OpenMTL Bus Tracking

Mathematical Evolution of the Motion-Correction Algorithm

Jessee Lord Dushime

March 11, 2026

Abstract

This document records the evolution of the OpenMTL bus-position correction algorithm across six stages. The goal was not merely to make markers look smoother, but to reconcile two competing signals in a principled way: *(1)* a model-based prediction that advances continuously between polls, and *(2)* a sparse GPS observation arriving every few seconds. Each iteration solved a specific defect observed in the previous one: teleportation, visible sliding, oscillation, stiffness, segment-boundary discontinuity, and finally the need for globally distributed correction. The final stage is a horizon-aware additive controller that spreads tracking error across the entire remaining route using a precomputed correction rate, achieving smooth segment-invariant convergence at $O(1)$ per frame.

Contents

1 Context and problem

OpenMTL displays buses moving along a route in real time. However, the rendering loop and the GPS feed do not operate in the same way:

- the **visual bus** must move continuously every frame;
- the **GPS position** arrives only at polling boundaries;
- the **model prediction** provides an expected forward motion between two GPS observations.

This creates a core problem:

> How can the displayed bus remain visually continuous while still converging toward reality when a new GPS point arrives?

The challenge is not just interpolation. A naive update may produce a marker that jumps, slides unnaturally, accelerates abruptly, or oscillates between prediction and observation. Over time, the work evolved from a basic visual fix into a genuine motion-control problem.

2 Notation

The following notation is used throughout the document.

Symbol	Definition
$x(t)$	displayed bus distance along the route
x_g	latest GPS-measured route distance at poll time
$x_r(t)$	moving reference generated by the predictive model
v_{ml}	model-predicted speed along the current segment
T	polling interval
$e(t)$	tracking error (defined per-stage against GPS or moving reference)

The central design tension is:

$$\underbrace{\text{prediction}}_{\text{gives continuity}} \longleftrightarrow \underbrace{\text{GPS}}_{\text{gives truth.}}$$

The algorithm must preserve both.

3 Stage 1 — Instant re-anchor

Stage 1 — Instant re-anchor

The first version simply snapped the visual position to the newly received GPS distance:

$$x \leftarrow x_g.$$

3.1 What this stage was trying to solve

At this point, the priority was correctness. When the GPS reported a different position, the rendering should immediately reflect it.

3.2 Why it was insufficient

This solved the state error instantly, but produced a position discontinuity — mathematically not even C^0; visually, a teleport.

3.3 Reasoning that led to the next stage

> Can I keep the exact correction target while avoiding the visible jump?

That naturally led to blending the correction over time.

4 Stage 2 — Position offset blend

Stage 2 — Position offset blend

A decaying positional offset is subtracted over one poll interval.

4.1 Formula

When a GPS update arrived, the discrepancy between the predicted and observed position was stored:

$$\text{error} = x_{\text{pred}} - x_g.$$

A decaying position offset was then subtracted over the course of one poll interval:

$$\text{offset}(t) = \text{error}\,\big(1 - \text{easeOut}(t)\big), \quad \text{where} \quad \text{easeOut}(t) = 1 - \left(1 - \frac{t}{T}\right)^2.$$

The displayed distance became

$$x(t) = x_{\text{advance}}(t) - \text{offset}(t).$$

At the boundaries:

$$t = 0 \;\Rightarrow\; \text{offset}(0) = \text{error} \quad \text{(no jump)}, \qquad t = T \;\Rightarrow\; \text{offset}(T) = 0 \quad \text{(aligned)}.$$

4.2 Why it was better

Position is now continuous. The bus no longer jumps at the poll boundary.

4.3 Why it was still wrong

Although continuous, the motion was not physically convincing. The correction acted as a direct *position shift*, not as a velocity change — the bus looked like it was being *dragged*.

4.4 Reasoning that led to the next stage

> A transport object should be corrected through *motion*, not through a sliding positional warp.

That reframed the problem: instead of altering *where* the bus is, alter *how fast* it moves.

5 Stage 3 — Constant speed multiplier

Stage 3 — Constant speed multiplier

Positional offset is replaced by a speed multiplier μ.

5.1 Formula

At GPS arrival:

$$\text{error} = x_{\text{visual}} - x_g.$$

With normal travel $\text{normalTravel} = v_{\text{base}} \cdot T$, the correction multiplier is:

$$\mu = \frac{\text{normalTravel} - \text{error}}{\text{normalTravel}} = 1 - \frac{\text{error}}{\text{normalTravel}}.$$

The effective speed becomes

$$v_{\text{eff}} = v_{\text{base}} \cdot \text{clamp}(\mu,\, 0.82,\, 1.18).$$

5.2 Conceptual improvement

> Correction should act on *velocity*, not on *position*.

5.3 Why it failed

The multiplier μ remained constant during the whole interval, producing:

1. **Overshoot:** strong correction crossed the true position, so the next poll found the bus on the opposite side.
2. **Oscillation:** the sign of the error flipped, the multiplier flipped, and the system entered a repeated speed-up / slow-down loop.

5.4 Reasoning that led to the next stage

> Do not try to fix the full error in one cycle. Correct only part of it, and relax the correction smoothly.

6 Stage 4 — Damped multiplier with partial gain

Stage 4 — Damped multiplier with partial gain

Only a fraction γ of the error is corrected; the multiplier then decays back to 1.

6.1 Formula

First, correct only a fraction of the observed error:

$$\text{correctedError} = \gamma \cdot \text{error}, \qquad \gamma = 0.6.$$

Then

$$\mu_0 = \frac{\text{normalTravel} - \text{correctedError}}{\text{normalTravel}}, \qquad \mu_0 = \text{clamp}(\mu_0,\, 0.85,\, 1.15).$$

With $\gamma = 0.6$, repeated application yields geometric error decay:

$$e_n = e_0\,(1-\gamma)^n = e_0 \cdot 0.4^n.$$

The multiplier then decays toward nominal speed over the interval:

$$\mu_{\text{eff}}(t) = 1 + (\mu_0 - 1)\left(1 - \frac{t}{T}\right)^2,$$

so that

$$v_{\text{eff}}(t) = v_{\text{base}} \cdot \mu_{\text{eff}}(t), \qquad \mu_{\text{eff}}(0) = \mu_0, \qquad \mu_{\text{eff}}(T) = 1.$$

6.2 Why it was better

Compared with Stage 3, this version was significantly more stable. The speed profile was also smoother because the effective multiplier decayed continuously over time.

6.3 Remaining limitations

1. **Hard saturation:** very different large errors collapsed to the same clamped correction.
2. **Boundary stitch:** since $\mu_{\text{eff}}(T) = 1$, the correction disappeared exactly at the poll boundary.

6.4 Reasoning that led to the next stage

> Instead of a manually clipped linear controller, use a smooth bounded nonlinear function that behaves gently near zero and saturates naturally for large errors.

7 Stage 5 — Tanh sigmoid correction

Stage 5 — Tanh sigmoid correction

A smooth saturating nonlinearity replaces ad hoc clamping.

7.1 Formula

Stage 5 — Core controller

$$v_{\text{eff}} = v_{ml}\left(1 + \delta\, \tanh\left(\frac{e}{\lambda}\right)\right)$$

- v_{ml}: model-predicted segment speed
- $e = x_r - x$: signed tracking error
- $\delta = 0.30$: maximum relative correction amplitude
- $\lambda \approx 40\,\text{m}$: error scale (in route-distance units)

The moving reference evolves as $\dot{x}_r = v_{ml}$, and the error dynamics are:

$$\dot{e} = \dot{x}_r - \dot{x} = -\delta\, v_{ml} \tanh\left(\frac{e}{\lambda}\right).$$

7.2 Designed properties

1. **Smoothness everywhere** — no hard clamp boundaries.
2. **Bounded correction** — no risk of infinite acceleration.
3. **Self-regulating gain** — gentle when close, stronger when far.
4. **Always active** — no forced decay to exactly 1.0 at the poll boundary.

> Do not schedule correction in time. Make correction a direct smooth function of the current error state.

7.3 Local behavior for small errors

For small $|e|$, $\tanh(e/\lambda) \approx e/\lambda$, so

$$\dot{e} \approx -\frac{\delta v_{ml}}{\lambda}\, e \quad \Longrightarrow \quad e(t) \approx e(0) \exp\left(-\frac{\delta v_{ml}}{\lambda}\, t\right).$$

The time constant is $\tau = \lambda/(\delta\, v_{ml})$.

7.4 Behavior for large errors

For $|e| \gg \lambda$, $\tanh(e/\lambda) \approx \text{sign}(e)$, giving

$$\dot{e} \approx -\delta\, v_{ml}\, \text{sign}(e).$$

The system converges at a near-constant rate — decisive recovery without explosive response.

7.5 Speed bounds

Since $\tanh(\cdot) \in (-1, 1)$:

$$(1-\delta)\, v_{ml} \;\leq\; v_{\text{eff}} \;\leq\; (1+\delta)\, v_{ml}.$$

With $\delta = 0.30$, the bus never stops, reverses, or spikes to an unrealistic corrective speed.

7.6 Lyapunov stability

Consider the candidate Lyapunov function $V(e) = \frac{1}{2}e^2$. Then

$$\dot{V}(e) = e\dot{e} = -\delta\, v_{ml}\, e\, \tanh\!\left(\frac{e}{\lambda}\right) \;\leq\; 0,$$

with equality only at $e = 0$. The equilibrium is globally asymptotically stable in the continuous ideal model.

7.7 Remaining structural flaw

The correction $\delta\, v_{ml} \tanh(e/\lambda)$ is *multiplicative* in v_{ml}. When the bus crosses a segment boundary where v_{ml} changes discontinuously, the absolute correction magnitude jumps too — causing a visible slide in the displayed position, especially on routes alternating between fast arterial and slow residential segments.

7.8 Reasoning that led to the next stage

> Correction should be *additive* to the speed, not *multiplicative*. And it should spread the error across the entire remaining route — not just the current segment.

8 Stage 6 — Horizon-aware additive correction

Stage 6 — Horizon-aware additive correction (current production stage)

A single constant velocity offset, computed once per poll, distributes the tracking error uniformly across the entire remaining route.

8.1 Core idea

Instead of scaling the ML speed by a multiplicative factor, the correction is an *additive* velocity term, computed once per GPS poll and held constant until the next poll. The correction magnitude is determined by how much error exists and how much route remains to absorb it.

8.2 Definitions

Symbol	Definition
e	signed tracking error: GPS distance minus visual distance
$\lambda = 40\,\text{m}$	error scale parameter (route-distance units)
$\delta = 0.30$	maximum relative speed deviation (safety bound)
$v_{ml}^{(k)}$	model-predicted speed for segment k
d_k^{rem}	remaining distance in segment k from current position
d_k	total distance of downstream segment k

8.3 Horizon computation

The **horizon** H is the total remaining travel time (ms) from the current position to the end of the last known downstream segment:

$$H = \frac{d_{k_0}^{\text{rem}}}{v_{ml}^{(k_0)}} + \sum_{k=k_0+1}^{K} \frac{d_k}{v_{ml}^{(k)}},$$

where k_0 is the current segment and K is the last known downstream segment. This quantity is the *time budget* available to correct the error.

8.4 Correction rate formula

Stage 6 — Final formula

$$r = \frac{\tanh\big(|e|/\lambda\big) \cdot |e|}{H}, \qquad v_{\text{eff}} = v_{ml}^{(k)} + r, \qquad H = \sum_{j \geq k} \frac{d_j}{v_{ml}^{(j)}}$$

with safety clamp $(1-\delta)\, v_{ml}^{(k)} \leq v_{\text{eff}} \leq (1+\delta)\, v_{ml}^{(k)}$.

The sign of r follows the error direction: positive e (bus behind) $\Rightarrow$ speed up; negative e (bus ahead) $\Rightarrow$ slow down.

8.5 Why the correction is additive

The term r has units of distance per millisecond — a constant velocity offset, independent of the current segment. When the bus crosses a segment boundary:

- v_{ml} changes (different segment speed);
- r does *not* change (cached at poll time).

Compare with Stage 5: if v_{ml} doubles at a boundary, the Stage 5 correction $\delta v_{ml} \tanh(e/\lambda)$ doubles too — a visible speed jump. In Stage 6, r stays the same.

8.6 Analysis of the correction rate

8.6.1 Behavior for small errors

For $|e| \ll \lambda$, $\tanh(|e|/\lambda) \approx |e|/\lambda$, so

$$r \approx \frac{|e|^2}{\lambda H}.$$

This is *quadratic* in the error — extremely gentle for small discrepancies. A 5-meter error on a route with 20 minutes remaining produces a correction invisible to the eye.

8.6.2 Behavior for large errors

For $|e| \gg \lambda$, $\tanh(|e|/\lambda) \approx 1$, so

$$r \approx \frac{|e|}{H}.$$

The natural rate: spread the entire error uniformly across remaining travel time. An 80-meter error with 15 minutes of route left corrects at $\approx 0.089\,\text{m/s}$ — barely perceptible.

8.6.3 Horizon scaling

$$r \propto \frac{1}{H}$$

- **Long route remaining** (H large): gentle correction, distributed over many segments.
- **Near terminus** (H small): more aggressive correction — less distance to absorb the gap.
- **At terminus** ($H \to 0$): falls back to Stage 5 multiplicative form as a safety measure.

8.7 Mode transitions and guard conditions

The controller operates in three modes:

Mode	Condition	Effective speed
Predicted	$\lvert e\rvert < \varepsilon_{\text{forgive}}$	$v_{ml}^{(k)}$
Correcting	$\varepsilon_{\text{forgive}} \le \lvert e\rvert < \varepsilon_{\text{abandon}}$	$v_{ml}^{(k)} + r$
Fallback	$\lvert e\rvert \ge \varepsilon_{\text{abandon}}$ or $n_{\text{growing}} \ge 6$	0

Parameters: $\varepsilon_{\text{forgive}} = 5\,\text{m}$, $\varepsilon_{\text{abandon}} = 120\,\text{m}$, $\lambda = 40\,\text{m}$, $\delta = 0.30$.

The growing-error guard ($n_{\text{growing}} \ge 6$) detects when correction is diverging across consecutive polls and gracefully abandons the prediction.

8.8 Computational cost

A critical design requirement: the 60 fps rendering loop must remain $O(1)$ per vehicle per frame.

- H: iterate over the segment map once — $O(K)$, $K \le 40$ typically. Computed *once per poll* ($\sim$10 s).
- r: single arithmetic expression. Computed *once per poll*.
- $v_{\text{eff}} = v_{ml} + r$ with a clamp: $O(1)$ per frame.

For 200 active vehicles at 60 fps this is 12,000 additions per second — negligible.

8.9 Properties satisfied

Property	Stages 1–5	Stage 6
Position continuity	partial	✓
Bounded v_{eff}	partial	✓
C^∞ in error	Stage 5 only	✓
Segment-boundary invariance	×	✓
Global route awareness	×	✓
$O(1)$ per frame	✓	✓

9 Summary of the evolution

Stage	Core idea	What it solved	Main limitation
1	Instant snap to GPS	Exact correction	Teleportation / discontinuity
2	Blend positional offset	Removes jump in position	Velocity discontinuity; dragging look
3	Constant speed multiplier	Correction through motion	Overshoot and oscillation
4	Partial gain + decay	Better stability, smoother recovery	Hard clamp; correction dies at boundary
5	Nonlinear tanh controller	Smooth, bounded, self-regulating	Correction jumps at segment boundaries
6	Horizon-aware additive	Segment-invariant global correction	Current production stage

10 Engineering narrative: what changed in the thinking

The most important evolution was not only mathematical — it was conceptual.

From correctness to continuity

The first concern was simple correctness: *"put the bus where GPS says it is."* But that immediately exposed the cost of discontinuity.

From continuity to physical plausibility

Stage 2 made the motion continuous, but the bus looked manipulated. The deeper requirement emerged: the marker should not merely *end* at the right place; it should *get there* in a way that

resembles actual vehicle motion.

From visual patch to control problem

By Stage 3, the problem had become a feedback problem. The visual system was being controlled through speed rather than patched after the fact.

From aggressive correction to stable convergence

Stage 4 recognised that full correction per cycle is often too aggressive in sampled systems. Partial correction and damping turned a brittle controller into a manageable one.

From manual tuning to nonlinear regulation

Stage 5 replaced ad hoc clamp logic with a smooth saturating nonlinearity. This is the point where the method stops looking like a UI trick and starts looking like a genuine control law.

From local control to global trajectory planning

Stage 6 was the final conceptual leap. Instead of asking *"how should I adjust speed on this segment?"*, the question became *"how should I distribute this error across the entire remaining route?"*

The correction stopped being a reactive local loop and became a planned trajectory adjustment. The horizon H encodes global knowledge about the route ahead, and the correction rate r is a single precomputed constant that applies uniformly regardless of which segment the bus is traversing.

11 What remains true even in the current stage

The current controller is the strongest version so far, but several realities remain:

- the implementation is still discrete in time, even if the analysis is continuous;
- GPS observations may be noisy or delayed;
- route projection and segment transitions can introduce error sources not caused by the controller itself;
- the horizon H depends on the accuracy of the ML speed predictions — if the model is systematically wrong about future segments, H will be miscalibrated.

So the current algorithm is best understood as a strong motion-correction layer built on top of an ML prediction model, not a standalone state estimator. Future work could incorporate filtered residuals, confidence weighting, adaptive scaling of λ, or a Kalman-style fusion of prediction and observation.

12 Conclusion

Across these iterations, the OpenMTL correction logic evolved through a clear sequence:

snap $\rightarrow$ position blend $\rightarrow$ speed control $\rightarrow$ damped correction $\rightarrow$ tanh feedback $\rightarrow$ horizon-aware additi

Each step emerged from an observed failure mode in the previous version, reflecting a genuine engineering process: identify the artifact, isolate its cause, redesign the correction rule around a better model of motion.

Final formula

$$v_{\text{eff}} = v_{ml}^{(k)} + \frac{\tanh\big(|e|/\lambda\big) \cdot |e|}{H}, \qquad H = \sum_{j \geq k} \frac{d_j}{v_{ml}^{(j)}}.$$

It combines six desirable properties in a single expression: **continuity**, **boundedness**, **smoothness**, **segment invariance**, **global awareness**, and $\mathbf{O(1)}$ **per-frame cost**.
It is not merely a visual smoothing trick; it is a principled tracking controller for a sampled real-time transit display, built on top of an ML-predicted speed model trained on 9.7 million historical transit observations.