Lap-Time Simulator & Setup Optimiser

A quasi-steady-state F1 lap simulator, validated against real telemetry and combined withg a setup optimiser.

Mathias · Personal engineering project · July 2026

I built a quasi-steady-state (QSS) lap-time simulator in Python, validated it against real F1 telemetry to **within 1.25 % on lap time and 16 km/h RMS on the full speed trace**, then combined it with an optimiser that, given only a circuit's geometry it **independently chooses the right aero setup per track**: trimmed for Monza, loaded for Hungaroring.

1. The problem

Formula 1 teams run lap simulations constantly to choose setups, evaluate upgrades, and plan strategy. I wanted to build a genuine, scaled-down version of that workflow to see how accurate it could be. The project had three goals, staged so that each one works on its own:

- **Predict** a real F1 lap's speed trace from a physical car and track model.
- **Validate** that prediction against actual telemetry
- **Optimise** Have the tool *recommends* a setup rather than just reporting values.

The design is deliberately failure-resistant: a working, validated simulator is already good; however, I wanted the optimiser to allow it to adapt and refine itself.

2. The model

A **quasi-steady-state** simulation assumes the car is in equilibrium at every point on the track with no transient dynamics. The track is discretised into points, and at each one I ask: *"what is the fastest the car can go here without exceeding grip or power?"*

```
1. Track model     -> x,y racing line -> curvature k(s) at each point
2. Cornering limit -> v_max from grip + downforce vs curvature
3. Forward pass    -> accelerate out of each apex (power + traction + drag limited)
4. Backward pass   -> brake into each apex (braking-grip limited)
5. Combine         -> speed(s) = min(corner, accel, brake limits)
6. Integrate       -> lap time = sum ds / v(s)
```

The main component is the grip and power envelope: tyres cannot deliver maximum braking and maximum cornering at once, and that coupling is what makes this a real model rather than arithmetic.

Tech stack: Python (NumPy, SciPy, Matplotlib), **FastF1** for real telemetry, and `scipy.optimize` for both correlation and setup optimisation.

3. Build, stage by stage

Stage 0: Point-mass

Car as a point mass, synthetic track, forward and backward passes on a friction circle. Output: a working speed trace and lap time. I proved the solver logic on a hand-checkable single corner before touching real data.

Stage 1: Real track and validation

I used the **2024 Italian GP pole lap (Norris)** from FastF1, extracted the racing line, and built an auto-smoother that selects the spline smoothing on *geometric grounds alone* (an L-curve knee) and never tuned against the speed trace. Not being tuned against the speed trace is crucial, as adjusting geometry to hit a target speed hides important car-model errors.

```
Predicted lap : 80.33 s
Real lap      : 79.34 s   (+1.25 %)
Top speed     : 344.6 vs 347.0 km/h
```

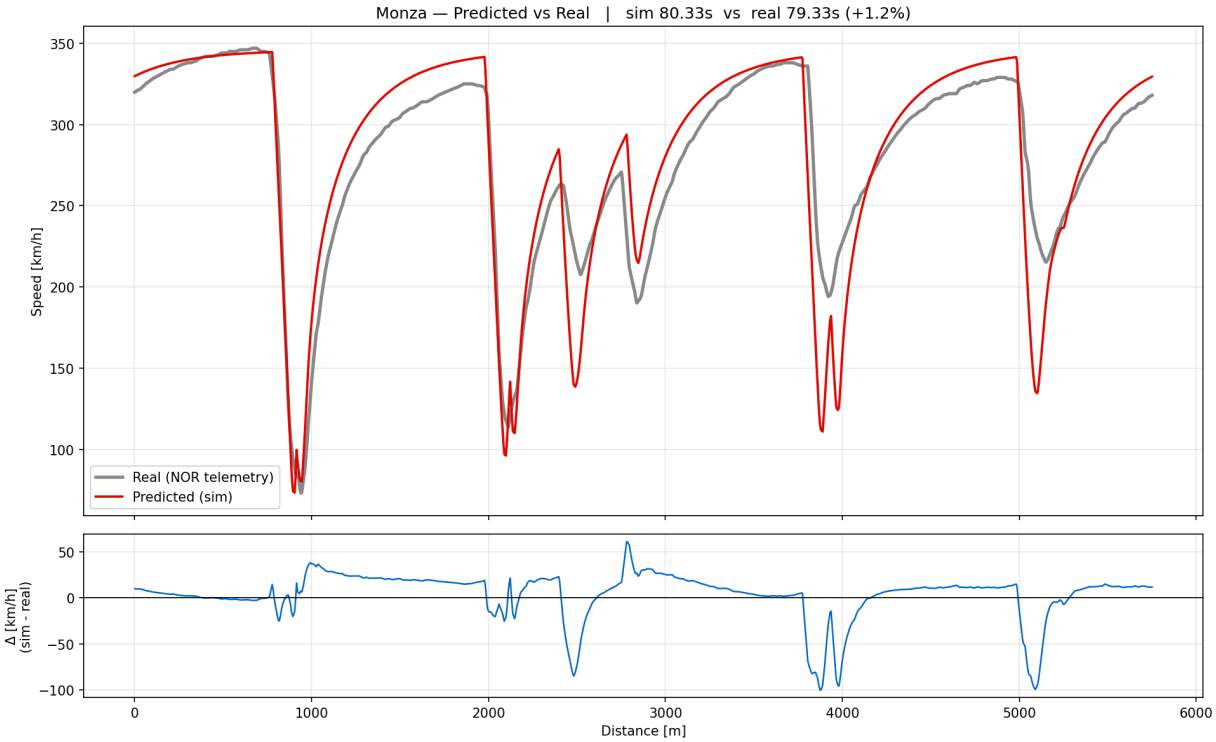

Figure 1. *Predicted (red) vs real telemetry (grey) speed trace at Monza, with the delta below.*

The insight that drove everything after: the +1.25 % lap time *flatters* the model. The honest measure is the **RMS trace error of 26.1 km/h**. The errors compensate too fast over 74 % of the lap (corner exits), too slow over 26 % (sharp apexes, worst −100 km/h at Ascari), so they nearly cancel over a lap. The next step is to validate the trace, not just the lap time. The root cause is the constant-μ point-mass assumption.

Stage 2: Load-sensitive tyre and correlation

I added a **load-sensitive tyre model** so the grip coefficient falls as vertical load rises:

$$\mu(F_z) = \mu_0 \left(1 - k_{LS} \left(\frac{F_z}{F_{z0}} - 1\right)\right)$$

A slow apex has little downforce, low load, and so more grip; a fast corner has high load and less grip. **One mechanism fixes both Stage 1 error directions at once.** I then built a parameter fitter (Nelder–Mead) that correlates the car parameters to telemetry which is exactly what teams call *correlation*, and the same machinery the Stage 3 optimiser later reuses.

```
RMS trace error : 26.1 -> 21.2 (spike-clip) -> 16.1 km/h (fitted)   -38%
Error balance   : compensating -> balanced (+11 / -12)
Fitted car      : mu0=2.15, k_LS=0.16, ClA=5.0, CdA=1.59
```

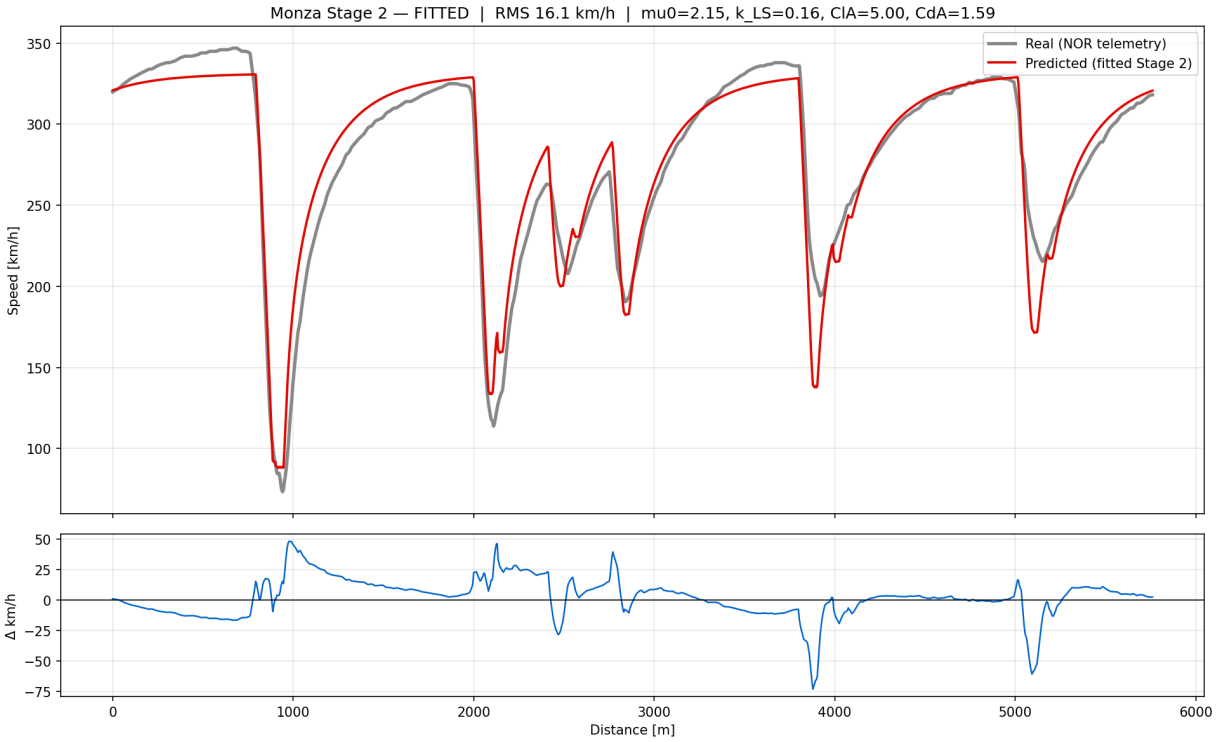

Figure 2. *The correlated fit after adding the load-sensitive tyre to the trace now follows the telemetry closely.*

Important Checks: I proved the residual 16 km/h floor is a *model-class* limit, not a data problem, **four independent ways** (smoothing, arc-length resampling, despike-and-refit, and an independent circle-fit of curvature). A QSS point mass maxes around 3.9 g in fast corners versus the roughly 5.4 g real cars pull; closing that gap needs transient dynamics, which is out of the current scope.

Stage 3: The Optimiser (Main Result)

Same engine as the Stage 2 fitter, **flipped**: instead of tuning car parameters to *match telemetry*, tune the **setup** to *minimise lap time*.

The physics that makes this a real trade-off: more wing means more downforce (faster corners) but more drag (slower straights), tied together by a drag polar $C_dA = 0.70 + 0.0356\,(C_lA)^2$. I optimise with `scipy.optimize.minimize_scalar` plus a full sweep for the trade-off curve.

	Stage 2 (correlation)	Stage 3 (optimisation)
Variable	car parameters	**wing level (ClA)**
Objective	min RMS vs telemetry	**min lap time**
Answer	"what *is* the car?"	"what setup is *fastest*?"

```
Monza       length 5760 m | OPTIMUM wing ClA = 3.29 | best lap 76.74 s
Hungaroring  length 4328 m | OPTIMUM wing ClA = 6.60 | best lap 72.27 s
=> the twisty track wants +3.31 more wing. Same car, different answer.
```

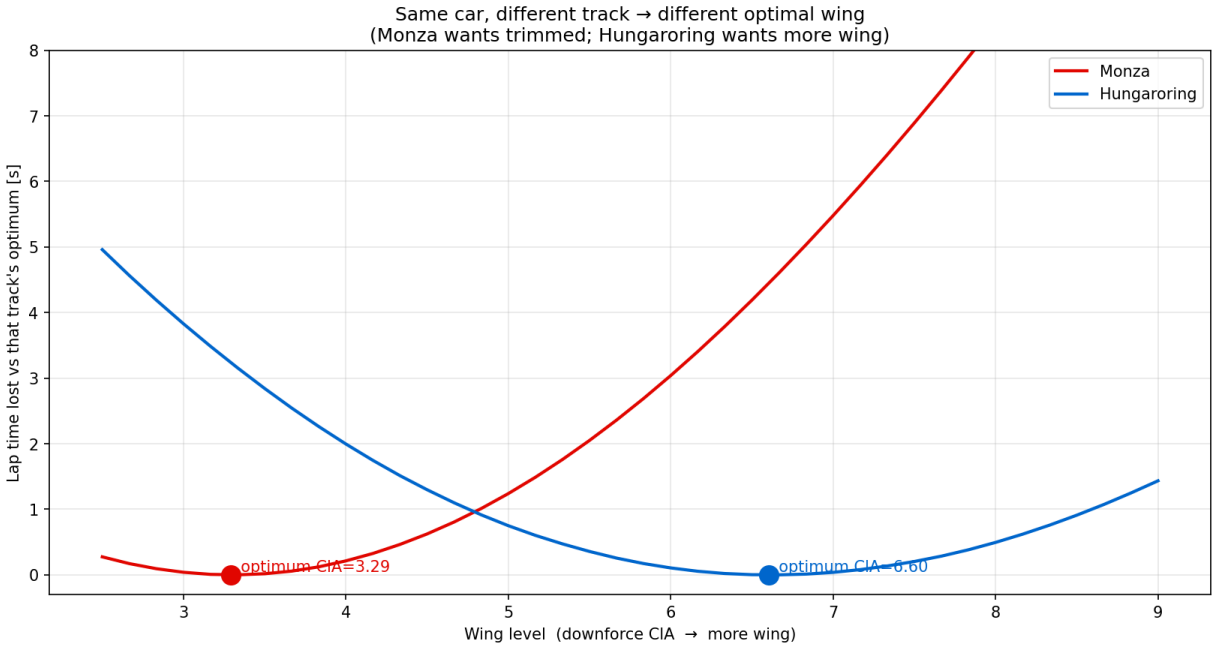

Figure 3. *Same car, two circuits, two different optimal wing levels—recovered from geometry alone.*

Why this result matters: Monza is the real-world *lowest*-downforce race of the season; Hungaroring one of the *highest*. The model rediscovers this from geometry alone. The two trade-off curves cross around $C_lA \approx 4.8$: Monza punishes excess wing (drag-limited), Hungaroring punishes too little (grip-limited). That is a tool making a correct *engineering recommendation*, not just a number.

Second design variable: Gearing. I added a real gearbox and torque-curve model and made final drive a second variable. Once the power curve rolls off past peak rpm (as real engines do), gearing becomes a genuine optimisation with a non-convex, sawtooth landscape, so I use a grid search. The Monza 2-D optimum is $C_lA \approx 3.17$ **(trimmed) with long gears (final drive** ≈ 0.73**), giving 77.95 s**, which matches real Monza philosophy on both axes. The key finding: wing and gearing couple, and low wing demands long gears.

4. Honest limitations

- **Absolute lap times are approximate off-Monza**. The car was correlated to Monza only for this current iteration. The robust output is the *relative* setup direction, which uses the same model on both tracks so the shortfall largely cancels.

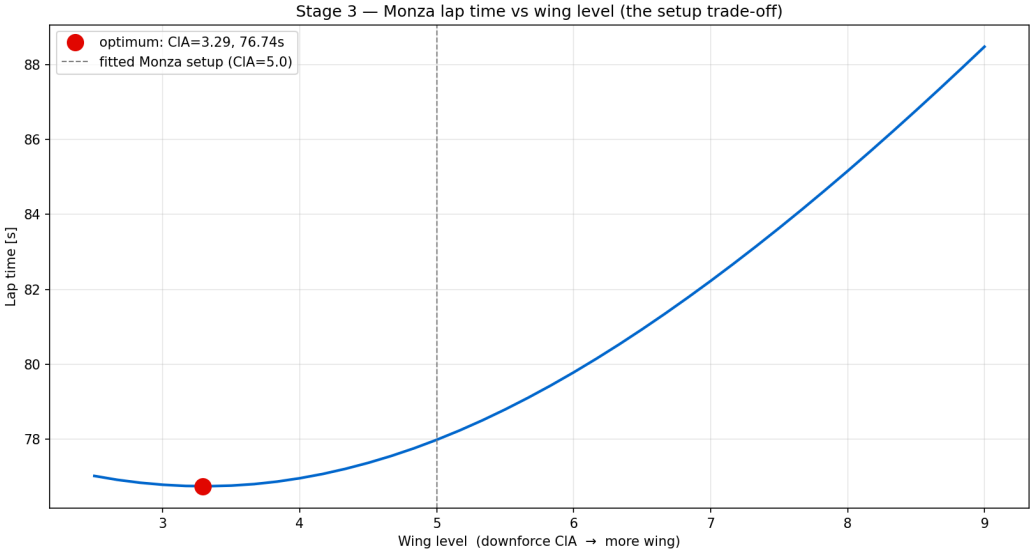

Figure 4. *The single-track Monza curve shows a genuine minimum, meaning the optimum is real, not edge-pinned.*

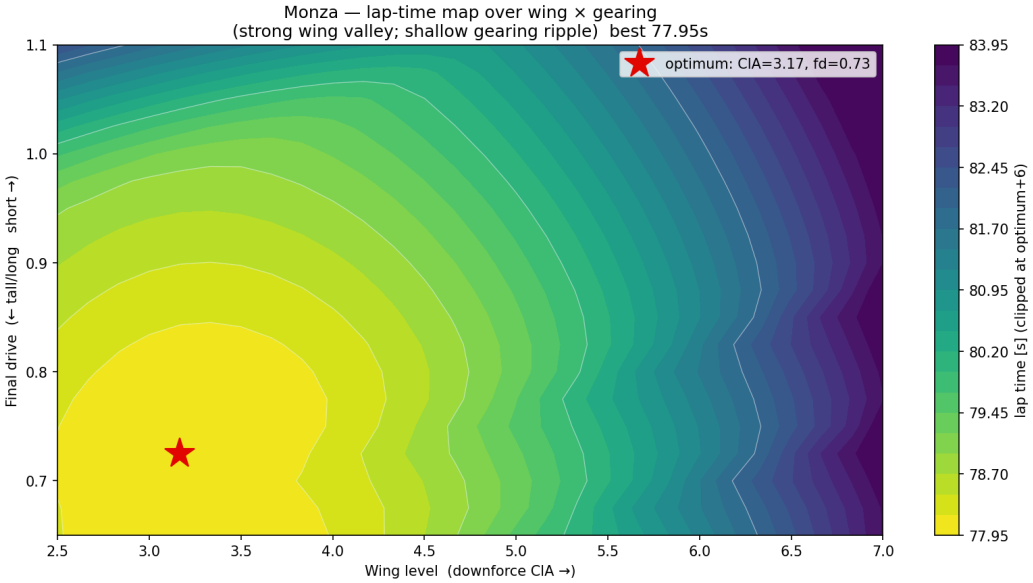

Figure 5. *The 2-D lap-time map over wing and final drive; the optimum position tends toward long gears at low wing.*

- **The drag polar constants** are a reasonable single-point calibration through the fitted Monza point, not independently measured but a stated assumption.
- **QSS ceiling:** no transient load transfer, no tyre slip model, linear aero. This is a modelling-altitude choice, not a bug, and it is evidenced.

5. Summary

I built a quasi-steady-state lap-time simulator, validated it against real telemetry to within 1.25 % on lap time and 16 km/h RMS on the full speed trace, then wrapped it in an optimiser that independently recommends a low downforce wing for Monza and a high downforce setup for Hungaroring. These match the common real-world setup directions, from track geometry alone.