

Neural Canvas: Painting With Bare Hands

Estimating drawing intent from a noisy 60 Hz landmark signal — why a fixed low-pass filter fails, and what the one-euro filter does instead.

Daniel Lichtenberger · 2026

danny-397.github.io/neural-canvas · [Source on GitHub](#)

Painting with bare hands in front of a webcam. The hard problem is not seeing the hand — it is turning a noisy estimate into a line that feels like a pen.

Summary

Neural Canvas is a browser application that turns a webcam into a brush. A neural network locates 21 three-dimensional landmarks on each hand roughly sixty times a second, entirely on-device; pinching thumb to index finger puts the pen down; the index fingertip draws. Gestures erase, change color, undo, and clear. Optional kaleidoscopic symmetry mirrors every stroke into a live mandala, and a body mode swaps the hand model for a full-body pose model so the whole body paints.

It was built for Hack the Arts 2026, whose theme was art that would not exist without technology. The design consequence of that theme is that hand tracking is not a control scheme bolted onto a paint program — it is the medium, and every decision below follows from taking that literally.

The entire application is one HTML file: no framework, no build step, no backend, no server-side anything. Nothing leaves the device.

Model	MediaPipe Tasks Vision — <code>GestureRecognizer</code> (hands), <code>PoseLandmarker</code> (body)
Landmarks	21 per hand in 3D, up to 2 hands; 33 for full-body pose
Execution	WebAssembly + GPU delegate, <code>runningMode: VIDEO</code> , in-browser
Rendering	Canvas 2D, additive blending, shadow-blur glow
Smoothing	One-euro adaptive filter, implemented from scratch
Source	One file, ~510 lines, zero dependencies beyond the model CDN

1. The actual problem

Getting hand landmarks is a solved problem you can import. Within an hour of starting, the fingertip position is on screen and tracking works.

Then you draw a line with it, and the line is unusable.

Landmark estimates carry per-frame noise of a few pixels even when the hand is perfectly still. Drawn directly, a "straight" line comes out visibly furry, and a slow deliberate curve — exactly what someone does when signing their name or outlining a shape — is the worst case, because the jitter amplitude is comparable to the motion itself. The signal you want and the noise you don't are the same size.

So the engineering problem is not detection. It is **estimating intent from a noisy 60 Hz position signal in real time**, and it is a signal-processing problem with a hard constraint: any smoothing you apply costs latency, and latency in a drawing tool is felt immediately. The stroke lags the finger, and the tool stops feeling like an extension of the hand.

2. Why a fixed low-pass filter fails

The obvious fix is an exponential moving average over the fingertip position. It does not work, and the reason it does not work is instructive.

A fixed-cutoff filter forces a single global choice:

- **Smooth heavily** — clean lines when the hand is still, and unacceptable lag when it moves. The stroke trails visibly behind the fingertip.
- **Smooth lightly** — responsive when moving, and visible jitter when holding still.

Both failure modes are unacceptable, and they are unacceptable *at the same time*, because a single stroke contains both regimes: a slow careful start, a fast sweep, a pause at a corner. No constant is right for the whole stroke.

The insight is that the correct amount of smoothing is not a constant — it is a function of speed. When the hand is nearly stationary, almost all measured variation is noise, so smooth hard. When the hand is moving fast, the motion dominates the noise and the user is prioritizing responsiveness over precision, so barely smooth at all.

3. The one-euro filter

That is exactly what the one-euro filter does (Casiez, Roussel & Vogel, 2012), and it is implemented here from scratch in nine lines:

```

class OneEuro{
  constructor(minCut=0.9, beta=0.04, dCut=1.0){ ... }
  a(cut,dt){const tau=1/(2*Math.PI*cut);return 1/(1+tau/dt);}
  f(x,t){
    const dx=(x-this.xP)/dt, aD=this.a(this.dC,dt), dxH=aD*dx+(1-aD)*this.dXP;
    const cut=this.mC+this.beta*Math.abs(dxH), a=this.a(cut,dt), xH=a*x+(1-a)*this.xP;
    ...
  }
}

```

The mechanism is a low-pass filter whose cutoff frequency is itself derived from the smoothed velocity estimate:

$$\text{cutoff} = \text{minCutoff} + \text{beta} \cdot |\text{velocity}|$$

At rest, velocity is near zero, cutoff falls to `minCutoff`, and the filter smooths aggressively — jitter disappears. During a fast sweep, the velocity term dominates, cutoff rises, the smoothing factor approaches 1, and the filter passes the signal through almost untouched — no perceptible lag. The velocity estimate is itself low-pass filtered at `dCut`, because a raw frame-to-frame derivative of a noisy signal is noisier than the signal.

Crucially, `dt` is measured per frame from `performance.now()` rather than assumed. Frame intervals are not uniform — the browser drops frames, the model occasionally takes longer — and a filter that assumes a fixed timestep changes its effective cutoff whenever the frame rate wobbles, which is precisely when it is most visible.

Tuning was empirical and the history is in the commits. The filter shipped at `minCutoff 1.3`, `beta 0.05` and was retuned to `0.9 / 0.04` after testing on a real webcam (`6db79e7`) — smoother at rest, at the cost of slightly more lag during fast motion. That tradeoff is the entire design space of the filter, and there is no setting that escapes it.

4. Tracking dropouts and stroke continuity

A second failure mode has nothing to do with noise. The detector occasionally loses the hand for one or two frames — a motion blur, an awkward angle, a hand crossing the frame edge.

Handled naively, the stroke ends. The pen lifts and comes back down, and a continuous signature is chopped into pieces at random. This is far more destructive to the artwork than jitter, because the damage is structural rather than cosmetic.

So the pen state carries a grace period: a hand that goes missing increments a counter, and the stroke is only terminated after **more than five consecutive missed frames** — roughly 80 ms. Below that threshold the pen is held and the stroke survives the dropout.

Relatedly, when only one hand is needed the model is configured with `numHands: 1` rather than 2. A single-hand model is measurably more stable, because the detector is not attempting to resolve a second hand from ambiguous evidence and occasionally flickering one into and out of existence.

5. Gesture vocabulary

The theme argued for no UI chrome between the person and the drawing, so the primary controls are gestures, classified by the model's own gesture head rather than by hand-rolled landmark geometry:

Gesture	Action
Pinch (thumb + index)	Pen down — the fingertip draws
👐 Open palm	Erase under the palm
✊ Closed fist	Clear the canvas
👍 Thumbs up	Clean view — hide camera and overlays, pause painting
✌ Victory	Next color
👎 Thumbs down	Undo

Destructive gestures need debouncing, and the two thresholds are different on purpose. Clearing the canvas requires the fist to be held for **14 consecutive frames** and at least 1,500 ms since the last clear — a momentary misclassification cannot destroy the drawing, and the deliberate hold makes the intent unambiguous. Non-destructive gestures (color, undo, view) fire on a single classification with a one-shot latch, because a false positive there costs nothing and waiting would feel sluggish.

Clean view exists specifically for the demo and for unattended display: 👍 hides the camera feed, the skeleton overlay and all UI, and pauses painting so a stray hand in frame cannot damage the finished piece. It is the gesture that turns the tool into a gallery.

6. Symmetry

Symmetry is applied as a set of coordinate transforms on the stroke path rather than as a post-process on the rendered image, so every brush effect — glow, taper, speed-dependent width — is preserved in every reflection.

For N-fold symmetry the transform list contains, for each wedge `k`, a rotation by `2πk/N` and the same rotation composed with a vertical flip. That mirrored second copy is what makes the output a kaleidoscope rather than merely a rotation — adjacent wedges are reflections of each other, which is the defining property of the form. Mirror mode is the degenerate case: identity plus a horizontal flip about the canvas center.

Five brushes — neon, ribbon, spray, comet, embers — differ in stroke-accumulation model rather than merely color, and stroke width responds to hand depth (the z coordinate of the landmark) and to drawing speed, so a fast sweep tapers the way a real brush does.

7. Body mode

Body mode swaps the hand model for `PoseLandmarker` and paints from the wrists. Two design details carry it:

Pen-down needs a whole-body analogue of pinch. There is no pinch in a pose skeleton, so the condition is *wrist above the corresponding shoulder* — comparing landmark 15 against 11 for the left side, 16 against 12 for the right. This is readable to an audience with no instruction, self-evident on camera, and physically comfortable to sustain.

Occluded joints must be rejected. Pose landmarks carry a visibility score, and an occluded wrist still reports a position — a confident-looking, completely wrong one. Any landmark below 0.35 visibility is discarded rather than drawn, which prevents phantom strokes when an arm passes behind the torso.

8. Constraints as design

The whole application is one HTML file with no build step, no bundler, no framework and no backend. The model loads from a CDN and runs through WebAssembly with a GPU delegate.

This is a real constraint, not an aesthetic pose, and it buys three things. Inference is on-device, so **no video ever leaves the machine** — the privacy property is architectural rather than a promise in a policy. There is no server to run, so the piece cannot break because something expired. And the source is one readable file, which for a hackathon submission means a judge can inspect the whole thing without a toolchain.

The cost is real too: no module system, no type checking, and a 510-line file that is at the upper limit of what is comfortable to navigate.

9. Limitations

Latency is not measured. The tuning above is empirical and judged by feel on one machine and one webcam. There is no instrumented end-to-end measurement from photon to pixel, and "60 inferences per second" is the model's target on the development machine, not a benchmarked figure across devices. On a low-end laptop without a usable GPU delegate the frame rate drops substantially and the filter — whose `dt` is measured, not assumed — smooths correctly but the whole experience degrades.

Gesture classification misfires. The model's gesture head is not perfectly reliable at odd angles or in poor lighting; the debouncing in §5 is a mitigation, not a fix. Clearing the canvas by accident is prevented; a color change firing unbidden is not.

Lighting is a hard dependency. Strong backlighting or a dim room degrades detection badly. There is no fallback input mode — if tracking fails, the application has nothing to offer.

No user study. Every claim about how the tool *feels* is my own judgment plus informal observation of people trying it. "Feels like a pen" is the design target and the evaluation criterion, and it has not been measured against anything.

Undo is shallow and bitmap-based. Snapshots are pushed at stroke start rather than maintaining a vector stroke list, which bounds how far back undo can go and makes it memory- hungry at high canvas resolutions.

10. What I would do differently

Instrument the latency properly — a visible timestamp overlay and a high-frame-rate capture of hand and screen together would turn the filter tuning from a matter of taste into a measurement, and would let the `minCutoff` / `beta` tradeoff be chosen from data rather than from feel. Keep strokes as vectors rather than bitmap snapshots, which fixes undo and makes resolution-independent export possible. And add a pointer fallback so the piece still does something when the camera fails, since right now a failed detection is a dead end.