADi), velocity-limited to 2.0 rad/s, and interpolated over 2 s transitions.

Getting the arms to actually move—the hard part. Most of the engineering effort, recorded across `28may.md` and `ik/SESSION_NOTES.md`, was not the neural model but discovering *how to command the arms at all*. The natural path—publishing an `rt/arm_sdk` overlay that blends with the locomotion controller—was published correctly (verified `weight=1.0`, `kp=80`, valid CRC) *but ignored* by the robot's active `ai` controller; a direct bypass ramp moved no joint. The working path, mirrored from `ik_demo_direct.py`, is to call `MotionSwitcher.Enter_Debug_Mode()` to release the high-level controllers and publish **directly to** `rt/lowcmd` (`motion_mode=False`). The cost is that the legs go limp the instant debug mode is entered, so **the robot must be suspended on a crane for the entire run**, and a `try/finally` always hands control back to the `ai` controller on exit. This is the difference between the demo "running" (the pipeline computing correct commands, which it did for days) and the demo "working" (the arms physically moving—the 29 May result).

4 What it means

1. Encoding models can be *run forward* as controllers. Encoding fits are normally evaluated by how well they predict held-out neural activity—a descriptive, backward-looking use. Here the same fitted parameters are used *generatively*: the stimulus-response geometry of a recorded population becomes a function from "what the robot sees" to "what the robot does." The demo is an existence proof that a population of weak, biologically-estimated encoders, when combined, carries enough coherent sign-and-latency structure to drive a stable, correct sensorimotor decision on real hardware.

2. A genuine perception→decision→action loop with a biological middle. The three classic stages of a sensorimotor agent—sense (camera+YOLO), decide (neural population vote), act (IK+motors)—are all present, and the *decide* stage is not a heuristic we wrote. It is inherited. The mixed-sign α population implements signed evidence accumulation "for free," which is conceptually the same job the `stim_side` and `choice` neurons were doing in the original task.

3. A reusable substrate. The controller's input is a single scalar in $[-1, +1]$. Anything that maps "the world leans right $\rightarrow +1$ / left $\rightarrow -1$" can drive it—ball position here, but equally optic flow, a face-tracking offset, or a joystick. The biological decision layer is decoupled from both the sensor and the effector, so the *same* neuron population could steer a waist-yaw in simulation (the original MuJoCo demo) or raise an arm on hardware (this demo) unchanged.

4. A bridge between two programs. It connects the neural-encoding analysis (`neural-encoding-humanoid`, 459 sessions, $\sim$86k units) to a physical robotics stack (`g1-netholabs`) through one small, well-specified interface (`NeuralController`). That interface is the scientifically interesting artefact: it is where measured biology meets actuated hardware.

5 Limitations — what this is *not*

Scientific honesty requires drawing the boundary clearly:

- **This is not a brain-machine interface.** No neural activity is recorded or decoded in real time. The neurons are *frozen parameters* fit offline; the animal is not in the loop. "Neural decision" means *a decision computed by a model whose parameters came from neurons*, not a live readout of intent.
- **The encoders are weak.** Median session $R^2 \approx 0.002$. The system works because a *population* of

weak, consistently-signed units integrates into a robust discrete readout—not because any neuron is a good predictor.

- **The mapping involves engineering choices, not claimed biological fidelity.** Evaluating the kernel at a fixed 300 ms lag, summing $\alpha \cdot$ rate, the ± 0.01 dead-band, the $8.6\times$ sign asymmetry, and the discretisation to three commands are design decisions. They are biologically *motivated* (polarity, latency, rectification) but not biologically *validated* as the brain's own readout.
- **The task is simple.** The decision is, in effect, a deterministic, sign-of-stimulus function. There is no learning, no closed-loop adaptation, and no claim that the robot "understands" the object.
- **The embodiment is constrained.** The robot is craned with legs released; only the arms (and, in the sim variant, waist-yaw) are driven. The arm-control path required bypassing the high-level controller via debug mode.

None of these undercut the result. They define it: a faithful, honest, end-to-end demonstration that *fitted neural-population structure is sufficient to close a real sensorimotor loop on a humanoid*—and a clean platform for making each of these limitations the subject of the next experiment.

6 Future directions

- **Richer readouts.** Use more of the 13 world variables (e.g. `wheel_velocity` for graded turn speed, `feedback` for a reward-modulated gain) and more than 20 units; move from a 3-way dead-band to a continuous, calibrated control law.
- **Graded control.** Drive arm *amplitude* or yaw *rate* from |vote| instead of selecting fixed waypoints, exposing the population's analog structure.
- **Per-region populations.** The units carry brain-region labels; contrast controllers built from different regions (cortical vs. midbrain `choice` units) to ask which regions yield the most stable robot behaviour.
- **Restore overlay coexistence.** Find the controller/FSM state that consumes `rt/arm_sdk` so the robot can move its arms *while balancing on its own legs*, removing the crane constraint.
- **Quantify the loop.** Measure end-to-end latency, decision accuracy vs. ball angle, and robustness to detection noise—turning the demo into a benchmark.

A How to reproduce

```
# Camera server on the robot (build once):
cd /home/unitree/g1-netholabs/camera \
   && make && ./mjpeg_server 8090

# Perception + neural decision only (no robot, no DDS)
    :
/home/unitree/g1-netholabs/ik/run_lift_arm.sh --dry-
    run
# annotated frames -> /tmp/lift_arm_dryrun/
# live overlay -> :8091

# Full loop (ROBOT MUST BE ON A CRANE -
# legs go limp in debug mode):
/home/unitree/g1-netholabs/ik/run_lift_arm.sh
# type 's' + Enter to start;
# Ctrl+C holds last pose then restores ai mode
```

Key files. `ik/lift_arm_for_ball.py` (the loop); `neural-encoding-humanoid/g1-neural-controller/.../neural_controller.py` (the population model); `.../config/selected_neurons.json` (the 20 frozen units); `figures/step1a_cat_sanity_summary.json` (encoding-fit provenance). The full selected-neuron table (α, τ, σ, score per unit) is in `NEURAL_CONTROLLER_README.md`.

The "cat-sanity" / `step1a` codename refers to the Phase-1 encoding-fit package; the world variables and brain-region acronyms in the provenance identify the source as a multi-region recording during a visual decision-making task.