

# WEC at Barcelona: estimating lap time for pole position

Andrea Patrucco, Vehicle Dynamics Engineer, Applus+ IDIADA Spain, 13/10/2026

*The views, thoughts and opinions expressed in this article solely those of the author, and they do not necessarily reflect the official policy or position of Applus+ IDIADA.*

## 1 Introduction

Due to the cancellation of Qatar and Bahrain races from the WEC calendar this year, the series is coming to Barcelona – Montmeló for the first time. This poses an interesting opportunity for whomever, like me, likes to play race engineer at times to answer the obvious question: how fast will current LM cars go there? Like, for example, what lap time could we expect from the qualifying session?

There are at least two simple ways to answer it, plus the extremely convoluted and not necessarily more accurate one I'm dedicating this article to.

### 1.1 Proportions

Thanks to the marvels of the public Italian school system, I learned at around the age of 12 that, if you assume the relation from two classes of values X and Y to remain constant, you can write:

$$x_1:y_1 = x_2:y_2$$

Now, even though we have no data for Barcelona/WEC, we do have:

- Data for Formula 1 cars at Barcelona
- A handful of tracks which appeared in both WEC 2025 and in the F1 2025.

These pieces of information allow us to populate the following table:

| Track     | F1 2025 | WEC 2025 | WEC 2026 | WEC/F1 | 26/25 |
|-----------|---------|----------|----------|--------|-------|
| Spa       | 100,562 | 119,617  | 120,653  | 1,189  | 1,009 |
| COTA      | 92,510  | 117,655  |          | 1,272  |       |
| Qatar     | 79,387  | 98,359   |          | 1,239  |       |
| Imola     | 74,670  | 88,920   | 90,127   | 1,191  | 1,014 |
| Bahrain   | 89,841  | 106,826  |          | 1,189  |       |
| Average   |         |          |          | 1,216  | 1,011 |
| Barcelona | 71,546  | 87,002   | 87,970   |        |       |
| Monza     | 78,792  | 95,814   | 96,879   |        |       |

Where we see that, on average, WEC pole position times in 2025 were 1.216 times higher than their F1 counterparts. We also see an increase in minimum lap times by 1% between 2025 and 2026 in WEC; chaining multiplications by these two factors to the real Formula 1 best qualifying times on Barcelona and Monza, we get 1'27.97 and 1'36.88, respectively.

### 1.2 Videogame Leaderboards

Sometimes, a decent representation of a car/track has already been created by somebody, for a variety of reasons: one of these are videogames. The WEC championship is featured in the Le Mans Ultimate game, published by Studio 397; notably, the game includes far more tracks than the official championship itself, among which we find both Monza and Barcelona.

In "TrackTitan" leaderboards we can access here [1], we can see that the current record for the Hypercar category stands at:

- Barcelona: 1'27.581 (User M.Imhof on Peugeot 9X8 Evo)

- Monza: 1'34.203 (User shae\_osullivan on Porsche 963)

We can see that predictions are reasonably close to each other (spoiler alert: they'll also be to the more convoluted ones we'll present later on).

### 1.3 The extremely convoluted and not necessarily more accurate way

Another way to answer the question than the two listed above is what I decided to dedicate some weeks of spare time, and this LinkedIn post. The basic approach was:

- Build a basic Hypercar model for a simplified laptime calculator
- Determine *somehow* the best racing line along Barcelona
- Run the former on the latter

There are commercial tools to do all this, like Danny Nowlan's ChassisSim but, in the spirit of having more fun, I decided as usual to just build everything from the ground up, to validate it, and to use it to generate my answer.

This went through:

- Defining a simple "GGV" model for racing cars with strong aerodynamic effects
- Validating the model by fitting its parameters on the Formula 1 2025 season data, which in itself required:
  - Finding a reliable way to extract trajectory information from available data
  - Validating two quite different ways to determine a vehicle's trajectory over a track
- Use the same model to describe a LM car
- Run the LM model on an optimized trajectory

## 2 The GGV Model

I already explained the basics of a GGV model in a previous article here [2]: to summarize, let's just say that we'll represent our vehicle as an entity which can generate longitudinal and lateral accelerations ( $A_x$ ,  $A_y$ ) as a function of speed alone.

The only information we are really interested to is the pareto-front of maximum lateral acceleration for each value of longitudinal acceleration or, seen in reverse, the maximum braking and positive acceleration we can achieve for a given amount of lateral acceleration.

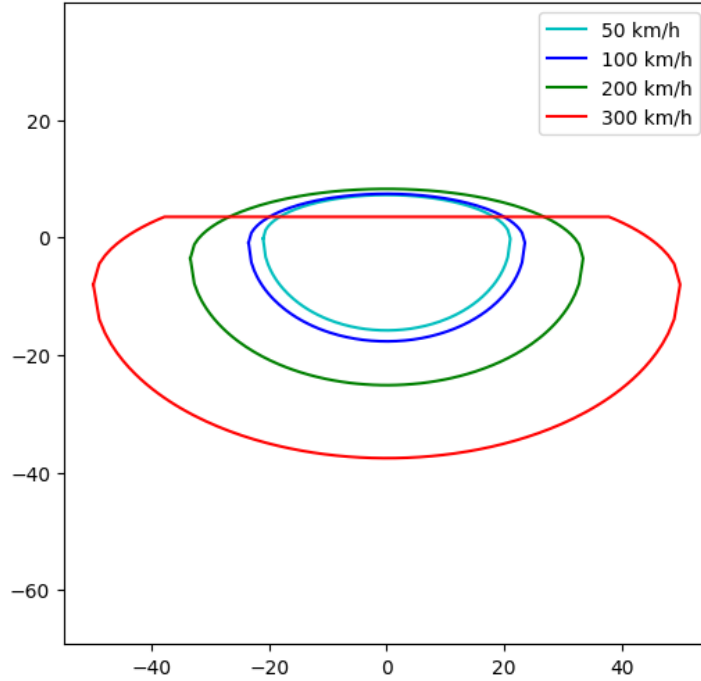
A particularly simple, yet effective, way to generate such a model for a racing car is to build it around the following parameters:

| Symbol   | Description                                                             | Unit  |
|----------|-------------------------------------------------------------------------|-------|
| P        | Engine power                                                            | kW    |
| m        | Vehicle mass                                                            | Kg    |
| $\mu_x$  | Friction coefficient – Longitudinal direction                           | -     |
| $\mu_y$  | Friction coefficient – lateral direction                                | -     |
| $C_{DS}$ | Drag coefficient (multiplied by front area)                             | $m^2$ |
| $C_{LS}$ | Lift coefficient (multiplied by front area) – positive <b>downwards</b> | $m^2$ |
| $\phi_S$ | Fraction of static vertical force acting on tractive wheels             | -     |
| $\phi_A$ | Fraction of aerodynamic lift force acting on tractive wheels            | -     |

Maximum longitudinal and lateral accelerations are given by the following formulae:

$$\begin{aligned}a_{y,max} &= \mu_y \left( g + \frac{1}{2m} \rho C_L S V^2 \right) \\a_{x,min} &= -\mu_x \left( g + \frac{1}{2m} \rho C_L S V^2 \right) - \frac{1}{2m} \rho C_D S V^2 \\a_{x,max} &= \min \left( \mu_x \left( \varphi_s g + \varphi_a \frac{1}{2m} \rho C_L S V^2 \right); \frac{1}{m} \left( \frac{P}{V} - \frac{1}{2} \rho C_D S V^2 \right) \right)\end{aligned}$$

I will not go much into detail about how the GGV *shape* is obtained starting from maximum longitudinal and acceleration values; a typical shape such a model produces can be seen in the following image:



The model is based on some debatable assumptions, like neglecting load transfer effects, assuming the track is perfectly flat and potentially other issues. Is this model still able to represent the effects of changes in its physical parameters? Does it correlate well with real data?

I think a good way – in general – to tell is to “reverse” the question: *if I correlate well the model with real data, do the physical parameters change in a meaningful way?*

Which led me to the second, and perhaps trickiest part of this research: processing the real data, and finding a “good” way to correlate a model on it.

## 3 Extracting Real Data

### 3.1 Selecting Data

A nice use-case to see whether the above model can represent the real behavior of an aero car is to use qualifying lap times for the Formula 1 2025 season. This is a good choice because:

- There is plenty of publicly available data on it – on the TracingInsights repository [3]
- During qualifying, we can use two useful assumptions to simplify the correlation process:
  - The vehicle is accelerating using its maximum available power (768kW)
  - The vehicle mass is very close to the minimum allowed (800kg)

Through this, we can impose  $P_{max}$  and mass as constants, and reduce the complexity of the correlation process.

- Vehicles are “similar” (we will be using different cars for each track, but still, they are all same-category cars using the same tyre supplier), but their set-up (namely, aerodynamic) is supposed to greatly change between tracks. This implies that, if we treat each pole position lap as a separate correlation problem based on the same parameter set, we would expect some parameters to change and some others to remain roughly constant.

Why Formula 1 and not WEC directly? Because, to my knowledge, there is no comparable open repository to that of TracingInsights for WEC, for which only lap times are publicly available.

Why the 2025 season and not the 2026 one? Two main reasons:

- Data for some pole-position times for the 2026 season seem to be defective – the GPS signal seems to disappear, sampling is not correctly cut; one could easily work around this by using the second, or third best, but 2025 seems cleaner (and more complete, since the 2026 season is still ongoing), in general.
- 2026 cars, with the addition of the “power budget” part of the rulebook, partly violate the assumption of accelerating with all available power: the expected correlation from the model is lower.

So, once downloaded all pole-position telemetry .csvs available in TracingInsight, we are presented with the first real problem posed by the project: how to extract curvature data.

### 3.2 Curvature

As I explained in [2], the iterative algorithm needs curvature and distance signals to calculate, for a given GGV model, speed at all points of a trajectory. Under ideal circumstances (the trajectory is described by a continuous function in the XY plane), curvature can be extracted from a collection of points as an “instantaneous” value.

Even though they are relatively clean, data from the TracingInsights repository are not regular enough, in positioning, to allow for such a naive calculation, and instead, they require some intermediate step for a usable signal to be extracted.

Given an input planar trajectory composed by a sequence of (x, y) points, the most immediate way one can calculate curvature associated to each point is:

- Transform points into segments, each defined as the line connecting each point to the following one.
- Determine the difference in orientation between segments
- Divide the orientation difference by *some* distance between consecutive segments. This may be the distance between mean points of each segment, or the length of either of the two segments: in practice, results are not intrinsically “better” for any of these methods.

Unfortunately, this methodology is extremely vulnerable to inaccuracies in input data, since both distance and orientation are strongly affected by even relatively small disturbances in sampled data points. To try and address this issue, I tested multiple strategies:

- **Smoothing:** The most instinctive countermeasure to this jittery signal is numerical convolution: take each point, and average it with its neighbors. While this certainly works (output curvature does become much cleaner), it comes at the cost of the constant doubt of whether it is degrading the “real” information, especially in tight corners. In some of the slower corners (some in Monaco, but even the initial chicane at Monza), curvature is supposed to steeply go up until an apex, and then decrease just as fast. Smoothing the signal here will inevitably result into a lower peak curvature value, with obvious negative impacts on correlation: if the algorithm “sees” a lower-than-real curvature, it will

underestimate the vehicle's ability to generate cornering forces in order for it to match the comparatively low, real speed.

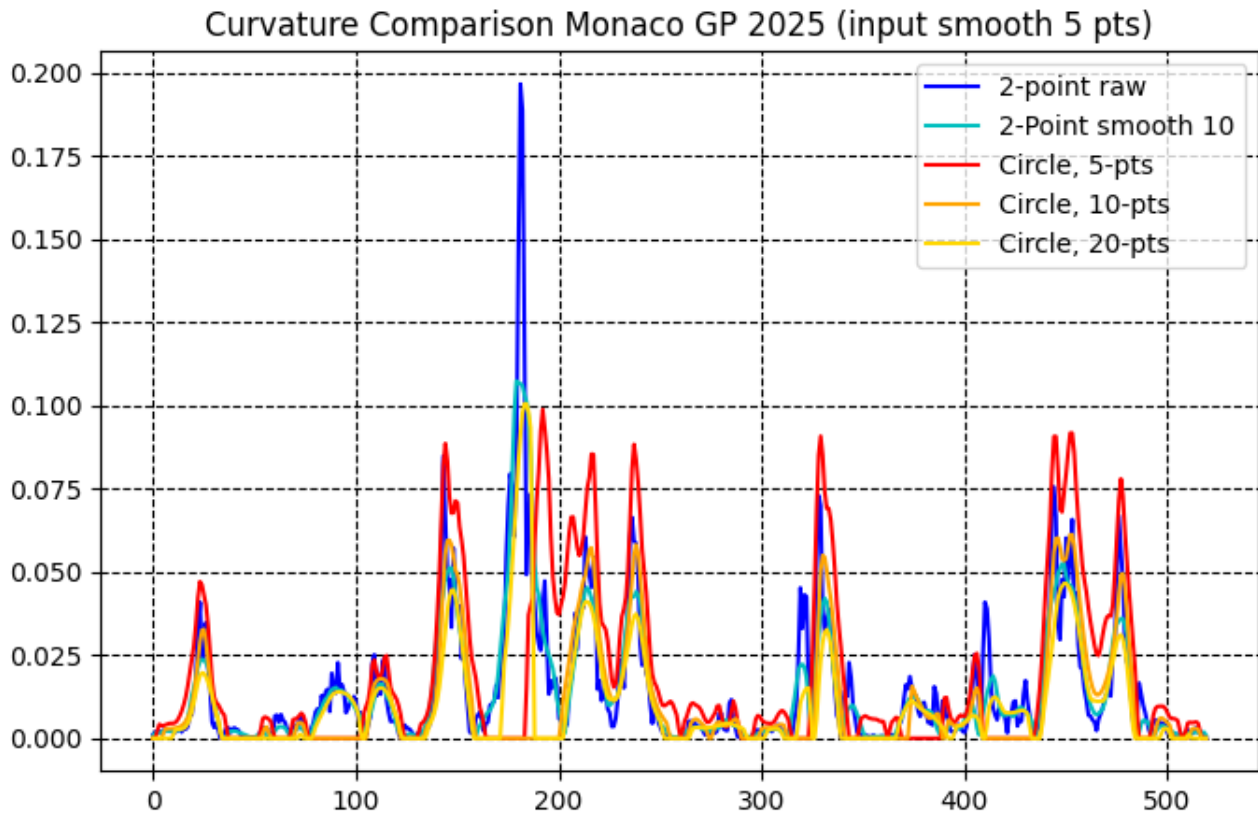
- **Butterworth Filtering:** Butterworth often seems to produce a similar result to smoothing, but with better ability to preserve peaks. In our case, it also “feels more correct”: it’s difficult to justify why the measured curvature in one point should become more precise by mixing it with the value in neighboring points; it’s much more instinctive to say that a F1 car won’t change direction in lower-than-X-meters wavelengths, which is what a low-pass filter implies. Still, applying the technique to space-signals on a constant-dt sampling is questionable, and, in practice, the typical “overshoots” of a low-pass filtered signal are well visible, for example in Monza: incapable of handling the fast change of sign in curvature during the first chicane, the filtering introduces some oscillations before and after the chicane.

- **Polynomial Fitting:** here, the idea is to try and address disturbances *before* calculating curvature: instead of calculating it, for each point, with two segments, use N points before and after the current to fit trends in the X and Y coordinate. Then, replace the raw data points with interpolated ones and extract curvature.

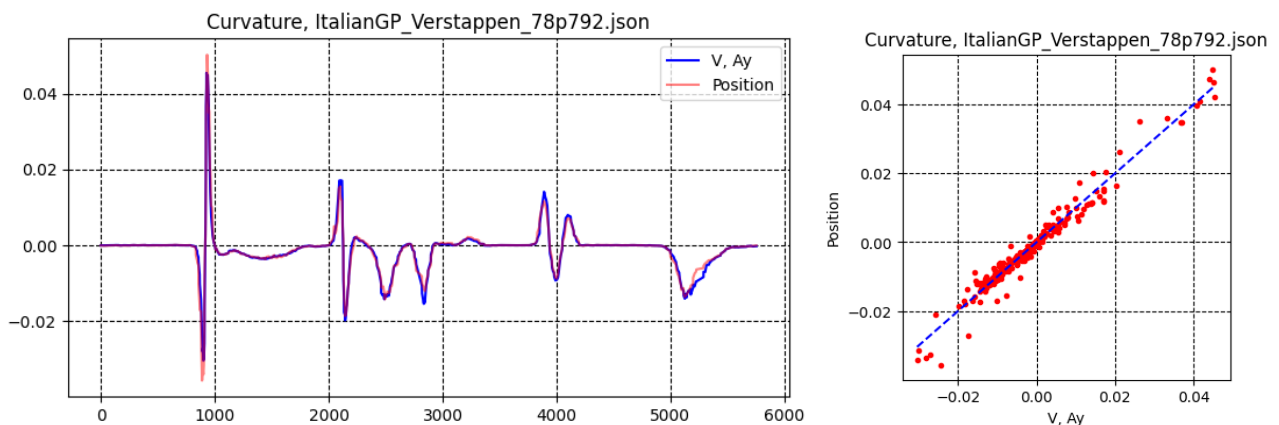
I guess this can be called a pretty crude spline regularization: again, it does impact how “continuous” the output signal looks, but, in practice, it does not seem to offer any real advantage compared to smoothing.

- **Circle fitting:** in this case, the curvature estimation ditches the “two segments” approach completely, to replace it with estimation based on fitting a circumference equation. The approach is based on, for each trajectory point:
  - Collect N neighboring points
  - For each of them, try and find a hypothetical rotation center, found numerically as the point which minimizes the difference between “radii” (distance between rotation center and each of the point”.

The procedure is numerically very inefficient, and it only becomes stable for a rather large number of neighboring points. Which again, defeats the purpose of “cleaning without losing detail in fast-changing zones”, since the output information is descriptive of a large chunk of trajectory.



In the above figure, we can see a comparison between raw curvature calculated with the 2-segment method, its smoothed variant, and various attempts with the “circle” strategy: We can see that fitting becomes more and more regular as the number of points used increases, but this directly affects the ability of the fitting routine to “follow peaks”, which become even more smoothed than with conventional convolution.



An useful way to validate whatever curvature extraction strategy is to compare the output curvature with the one calculated with the Ay, V signals. In fact, since TracingInsights data comes with a full history of speed and lateral accelerations, we can directly calculate curvature as  $Ay/V$ , instead of going through all the positional data processing.

In the plot above, you can see the “Ay/V” calculation outcome in blue, and, superimposed, the “10 point smooth” method outcome in red. While some systematic underestimation of curvature is visible, the two estimations are quite consistent.

In the end, I decided to just use smoothing and accept some degree of distortion. An interesting alternative would have been to directly use  $Ay/V$  curvature; however, this implies that, when

comparing a telemetry-based trajectory with one built with a completely different method (for example, geometrically), curvatures are computed in two very different ways, making a direct comparison questionable.

## 4 Building Models

### 4.1 Fitting the F1-2025 Model

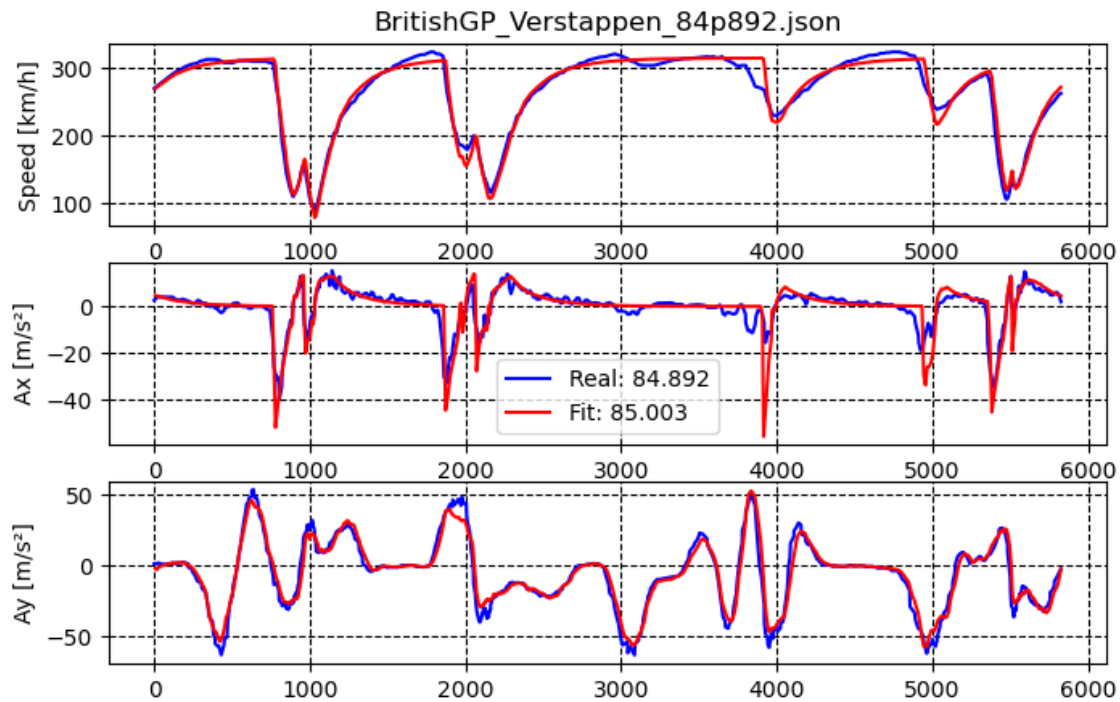
Now that we have a more-or-less stable way to extract curvature data from TracingInsights logs, we can use that to iterate over the physical parameters of the “simplified aero” model to find suitable parameters describing the F1 2025 car. As previously mentioned, we’ll do this separately for each track and pole-position time.

For each circuit, we will impose a genetic optimization problem where the model parameters are constrained between the following bounds:

| Parameter   | Unit  | Min | Max | Bits         |
|-------------|-------|-----|-----|--------------|
| P           | kW    | 768 | 768 | 0 (Constant) |
| m           | kg    | 800 | 800 | 0 (Constant) |
| $C_{DS}$    | $m^2$ | 0.3 | 5   | 16           |
| $C_{LS}$    | $m^2$ | 0.3 | 10  | 16           |
| $\varphi_S$ | -     | 0.1 | 0.9 | 8            |
| $\varphi_A$ | -     | 0.1 | 0.9 | 8            |
| $\mu_x$     | -     | 1.3 | 2.0 | 8            |
| $\mu_y$     | -     | 1.3 | 2.0 | 8            |

The optimization runs with 100 individuals for 40 generations; each track is independently re-fitted three times to obtain a rough idea of the parameter variability (repeatability of the fitted coefficients). Each complete optimization run takes around 20 minutes, running in Python on my local server, the three runs being launched in parallel.

Each optimization results into an optimized time history similar to the following one:

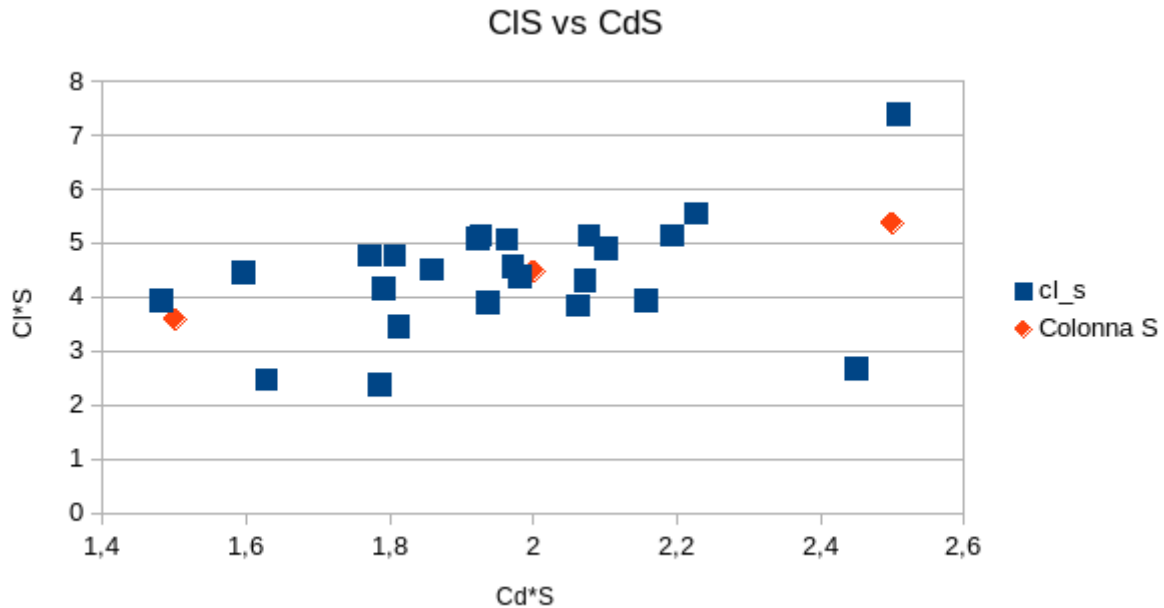


Where we can still see that there's a small yet noticeable effect of the curvature underestimation on lateral acceleration: longitudinal speed is better correlated than that, suggesting that the output parameters will somehow underestimate the real car's ability to generate lateral force.

| filename                              | max_kw | mu_x  | mu_y  | cd_s  | cl_s  | m   | tfrac_s | tfrac_cl |
|---------------------------------------|--------|-------|-------|-------|-------|-----|---------|----------|
| ChineseGP_Piastrri_90p641.json        | 768    | 1,428 | 1,844 | 1,971 | 4,577 | 800 | 0,882   | 0,604    |
| EmiliaRomagnaGP_Piastrri_74p670.json  | 768    | 1,318 | 1,920 | 2,100 | 4,896 | 800 | 0,897   | 0,383    |
| AbuDhabiGP_Verstappen_82p207.json     | 768    | 1,300 | 1,999 | 1,927 | 5,152 | 800 | 0,900   | 0,463    |
| AustralianGP_Norris_75p096.json       | 768    | 1,300 | 2,000 | 1,936 | 3,914 | 800 | 0,900   | 0,900    |
| BelgianGP_Norris_100p562.json         | 768    | 1,504 | 1,941 | 1,859 | 4,510 | 800 | 0,832   | 0,273    |
| QatarGP_Piastrri_79p387.json          | 768    | 1,300 | 1,992 | 2,226 | 5,553 | 800 | 0,899   | 0,897    |
| SaudiArabianGP_Verstappen_87p294.json | 768    | 2,000 | 2,000 | 1,812 | 3,451 | 800 | 0,700   | 0,101    |
| BritishGP_Verstappen_84p892.json      | 768    | 1,300 | 1,823 | 1,923 | 5,096 | 800 | 0,832   | 0,642    |
| SingaporeGP_Russell_89p158.json       | 768    | 1,321 | 1,894 | 1,982 | 4,378 | 800 | 0,870   | 0,509    |
| AustrianGP_Norris_63p971.json         | 768    | 1,300 | 1,910 | 1,963 | 5,067 | 800 | 0,743   | 0,825    |
| AzerbaijanGP_Verstappen_101p117.json  | 768    | 1,345 | 1,908 | 1,806 | 4,774 | 800 | 0,857   | 0,369    |
| MonacoGP_Norris_69p954.json           | 768    | 1,543 | 1,978 | 2,451 | 2,669 | 800 | 0,900   | 0,900    |
| BahrainGP_Piastrri_89p841.json        | 768    | 1,476 | 1,963 | 2,063 | 3,851 | 800 | 0,891   | 0,140    |
| SpanishGP_Piastrri_71p546.json        | 768    | 1,354 | 1,956 | 2,077 | 5,150 | 800 | 0,699   | 0,434    |
| LasVegasGP_Norris_107p934.json        | 768    | 1,300 | 1,518 | 1,629 | 2,478 | 800 | 0,502   | 0,883    |
| DutchGP_Piastrri_68p662.json          | 768    | 1,628 | 1,984 | 2,508 | 7,380 | 800 | 0,659   | 0,151    |
| JapaneseGP_Verstappen_86p983.json     | 768    | 1,405 | 1,952 | 2,193 | 5,151 | 800 | 0,756   | 0,526    |
| SãoPauloGP_Norris_69p511.json         | 768    | 1,300 | 1,999 | 1,791 | 4,159 | 800 | 0,780   | 0,530    |
| CanadianGP_Russell_70p899.json        | 768    | 1,879 | 1,993 | 1,786 | 2,390 | 800 | 0,837   | 0,100    |
| MiamiGP_Verstappen_86p204.json        | 768    | 1,622 | 1,893 | 1,772 | 4,773 | 800 | 0,795   | 0,100    |
| MexicoCityGP_Norris_75p586.json       | 768    | 1,346 | 1,799 | 1,481 | 3,933 | 800 | 0,779   | 0,590    |
| UnitedStatesGP_Verstappen_92p510.json | 768    | 1,424 | 1,892 | 2,071 | 4,303 | 800 | 0,857   | 0,404    |
| HungarianGP_Norris_74p890.json        | 768    | 1,532 | 2,000 | 2,156 | 3,952 | 800 | 0,900   | 0,336    |
| ItalianGP_Verstappen_78p792.json      | 768    | 1,549 | 1,748 | 1,597 | 4,457 | 800 | 0,560   | 0,455    |

We obtain the coefficients summed up in the above table. Coefficient estimation seems to be relatively stable (i.e., re-fitting multiple times on the same track leads to very similar parameters), suggesting that the amount of degrees of freedom in the model is *not* excessive and that there should be no collinear coefficients. However, the very high variability of both  $\mu$ s and the  $\phi$  variables suggests that, for whatever reason (curvature approximations, overall model simplicity), the process cannot be seen as a reliable method to extract track-specific variables. Over the course of the past weeks I made multiple attempts with a reduced set of variables and with different smoothing strategy with curvature: there indeed are ways to improve correlation between  $C_{DS}$  and  $C_{LS}$  (at the cost of a slightly reduced

correlation with the speed signal), however, it would be hard to sell that the original strategy of “extracting parameters automatically” works.



Correlation between  $C_{DS}$  and  $C_{LS}$  is quite weak at around 0.4. There *appears* to exist a correlation between the two, however some specific cases, for example Monaco, are located completely outside of the main trend. A linear regression applied to  $C_{LS}$  vs  $C_{DS}$  gives:

$$C_{LS} \sim 0.93 + 1.78 * C_{DS}$$

As for the plausibility of these values, the lower end of the  $C_{DS}$  spectrum seems believable if we assume that, in an open-wheeled racing car,  $C_D$  alone may reach values of 1.0 ([4]); the “aerodynamic efficiency” seems quite low, though: it might be due to optimistic assumptions about static friction, to the aforementioned underestimation of curvature, or both.

## 4.2 Synthesizing the F1 2025 model

Even though the model parameter extraction can hardly be called a success, we can still try and use the repeated fitting attempts to build a model for the F1 2025 car, with the following, combined strategy:

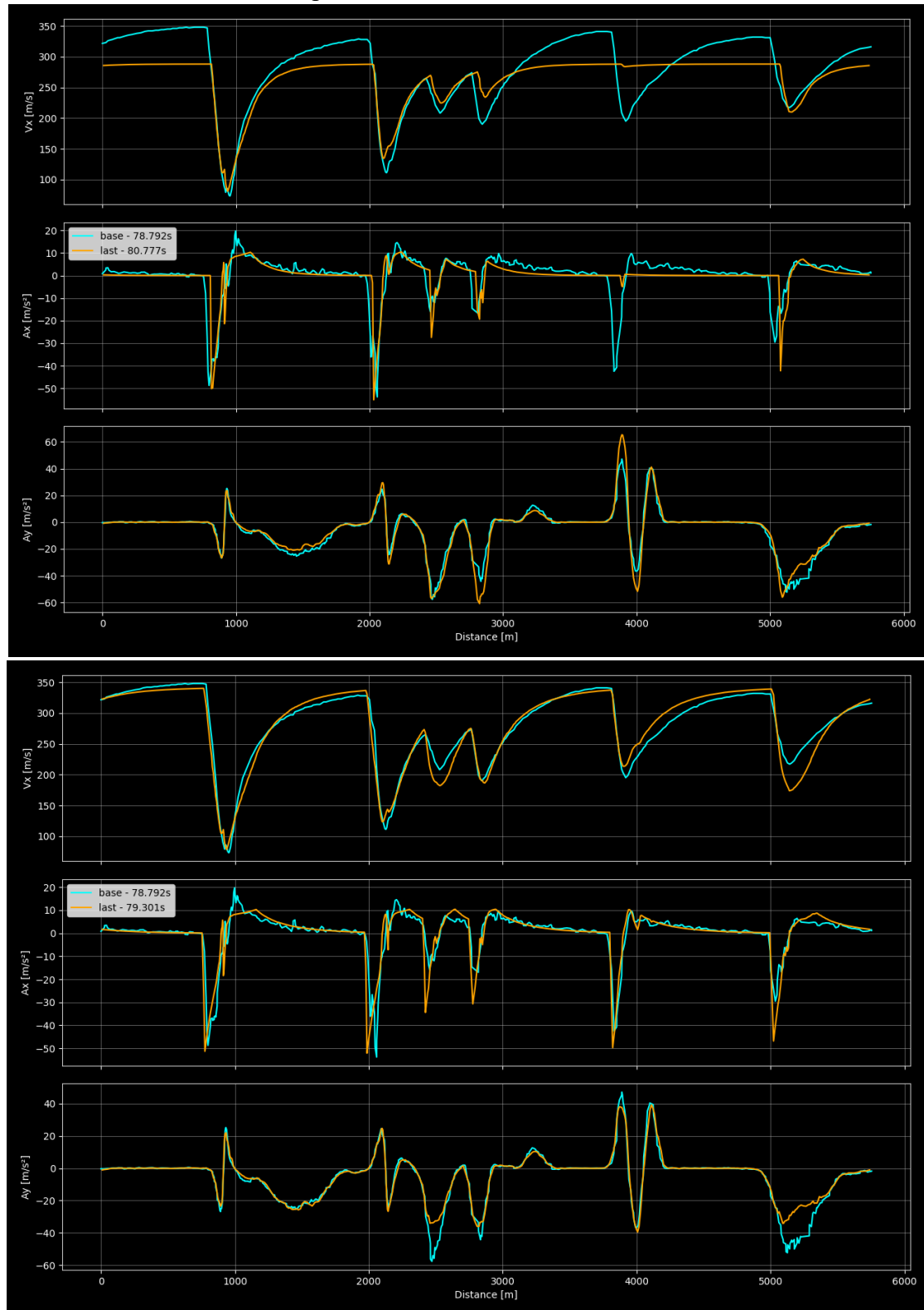
- $C_{DS}$  and  $C_{LS}$  will follow the linear law extracted from fitting results
- All other coefficients will be set to average values ( $\mu_x = 1.45$ ,  $\mu_y = 1.91$ )

A notable exception is constituted by  $\phi_S$  and  $\phi_A$ , which I am setting to 0.54 and 0.9, respectively. The reason is, 0.54 *should* better represent the static weight distribution on real Formula 1 vehicles; an attempt to repeat the process described in the previous point by only letting the algorithm fit  $C_{DS}$ ,  $C_{LS}$  and  $\phi_A$  results into values of 0.9 most of the times.

This 0.9 value is quite suspicious, as it implies a rather unbalanced aerodynamic load towards the rear axle. [5] suggests that load should be evenly balanced between axles, and such a high distribution, apart from being likely unachievable, would cause dramatic changes in the vehicle’s balance at different speeds. However, impacts of this parameter in our model is just limited to the vehicle’s ability to generate forward acceleration, with no further impact on the cornering phase: for this reason, I am leaving it as-is since it tends to provide better correlation with data.

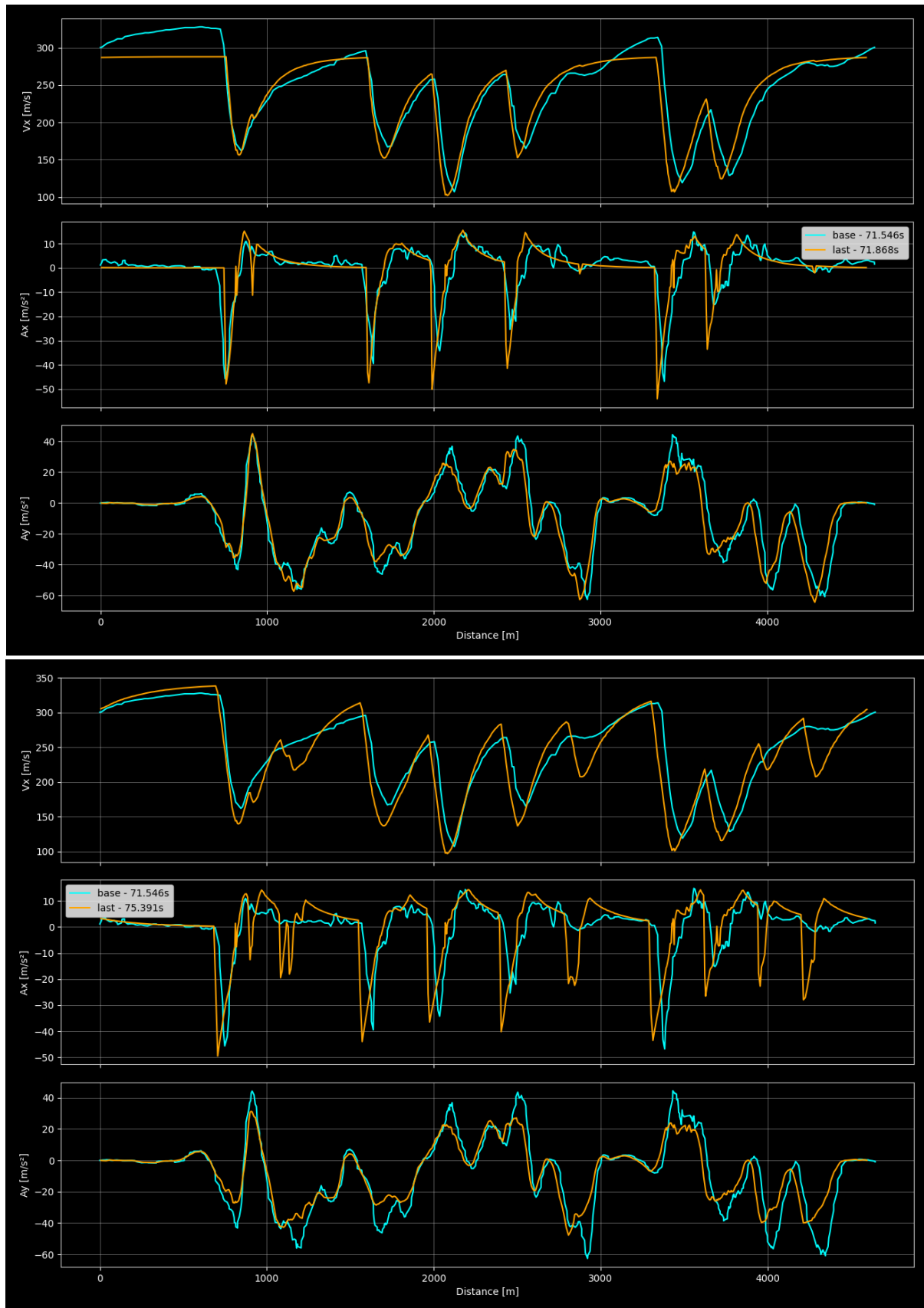
## Patrucco, WEC at Barcelona: estimating lap time for pole position, 2026

The output model further degrades correlation with respect to input data for the sake of simplicity: with a same set of coefficients, we can now play with the model around multiple F1 tracks to see the effects of different downforce set-ups.



For example, here we can see effects of downforce at Monza: the fastest set-up is clearly the low-downforce one below (1'19"3 vs 1'20"8), however I find it fascinating that it would be *technically* possible to fly through the Ascari variant at maximum speed with the high downforce setting.

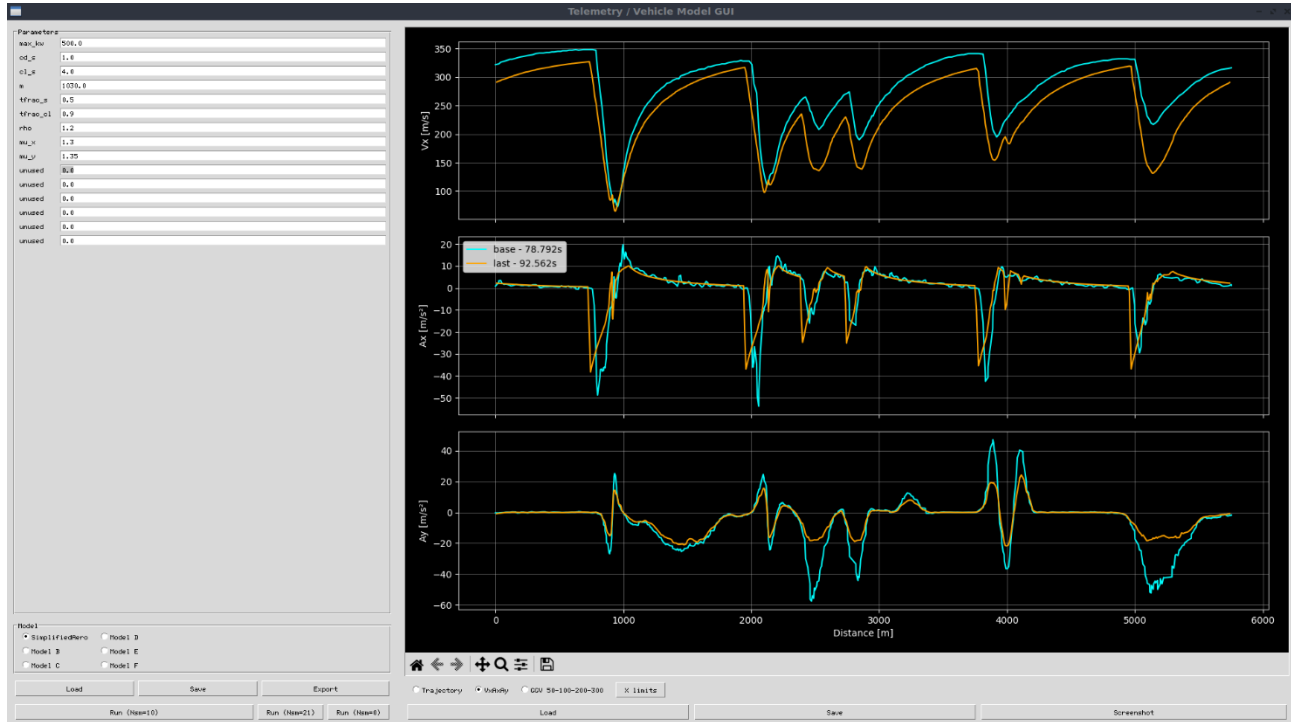
## Patrucco, WEC at Barcelona: estimating lap time for pole position, 2026



At Montmelo correlation is notably lower than in other cases. Still, even accepting the clear under-estimation of maximum speed obtained with the higher-downforce model (upper plot), this configuration is 3.5 seconds faster than the low-downforce one (lower plot).

### 4.3 Inventing the Hypercar Model

As previously mentioned, there doesn't appear to exist a public repository of telemetry data, for the WEC championship, as TracingInsight for Formula1. I hence could not attempt the same approach I applied to F1 (not that it really worked), and I had to do it with large combinations of guessing and trial and error.



To make my guessing and trial and error more tolerable, I had ChatGPT build a GUI around my model, GG-iterator and trajectory parsing routines, which you can see in the figure above. By using TracingInsight, F1 trajectories as the initial guess and modifying model parameters observing lap times, I settled on this final selection of parameters:

| Parameter   | Unit  | F1 2025               | LM                 | Notes                                       |
|-------------|-------|-----------------------|--------------------|---------------------------------------------|
| P           | kW    | 768                   | 500                | By rulebook                                 |
| m           | kg    | 800                   | 1030               | Minimum value w/o BoP penalties             |
| $C_{DS}$    | $m^2$ | 1.5 – 2.5             | 1.0 – 1.3          | From rulebook                               |
| $C_{LS}$    | $m^2$ | $0.9 + 1.8 \cdot C_d$ | $4.0 \cdot C_{DS}$ | From rulebook                               |
| $\varphi_S$ | -     | 0.54                  | 0.54               | <i>Wrong</i> : it should be closer to 0.515 |
| $\varphi_A$ | -     | 0.9                   | 0.9                | From manual optimization                    |
| $\mu_x$     | -     | 1.45                  | 1.3                | From manual optimization                    |
| $\mu_y$     | -     | 1.91                  | 1.35               | From manual optimization                    |

For aerodynamics, LMH and LMDh cars seem to be constrained to quite rigid rules. I could not access the Annex of the rulebook containing its definition; for some reason, however, whatever AI system I asked was consistently telling me that:

- $C_{DS}$  must be equal or greater than 1.0
- $C_{LS}$  shall be equal or lower than 5.2
- The ratio  $C_L / C_D$  should never exceed 4

Based on this, I am assuming that whatever set-up in LM cars fully uses the maximum allowed aerodynamic efficiency, and that  $C_D S$  should be free to move between 1.0 and 1.3, resulting into  $C_L S$  values between 4 and 5.2.

The higher aerodynamic efficiency of LM cars is, however, more than compensated by the higher power, lower mass and higher friction set for the F1, which, in practice, is around 20% faster.

## 5 Trajectories, again

I find the trajectory problem interesting because there doesn't seem a clear consensus about their importance in the lap time optimization problem, at least for static, simplified iterators based on a GG model such as the one I'm using here. People I talked to have a wide-ranging array of opinions about this, from "it's an *academic* problem with limited real world importance" and "it's very important: even a minor set-up setting may cause the ideal line to shift considerably".

These two extreme positions are well represented by the two approaches I tried to follow to find suitable trajectory data for the remainder of this article:

- Geometric optimization
- Heuristic, model-dependent optimization

Independently on the approach, we always define a "track" as two margin lines (left, right), which I usually sacrificed some sleep hours to manually draw from Google Earth. We then discretize margins into two collections of  $N$  points, and we draw  $N$  segments going from the left to the right margin.

Now, we can describe all *valid* trajectories included in track limits as a  $N$ -long array of values bounded between, say, -1 (left margin) and +1 (right margin), and we can transform this array into a trajectory and evaluate with it some function we want to minimize.

The algorithm strategy only affects the choice of the minimized function, and the optimizer.

### 5.1 Geometric Trajectory Definition

With this approach, we are assuming that the *ideal trajectory* exists independently on the vehicle we are using to follow it. In terms of fitness function, there are two obvious, common-sense choices we could follow:

- Minimize total distance
- Minimize total curvature

It's immediately clear that the second has better chances at producing a "good" trajectory than the first one; however my attempts seem to suggest that if you *first* minimize distance, and *then* use the output of the distance minimization as the initial solution for curvature minimization, you save some hundreds iterations, enjoying a reduction in overall computational times, since the optimization of the former is already closer to the final result than other trivial initial guesses (for example, "always stay in the center" = initial array of zeros).

Distance minimization is rather trivial, since we can express total distance as:

$$D = \sum_{i=1}^{N-1} d_i = \sum_{i=1}^{N-1} \sqrt{((x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2)}$$

Where  $x_i$  and  $y_i$  represent the points composing a trajectory, identified by the free coordinates  $\varphi_i$  as in:

$$\begin{aligned}x_i &= \frac{1}{2}(x_{Li} + x_{Ri}) + \frac{1}{2}\varphi_i(x_{Li} - x_{Ri}) \\y_i &= \frac{1}{2}(y_{Li} + y_{Ri}) + \frac{1}{2}\varphi_i(y_{Li} - y_{Ri})\end{aligned}$$

We can hence compute the distance array as a function of the free variable, including analytical derivatives, and obtain a N-1-long fitness array plus a N-1 x N jacobian matrix, which we can use to minimize the sum of components with Gauss-Newton (if we impose the trajectory to be closed, the solution is N-long and the jacobian is square).

Total curvature minimization is only slightly trickier. Instead of the distance between two consecutive points, we must use the “two-segment method”, which implies relying onto point triplets. This slightly complicates the procedure since we’ll have to assemble an N-2 x N Jacobian matrix where each solution’s index derivative (row) affects three columns; still, derivatives are simple enough to allow hand calculation. Assuming the same nomenclature we used before, for a triplet of points, we can calculate curvature as:

$$\gamma_i = \frac{\sqrt{2}}{C_i} * \sqrt{1 - k_i}$$

$$C_i = \sqrt{((x_{i+2} - x_{i+1})^2 + (y_{i+2} - y_{i+1})^2)}$$

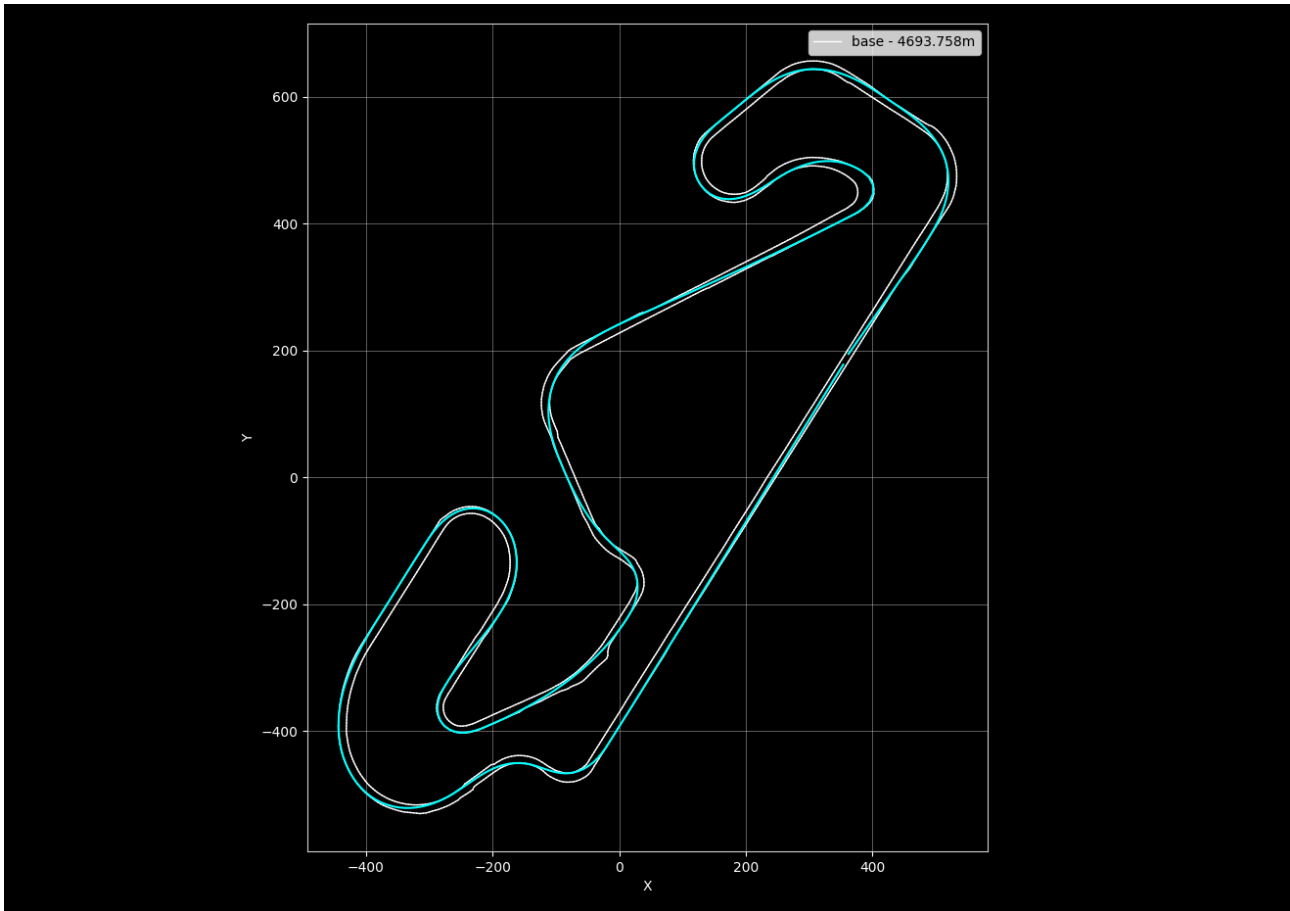
$$k_i = \frac{(x_{i+1} - x_i)(x_{i+2} - x_{i+1}) + (y_{i+1} - y_i)(y_{i+2} - y_{i+1})}{\sqrt{((x_{i+2} - x_{i+1})^2 + (y_{i+2} - y_{i+1})^2)} * \sqrt{((x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2)}$$

The above has been obtained by defining curvature as the ratio between:

- The angle between two consecutive segments, calculated as the arc-cosine of their scalar product
- The length of the *second segment* (again, this is somewhat arbitrary: either of the first segment, second segment, or line connecting first and third point might be a good choice; apart from the obvious formulation differences, results do not seem to change much).

Since the Gauss-Newton optimizer is only apt to optimize unbounded problems, it would for sure exceed the (-1, 1) constraint we are imposing to the solution: a good way to avoid this is to just pre-transform the array of unknowns with a hyperbolic tangent: this way, points in the solution array are free to take any value between  $(-\infty, +\infty)$  but the output trajectory always remains valid, and the solution and its jacobian always remain continuous, something that numerical optimization methods always seem to like a lot.

We can apply this method to the Montmelo circuit, obtaining this:

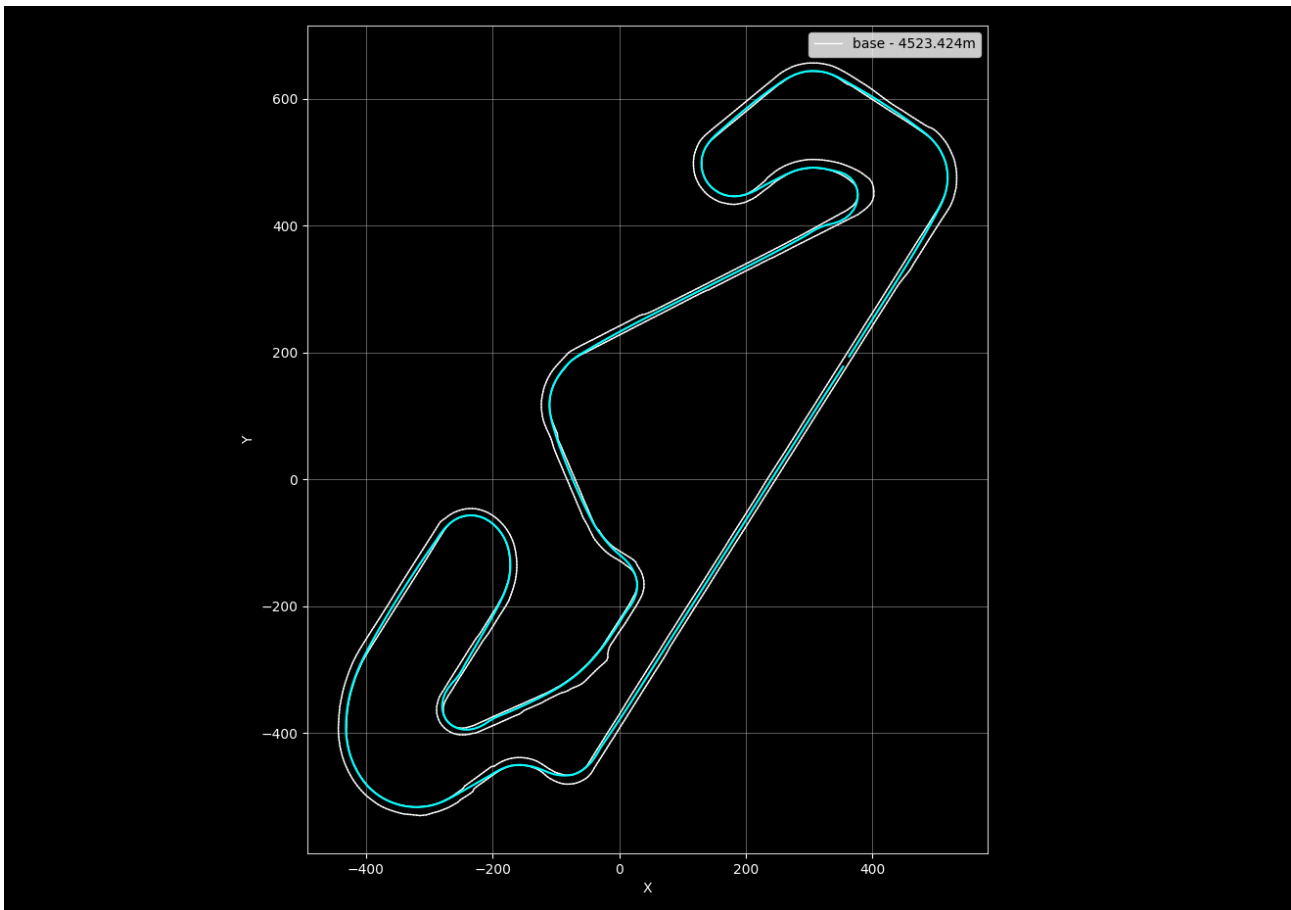


While this may be *technically correct*, i.e., it might be a believable minimum-overall-curvature line, it's clear it is far from ideal as a racing line. Starting from curve 3 to curve 5, it consistently seeks the exterior of each corner, coherently with its basic objective (after all, what would happen by attempting to minimize curvature over a steering pad?).

Looking at curve 5, one could still ask himself, why isn't it able to, at least, "pull the line closer to the apex"? After all, this would result into a very similar curvature, and, since we're starting from the minimum-distance solution, it's reasonable to assume the algorithm *went through* a similar solution. My explanation for this is, it hasn't to do with the corner itself, but rather, with the preceding and the following one. You see, the line gets out of curve 4 in the left margin, and then needs to move to the right margin before entering curve 5. In the same period of time, the vehicle, which is facing down-right, must turn to face down-left, and then another time, to turn down-right again. This is all done at comparatively small curvature values, however, if we shorten the available straight section to enter curve 5 earlier, overall curvature in said section will increase since we're reducing distance while keeping the overall "need to turn" roughly constant.

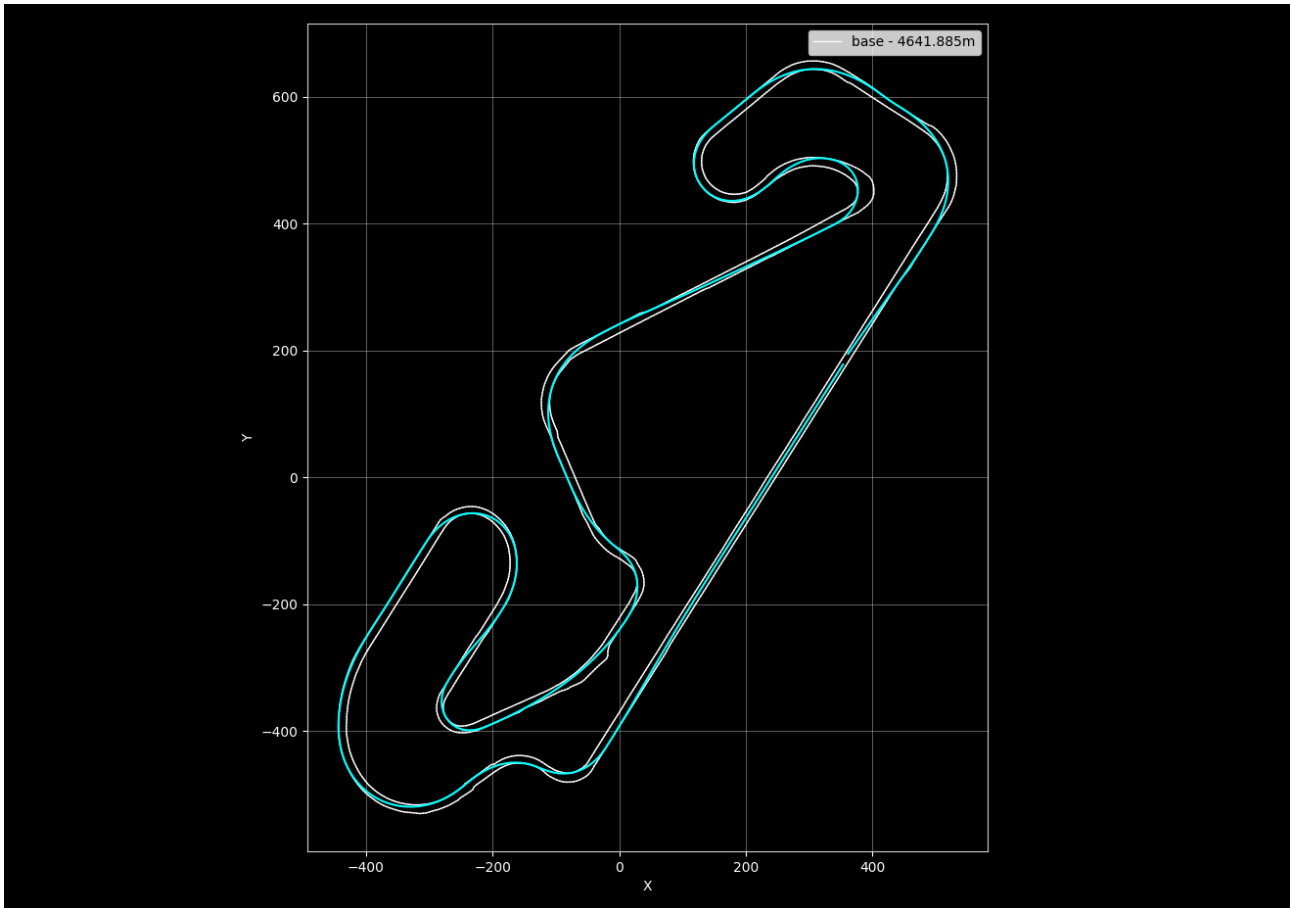
I have no direct proof of this, but I'd say that if the advantage, in terms of curvature, of widening trajectory in curve 5 (going from outside-apex-outside instead of outside-outside-outside) is smaller than the total-curvature-benefit of using a longer straight section for the approach, the algorithm will find no interest in doing it.

Now, without some assumption on the vehicle we're using, it's difficult to *explain* to this mathematical formulation that some curvatures are more important than others. A nice way to do this is to replace curvature in the objective function with its value elevated to some exponent. The idea behind this is to force the algorithm to consider *more important* to reduce already-high curvature values (associated with tighter corners) than to reduce it where it's already low.



With an exponent of 2, the algorithm understands it almost *too well*, and now it becomes a big fan of always taking the inside of corners, save for some pre-corner “preparation” where it nervously oscillates zig-zags to smooth corner entry.

In an extremely scientifically-rigorous shot, I just tried with a value in the middle to get, with an exponent of 1.5, the following line:



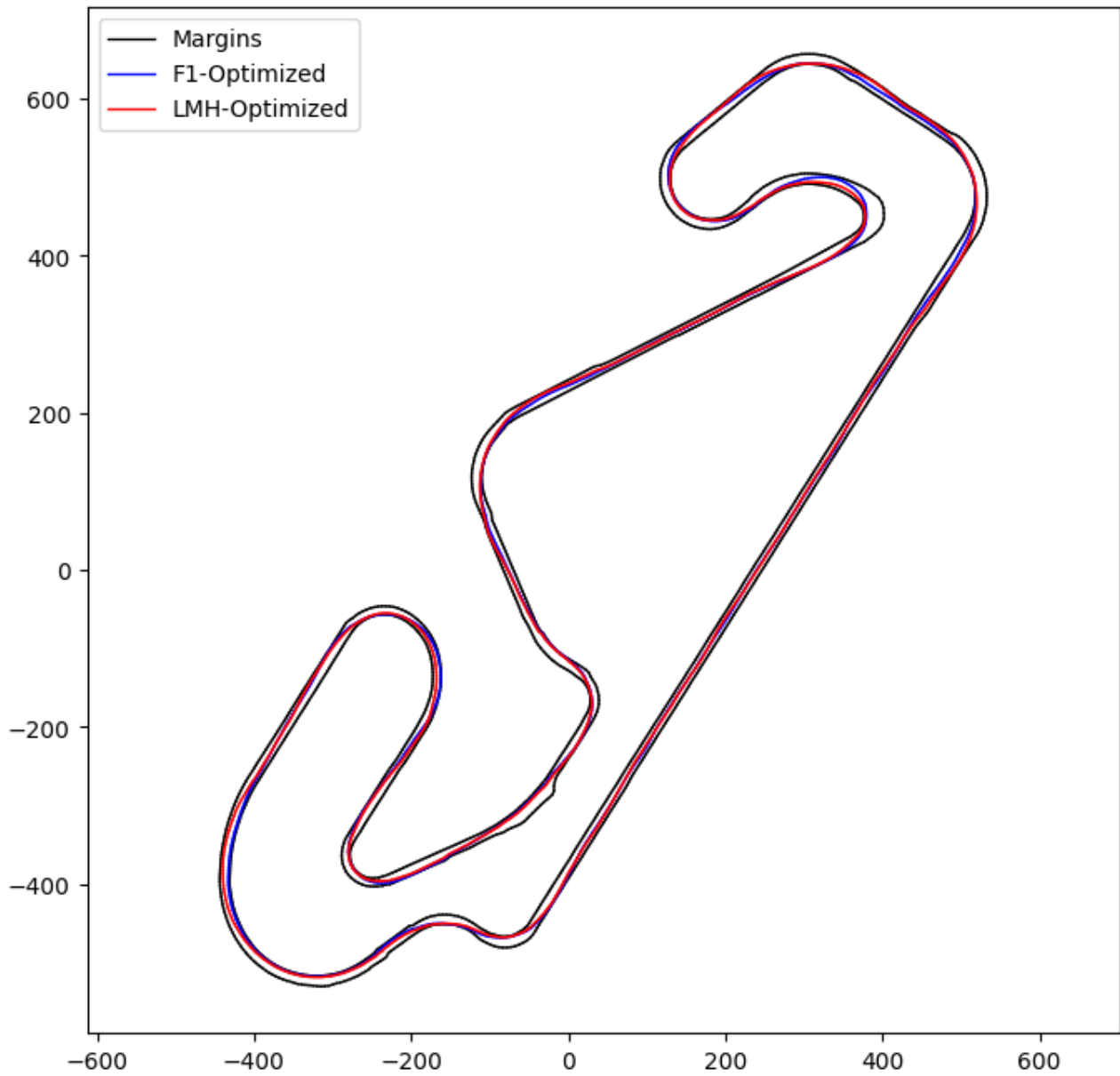
Which is still not perfect (look at curve 11) but starts entering *believable* territory.

## 5.2 Heuristic Trajectory Optimization

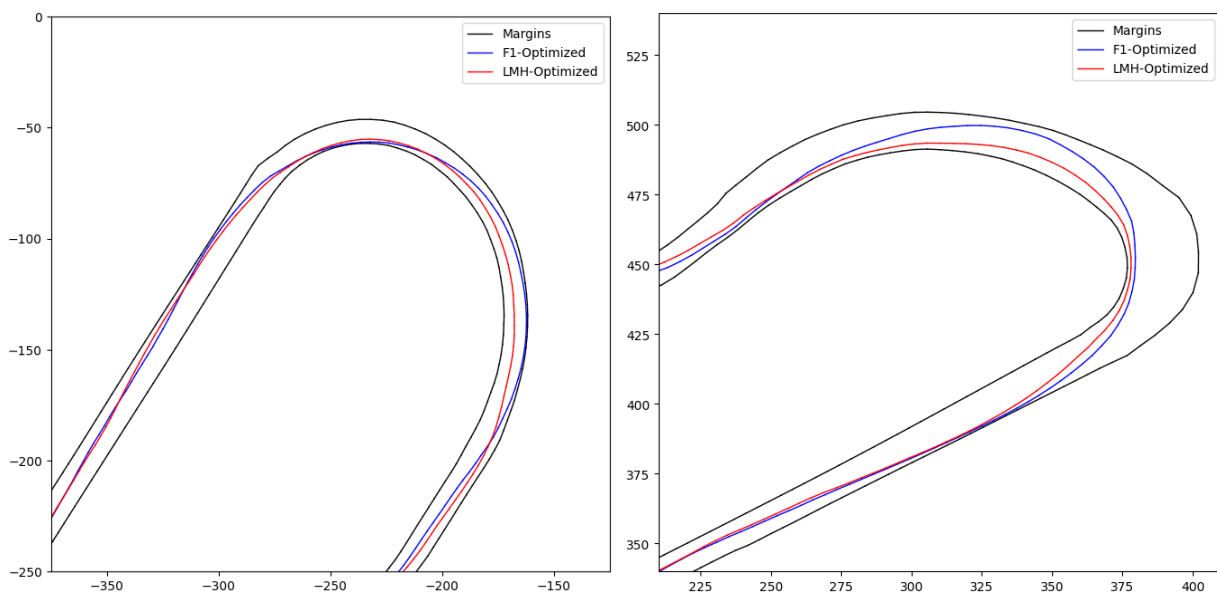
Here, the approach is much simpler, and more brutal. We take a GGV model and a track, and we completely randomize the solution array between -1 and +1. We represent each position of such array in binary, we concatenate them into a N by 8 bits code, and we let genetic optimization do the trick. The obvious shortcoming of this approach is its inefficiency. While the curvature-based approach is based on a completely analytical formulation, which supports multiple optimization approach, this one is basically shooting at the problem and mixing together the best shots. In Python, calculating a single solution (i.e., performing back-and-forth iteration of a specific model on a specific trajectory) can take up to 1 second on my venerable Lenovo T400 laptop.

Due to this, I decided to port the entire algorithm – plus the genetic optimization routine – to C++, a task that took me one week of work, which included a crash-course of Valgrind to solve some nasty memory leaks I had left all around the code. It turns out, when you want to iterate with generations of 1000 individuals, each creating a new 1000-points-long trajectory object, you need to be quite careful with memory allocation.

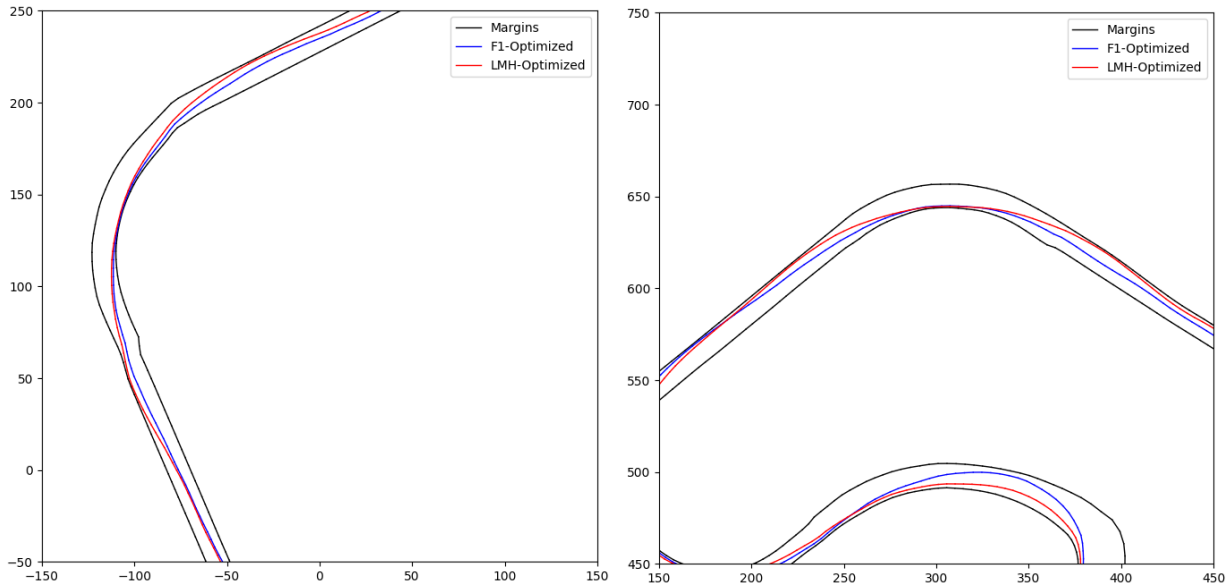
As already experienced with other projects, C++ allows for a time save over Python of around 90%, which in this case makes the difference between a nice demo and something applicable to real cases. For each vehicle, I chose the highest-downforce configuration, then I launched genetic optimization over 400 generations and with 1000 individuals. With some degree of surprise, the two output trajectory came out quite different, as shown in the plot below.



Some of the differences are directly visible to the naked eye, like the ones in corner 4 and 10:



There are however some slightly subtler one which need to be taken into account, in corners 9 and 13:



We'll come back to these in the following section.

An obvious way to validate whether this approach is returning realistic results or not is to take the camera car from Oscar Piastri's pole position lap at Montmelo in 2025 (available on Youtube: [6]) and compare it to the line we have here: while very approximate as a comparison, it can be used at least to determine whether results are plausible or completely wrong. And in F1 case, they seem to be plausible, perhaps with the notable exception of Curve 3 (where the driver seeks the exterior much earlier than in this optimization), and the honorable mention of Curve 10 (where it really seems he finds a very early apex and goes to the outside early on, to better prepare the following corner).



### 5.3 Results

Given that the heuristic search is some tens of times less efficient than the geometric one, does it even guarantee some advantages? How much are we losing if we use the LM trajectory to calculate the F1 time, and back?

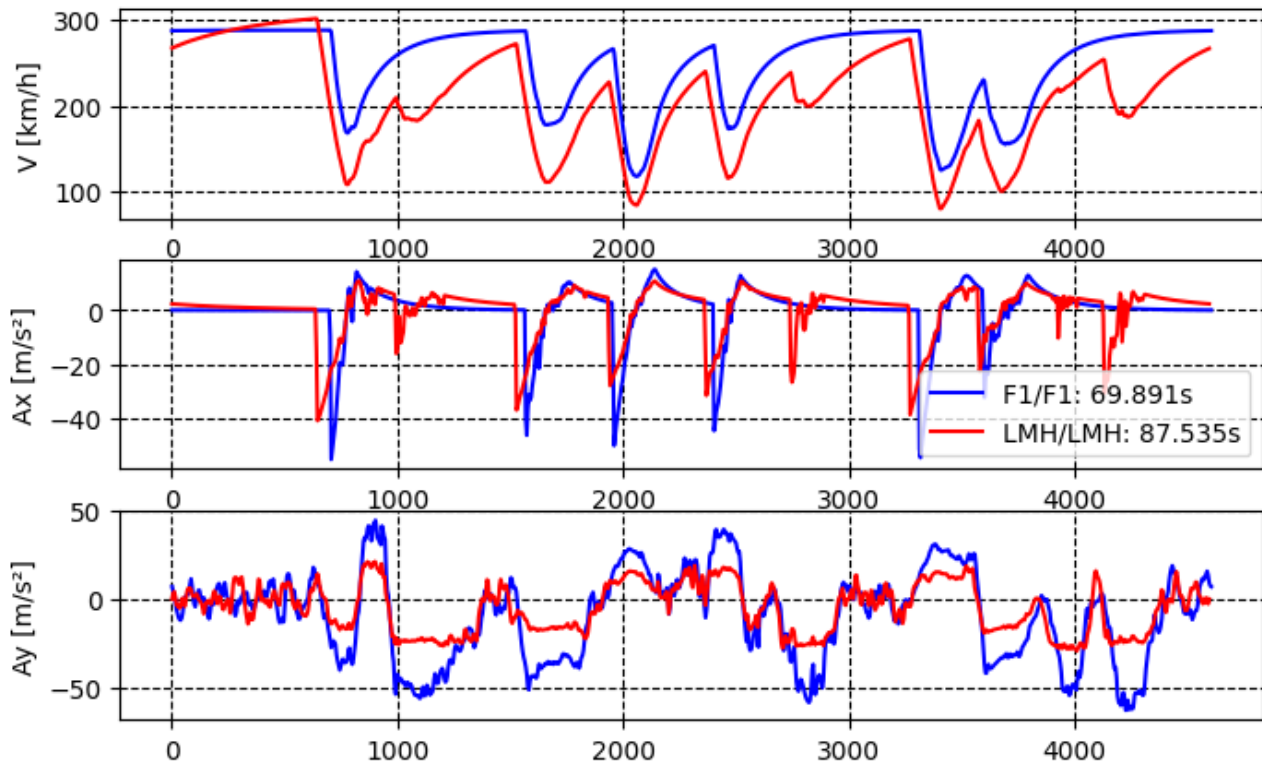
|       |    | Trajectory |        |              |            |            |            |
|-------|----|------------|--------|--------------|------------|------------|------------|
|       |    | F1         | LM     | F1 telemetry | Geom (1.0) | Geom (1.5) | Geom (2.0) |
| Model | F1 | 69,893     | 71,211 | 71,821       | 72,458     | 69,955     | 74,959     |
|       | LM | 89,966     | 87,532 | 90,807       | 90,114     | 87,876     | 94,690     |

We can observe that:

- The geometrically-optimize trajectory is sub-optimal for both vehicles; however, for both, it comes much closer than heuristic optimization of the *other* car in terms of lap time.

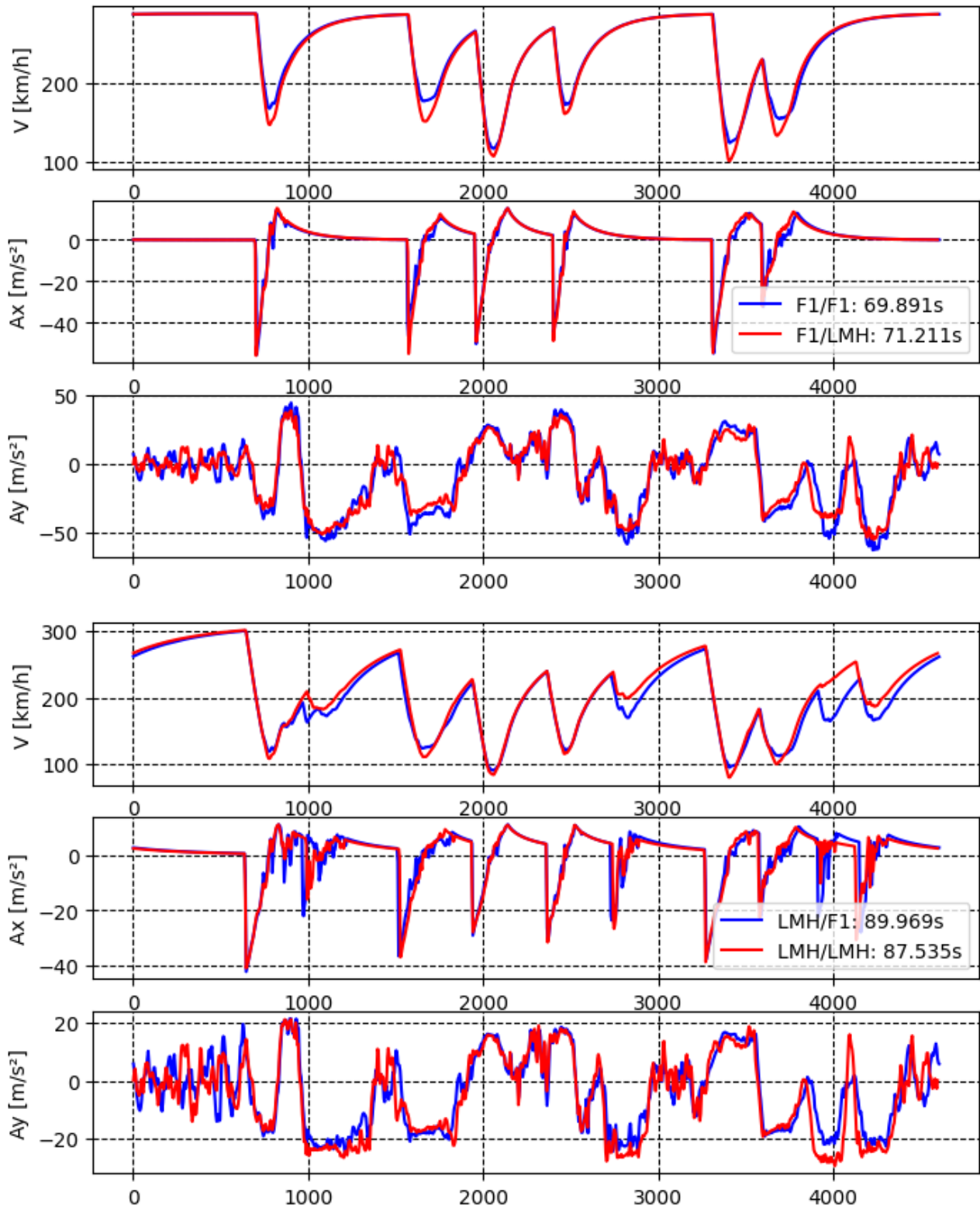
- The competitiveness of the geometric trajectory is strongly dependent on the exponent used for optimization
- Overall, the heuristically-optimized trajectory tends to under-estimate real world lap time (with Formula 1, by two seconds).
- Both the Formula 1 and WEC models lose a significant amount of time (1.3 and 2.4 seconds, respectively) when crossing trajectories.

About this last point, why does this happen? If we compare the speed data of the best laps identified by the heuristic solution, we already get an important hint:



We can see that some corners are practically *invisible* to the F1 speed history: the vehicle is able to negotiate them with no effects on its ability to accelerate. The LM vehicle, instead, has to break, enter the corner, and reaccelerate on exit.

Things get clearer when keeping trajectory constant, and analyzing the behavior of the F1 model over the two trajectories (first plot below) or the LM model over the two trajectories (second plot below):



We can see that:

- Apex speeds of tight corners, with the LM trajectory, are almost always lower:
  - With F1, this disadvantage comes at no real benefit, since the lower apex speeds results in a lower line for the whole straight section following.
  - With the LM, corner exit seems to always be better with its own trajectory, carrying more speed for the whole duration of the subsequent straight section.
- Focusing on the LM vehicle plot, we see that in two corners (9, 13) the F1-Trajectory forces the vehicle to brake much more than with the LM-Trajectory. Going back a couple of pages,

we can see how this correlates to the relatively-subtle differences we saw in those two corners: the F1 optimization sees no benefit in increasing radius at entry/exit, since the vehicle is already able to withstand that curvature with no speed increase; the LM car, instead, given its lower ability to generate cornering force, benefits from a smoother entry/exit line.

## 6 Some sensitivity analyses

Among the collateral effects of having developed a plausible LM model, we can roughly calculate how BoP decisions may affect the lap time performance of a vehicle. In this case, we'll just keep the original trajectory without further modifications – i.e, we'll ignore the fact that a different configuration may benefit from a slightly different best trajectory in this case.

| Mass | 1030kg | 1040kg | 1050kg | 1060kg | 1070kg | 1080kg |
|------|--------|--------|--------|--------|--------|--------|
| Time | 87,535 | 87,742 | 87,947 | 88,148 | 88,346 | 88,541 |

In the above table, lap times for different values of the vehicle mass parameters are reported. At least in this circuit, such mass increases as the ones allowed by BoP decisions have a comparatively strong impact, each 10kg resulting in an increase in lap time by around 0.2s, or 0.23%, which is reasonably close to the value proposed in an article in the Racecar Engineering website [7].

| Power | 485kW  | 490kW  | 495kW  | 500kW  |
|-------|--------|--------|--------|--------|
| Time  | 87,773 | 87,692 | 87,613 | 87,773 |

Decreases in maximum power are not as dramatic, apparently, as they account for roughly 0.12s per each 10kW removed. This effect may be underestimated due to the fact that the model is over-representing both static and aerodynamic load acting on the rear wheels, hence the part of the speed-maximum acceleration curve which is determined by the power limit rather than friction may be larger than in the real vehicle.

## 7 Conclusions

Even a simplified GGV model, with a good pre-treatment of real world trajectory, can provide good correlation in terms of both speed and acceleration. Various strategy have been attempted to improve the curvature signal, however each improvement on peak values seemed to be connected to either lower reliability of the filtering routine, or to a still not usable signal due to abrupt variations in either the curvature or the distance array.

The attempt to extract a Formula 1's physical parameters from repeated fitting of multiple pole position lap times (and telemetries) from different circuits was mostly unsuccessful: while lift and drag coefficients show some correlation, tire friction parameters, as well as weight and aero load distribution ones do not appear to be trustworthy.

While it is possible to obtain a good approximation of an “ideal trajectory” with geometric optimization, trajectory does depend on the model's properties, with two different vehicles producing largely sub-optimal results when dealing with each other's optimal solution.

As per the Hypercar lap time at Montmeló, I have to admit I trust the first two methods (making a proportion from other F1 and WEC results, and looking at some videogame leaderboard) more than the more complex one; however, they appear to be remarkably close together. So I guess 1'27"5 might be a realistic estimate, after all: in a couple of days, we should be able to tell.

## Thanks

Un sentito ringraziamento ad Alfredo Facciuolo, che da cinque anni mi fornisce i server su cui far girare le mie routine di simulazione, ed il supporto tecnico per farli funzionare. Senza di lui dubito che avrei mai potuto pubblicare nessuno degli articoli di questi ultimi anni.

## 8 References

- [1] «TrackTitan LM Ultimate Barcelona LMH leaderboard,» [Online]. Available: [https://app.tracktitan.io/leaderboards/leMansUltimate/circuit\\_de\\_barcelona/classes/lmu\\_hypercars](https://app.tracktitan.io/leaderboards/leMansUltimate/circuit_de_barcelona/classes/lmu_hypercars). [Consultato il giorno 10 10 2026].
- [2] A. Patrucco, «Lap Time Simulation: steps 5, 6 and 7, and some divagation,» *LinkedIn*, 15 10 2025.
- [3] Tracing Insights, «Tracing Insights 2025 season data,» [Online]. Available: <https://github.com/TracingInsights-Archive/2025>. [Consultato il giorno 12 10 2026].
- [4] M. Guiggiani, *Dinamica del Veicolo - Nuova Edizione*, Torino: Città Studi Edizioni, 2012.
- [5] F. Fazi, «L'aerodinamica delle auto di Formula 1 (Auto Tecnica),» *Auto Tecnica*, p. 31, Ottobre 1989.
- [6] Youtube, «Oscar Piastri's Pole Lap | 2025 Spanish Grand Prix | Pirelli,» [Online]. Available: <https://www.youtube.com/watch?v=pZnBbzUuJmQ>. [Consultato il giorno 12 10 2026].
- [7] S. Raymond, «Bop options available for changes,» 19 Marzo 2023. [Online]. Available: <https://www.racecar-engineering.com/tech-explained/lm-tech-explained/bop-options-available-for-changes/>. [Consultato il giorno 12 10 2026].