Lateral controller

Antoine Roux

November 13, 2023

1 Intro

We're using a pure-pursuit approach: find a point on the line of waypoints to drive towards, find the circle passing through this target point and the car, tangent to the car direction (see the other document in the repo about pure-pursuit for detailed derivations). The target generator node outputs a target curvature: the curvature of the circle on which the car needs to drive to reach the target point (the curvature is the inverse of the radius).

The problem we're trying to solve in this document (corresponding to the feed-forward term of our lateral controller) is: what is the PWM command to send to the steering servo of the car so that the car turns on such a circle.

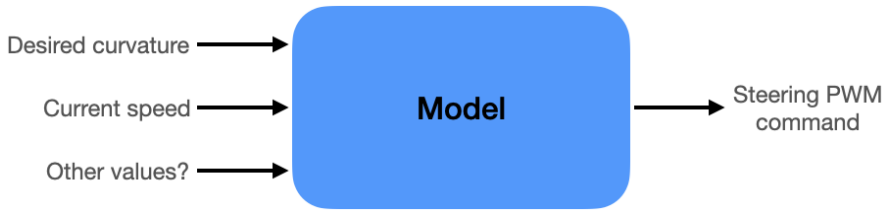

Figure 1: Representation of the model inputs and outputs

2 V1: bicycle model and linear interpolation

2.1 The bicycle model

One of the most basic approaches to model the behavior of a car is the bicycle model. It has some limitations (mainly: representing a 4-wheeled vehicle as a 2-wheeled vehicle, the fact that it does not consider the dynamics of the vehicle), but was a good starting point for us.

I'm skipping the derivation of the bicycle model here, derivation which results in the following equation:

$$\texttt{steering\_angle} = \arctan(\texttt{curvature} \times \texttt{WHEELBASE}) \tag{1}$$

2.2 From steering angle to PWM command

So now that we know how to go from `curvature` to `steering_angle`, we "just" have to find a relation between the `steering_angle` and the PWM command sent to the steering servo.

Here are some approaches that can be used to find this relation:

- geometrically compute the relation by measuring lengths of arms and angles
- send a lot of PWM commands to the steering servo, measure the resulting steering angle and interpolate

Because it is quite tricky to measure precise lengths and angles in the vehicle, we decided to go with the second approach. And with only two datapoints: at a steering angle of 0 and at a steering angle of 30 degrees (the maximum what the front wheels will turn).

PWM command (unit-less)	steering angle (degrees)
98	0
125	30

We defined $\texttt{STEER\_IDLE\_PWM} = 98$; $\texttt{PWM\_DIFF\_AT\_MAX\_STEER\_ANGLE} = 125-98 = 27$; $\texttt{MAX\_STEER\_ANGLE} = 30$ degrees, resulting in the following model:

$$\texttt{pwm\_command} = \texttt{STEER\_IDLE\_PWM} - \texttt{steering\_angle} \times \frac{\texttt{PWM\_DIFF\_AT\_MAX\_STEER\_ANGLE}}{\texttt{MAX\_STEER\_ANGLE}} \tag{2}$$

2.3 Correcting the model using physical world measurements

When testing this lateral controller on the car, we noticed that the car was not turning enough when trying to turn on circles of given a given radius. According to the bicycle model (Equation 1), a vehicle with a wheelbase of 0.406m and a maximum steering angle should be able to turn on a circle of radius:

$$\begin{aligned} & \texttt{steering\_angle} = \arctan(\texttt{curvature} \times \texttt{WHEELBASE}) \\ \iff & \texttt{steering\_angle} = \arctan(\frac{\texttt{WHEELBASE}}{\texttt{radius}}) \\ \iff & \tan(\texttt{steering\_angle}) = \frac{\texttt{WHEELBASE}}{\texttt{radius}} \\ \iff & \texttt{radius} = \frac{\texttt{WHEELBASE}}{\tan(\texttt{steering\_angle})} \\ & \text{So: } \texttt{radius} = 0.70\text{m} \end{aligned} \tag{3}$$

However, we measured that the radius of the smallest circle we could drive on (at the slow speed we tested at) is 1.25m. This means that, for a circle of radius 1.25m, the car should turn all the way, but the bicycle model was telling it to run less than all the way. This is likely caused by a limitation of the bicycle model, that we haven't investigated for now.

To account for this effect, we've defined the `EFFECTIVE_MAX_STEER_ANGLE`: the steering angle that the car appears to have when turning the wheels all the way. Which, at the low speed we tested at, was around 17 degrees. This resulted in a adapted version of Equation 2:

$$\texttt{pwm\_command} = \texttt{STEER\_IDLE\_PWM} - \texttt{steering\_angle} \times \frac{\texttt{PWM\_DIFF\_AT\_MAX\_STEER\_ANGLE}}{\texttt{EFFECTIVE\_MAX\_STEER\_ANGLE}} \tag{4}$$

With this model, the car was able to turn along circles at low speeds very precisely. We knew we might need to go for a more elaborate model at high speeds, but the current model was good enough at this point in time

3 V2: end-to-end model and region-based linear interpolation

3.1 Why do we need a better model?

The model described in Section 2 did the trick while we were testing at speed ranges close to the one where we did the initial measurement (below 3m/s). But, as expected, we realized that when testing at higher speeds (around 7m/s), the car was not turning enough to follow circles. So we decided to:

- take more measurements of the relation between: `speed`, `pwm_command`, and `radius`.
- use this data to fit a better model between the 3 parameters

3.2 Taking measurements

The process I followed was:

- send a constant PWM throttle command and send a constant PWM steering command
- record a bagfile of the car turning in circles
- ... repeat for different throttle and steering commands

So at the end, I ended up with 31 datapoints of `speed`, `pwm_command`, and `radius`. Here they are on a 2D scatter of `speed` and `radius`:

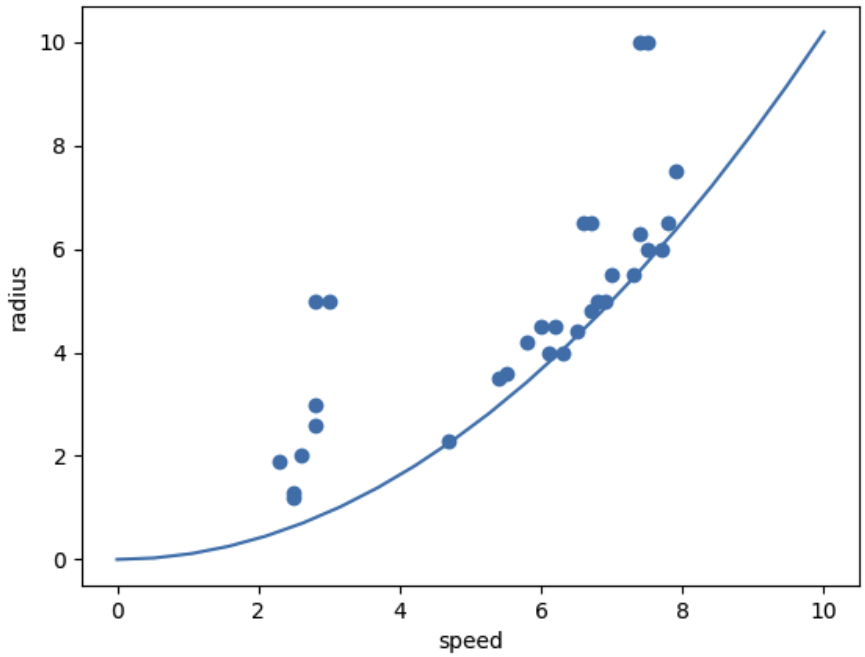

Figure 2: Scatter of the measured datapoints

The blue line represents the pairs {`speed`, `radius`} that have a lateral acceleration of 9.81 m/s^2, which illustrates the fact that I did most of the testing at the limits of lateral acceleration of the car.

Moving on to a 3D representation:

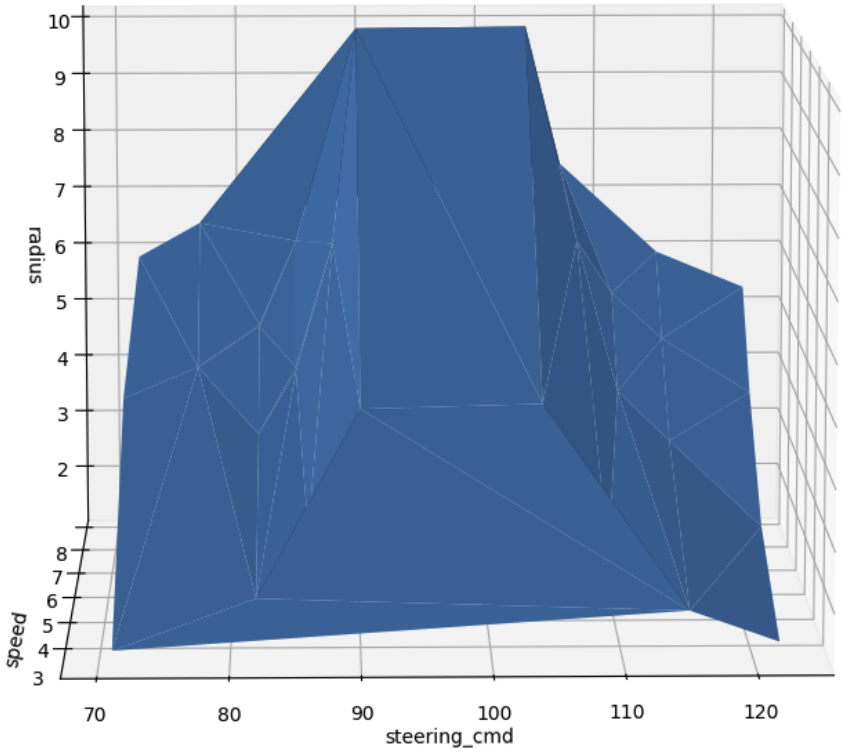

Figure 3: Surface representing the relation between {`speed`, `pwm_command` and `radius`}

We can see that:

- the surface is mostly symmetrical around a PWM command of 98, which is reassuring
- for a given PWM command (for instance 71), the `radius` of the circle increases with `speed`, which matches from the observation made in testing: at high speed we need to turn more to reach the same radius

Because of the symmetry, all the further modeling can be made by considering the `steering_diff`: the difference between the PWM command and 98 (the idle point of the steering). Here is how the plot looks when replacing the `steering_command` by the `steering_diff` and "rotating" the plot so that the `radius` and `speed` are inputs of the model, and `steering_diff` the output:

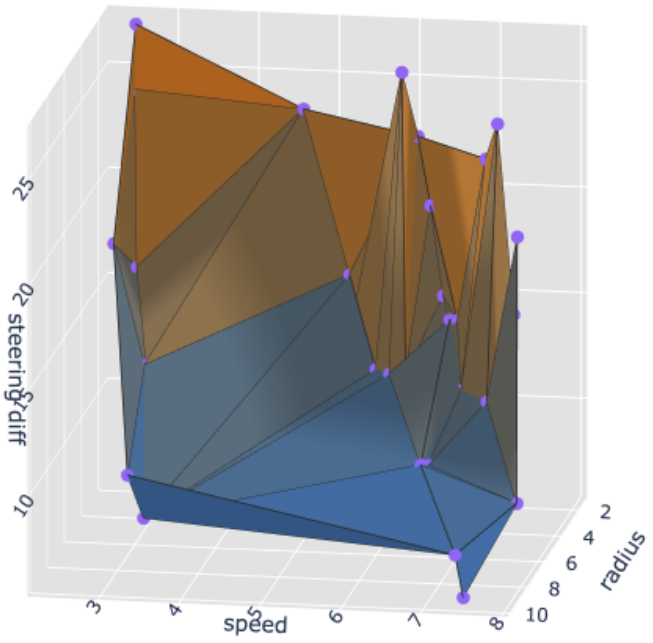

Figure 4: Surface representing the relation between {`speed`, `radius`} and `steering_diff`

We can measure two points of interest:

- the top-left point: at a `speed` of 2.5m/s, turning on a circle of `radius` 1.2m needs a `steering_diff` of 27.
- the second-leftmost peak point: at a `speed` of 6.1m/s, the same `steering_diff` of 27 results in a circle of radius 4m.

Let's now overlay our model from Section 2 (represented with blue dots) and see how it matches with these measured datapoints:

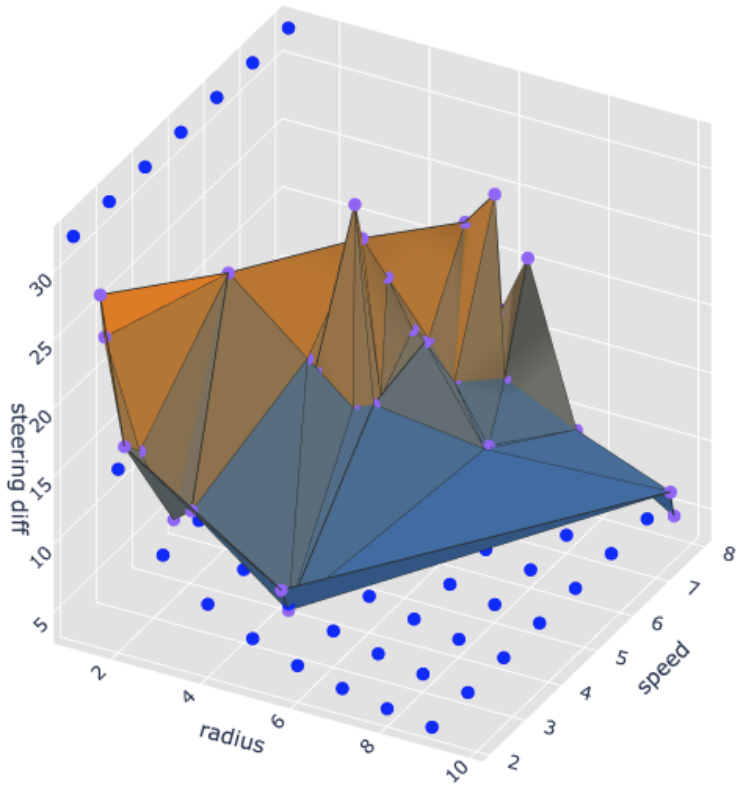

Figure 5: Representation of our first model (view 1)

As expected, we can see that the blue dots match the surface pretty well at low `speed`. But if we now look at high speed:

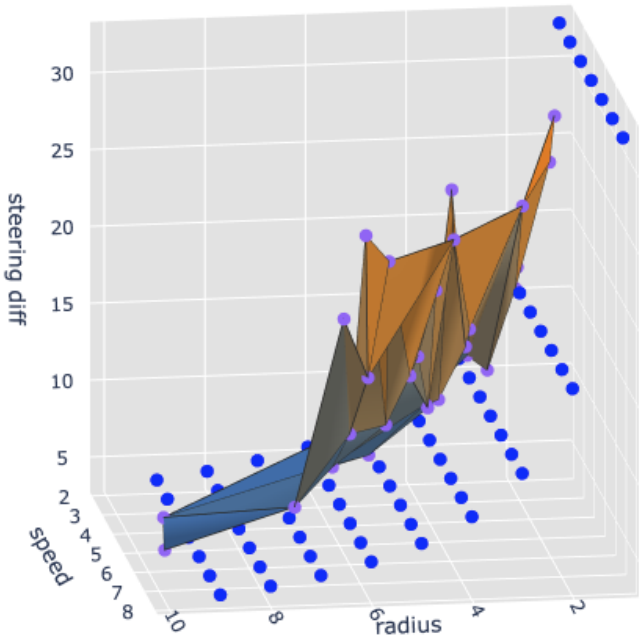

Figure 6: Representation of our first model (view 2)

We can see that they are far below the surface: the model underestimates the `steering_diff` required to make the circle. It's clearer when plotting the difference between the real `steering_diff` and the predicted `steering_diff` using the model:

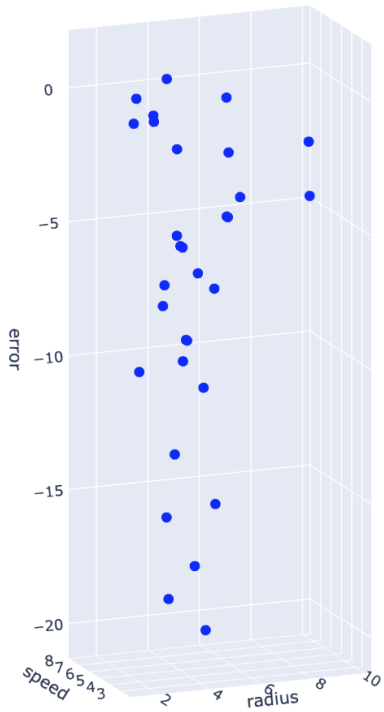

Figure 7: Evaluation of our first model

For some points, the model from Section 2 would underestimate the required `steering_diff` by as much as 20 units! An ideal model (not considering overfitting risks) would result in all the points at a z value of 0 in this graph.

Conclusion: at this point, we have now verified that our model created at low speed is pretty good at low speed, but underestimates the `steering_diff` at high speeds, which not-only makes sense given how we had built it, but also matches with what we've observed during testing.
So let's try to create a better model.

3.3 Fitting a better model

The approach in Section 2 was based on a physical model of vehicle and its kinematics: going from desired curvature to steering angle, and then from steering angle to pwm command. But now that we had data linking the pwm command the radius (which is just the inverse of the curvature), we decided to go with an "end-to-end" approach: building a model that directly maps the desired curvature to the pwm command, without using the steering angle as an intermediate.

3.3.1 Linear and non-linear regressions

The surface does not look like a plane, which means that a linear regression (*i.e.* a model of the form: $\texttt{steering\_diff} = \alpha + \beta\ \texttt{radius} + \gamma\ \texttt{speed}$) is probably not a good idea. I gave it a try anyway just to see, here is the result:

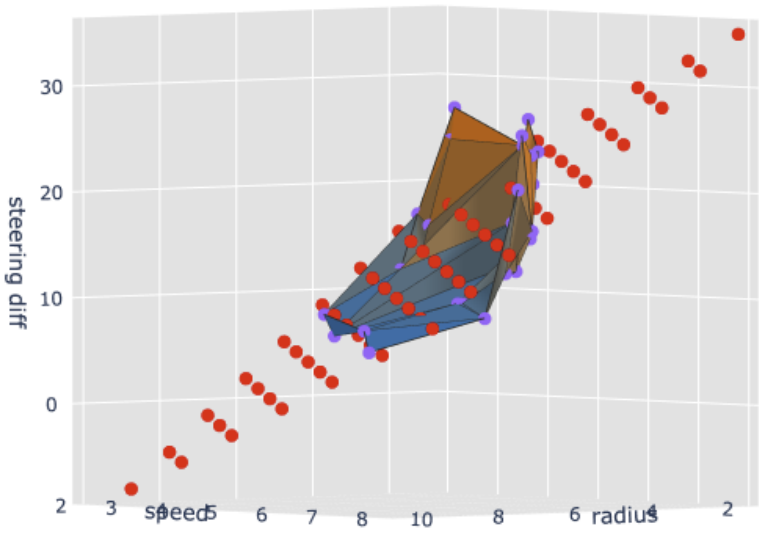

Figure 8: Linear regression

Unsurprisingly, the fit is bad. I also gave it a try with models of dimension 2 and 3, that fit the surface better, but get very bad for radii and speeds far from the points used for training:

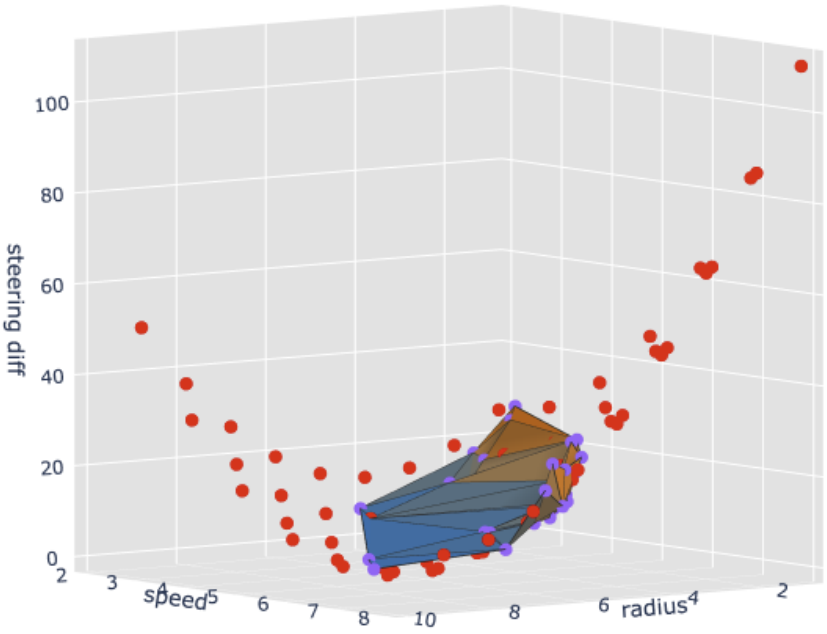

Figure 9: Quadratic regression

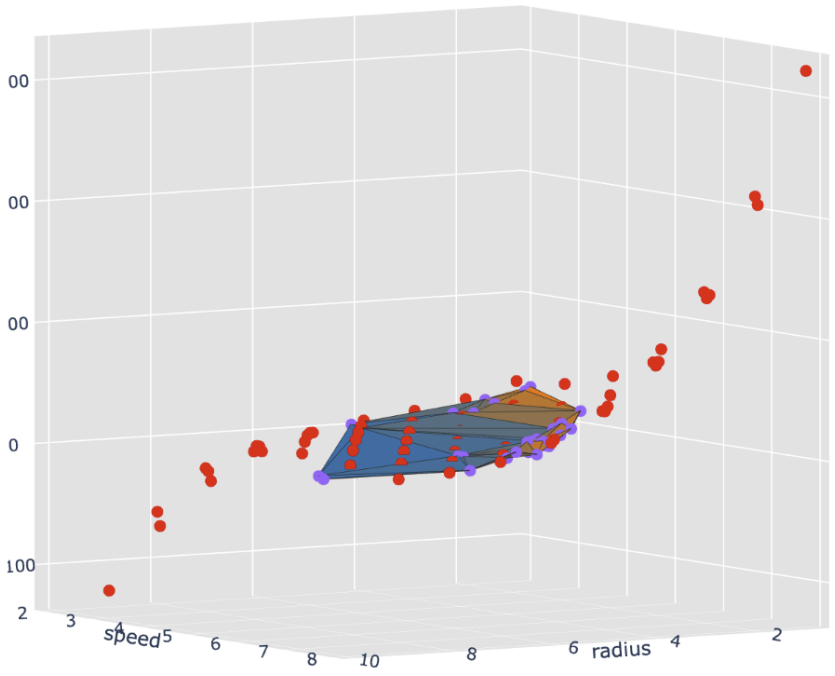

Figure 10: Degree 3 regression

So, until we have more points (and more evenly spread points in the speed and radii ranges) for training, going with one of these 3 models is not a good idea.

3.3.2 Nearest neighbors

I also gave a try to k-nearest-neighbors regression:

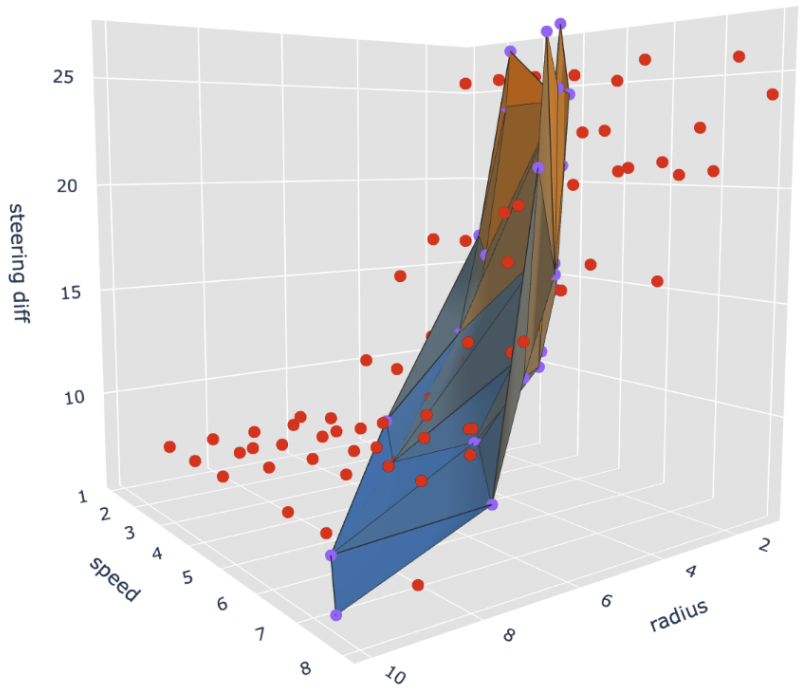

Figure 11: k-nearest-neighbors regression

We can see that the performance is very poor, likely because (once again) of the little number of points used to build the model, as well as the fact that are poorly distributed over the ranges of radii and speeds. It was here with k=2 neighbors, but the performance we not better for other values of the number of neighbors.
One remark though, given that the `radius` and `speed` are in 2 different units, and possibly in two different ranges, it would be better to normalize the values in a certain range. I don't think it would help much here, because they happen to be in similar range (roughly between 0 and 10).

3.3.3 Same as V1, but regions-based

Because the approach from Section 2 was working pretty well at low speeds (and because we needed something to test the very next day), I've decided to follow the same approach, but instead of having a single value of the coefficient linking radius and pwm value for the entire speed range, use different coefficients for different speed *regions*.

So instead of having a relation of the form:

$$\texttt{steering\_diff} = \frac{\texttt{curvature}}{\texttt{COEFF}} \tag{5}$$

Going with:

$$\texttt{steering\_diff} = \frac{\texttt{curvature}}{\texttt{coeff(speed)}} \tag{6}$$

To keep things simple, I've divided the speed space in 4 regions: $[0, 1.5]$; $[1.5, 5]$; $[5, 8]$; $[8, +\infty]$. For each region boundary, I've estimated the value of the coefficient at each region boundary with Equation 6 and using the data to get numerical values:

$$\texttt{coeff(speed)} = \frac{1}{\texttt{radius(speed)} \times \texttt{steering\_diff}} \tag{7}$$

In order to not get steering jumps when going from one region to the next, the coefficient inside a region in a linear interpolation of the values at the two boundaries.
Which, in the end, yields the following algorithm:

```
bound_region_1 = 1.5
bound_region_2 = 5
bound_region_3 = 8
coeff_region_1 = 27 * 1.25
coeff_region_2 = 24 * 2.3
coeff_region_3 = 26 * 4
if speed <= bound_region_1:
    coeff = 1 / coeff_region_1
elif speed > bound_region_1 and speed <= bound_region_2:
    coeff = 1 / (coeff_region_1
        + (speed - bound_region_1)
        * (coeff_region_2 - coeff_region_1)
        / (bound_region_2 - bound_region_1)
    )
elif speed > bound_region_2 and speed <= bound_region_3:
    coeff = 1 / (coeff_region_2
        + (speed - bound_region_2)
        * (coeff_region_3 - coeff_region_2)
        / (bound_region_3 - bound_region_2)
    )
elif speed > bound_region_3:
    coeff = 1 / coeff_region_3

sterring_diff = curvature / coeff
return min(27, sterring_diff)  # min is here to avoid crazy steering values
```

Let's overlay the real datapoints and points generated with this new model:

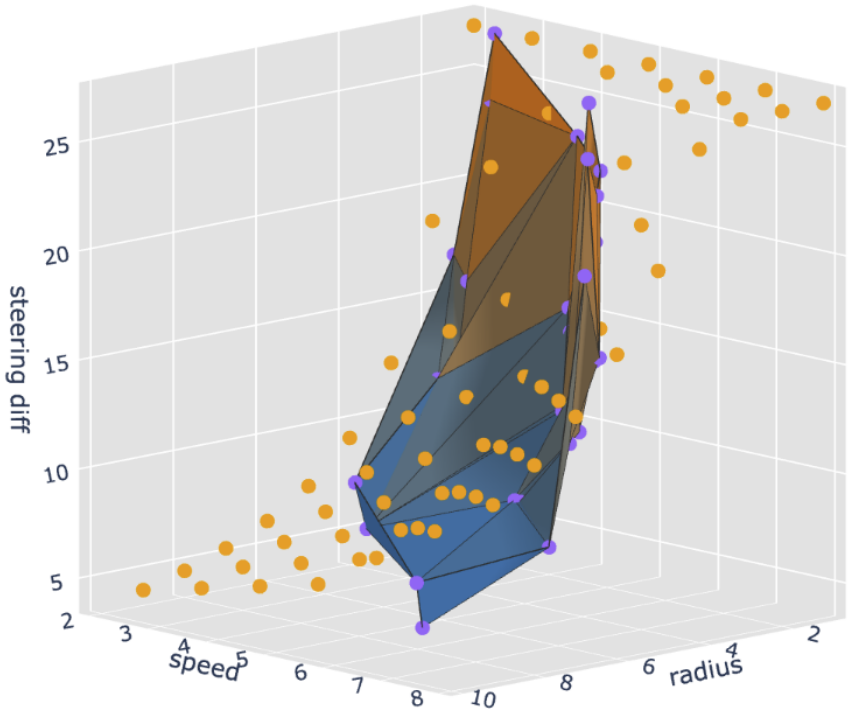

Figure 12: Overlay of the data points and new model

It looks much better than the model from Section 2! Looking at the error this model yields on the measure datapoints, we see that it's much better than the old model (blue is the old model, orange is the new model):

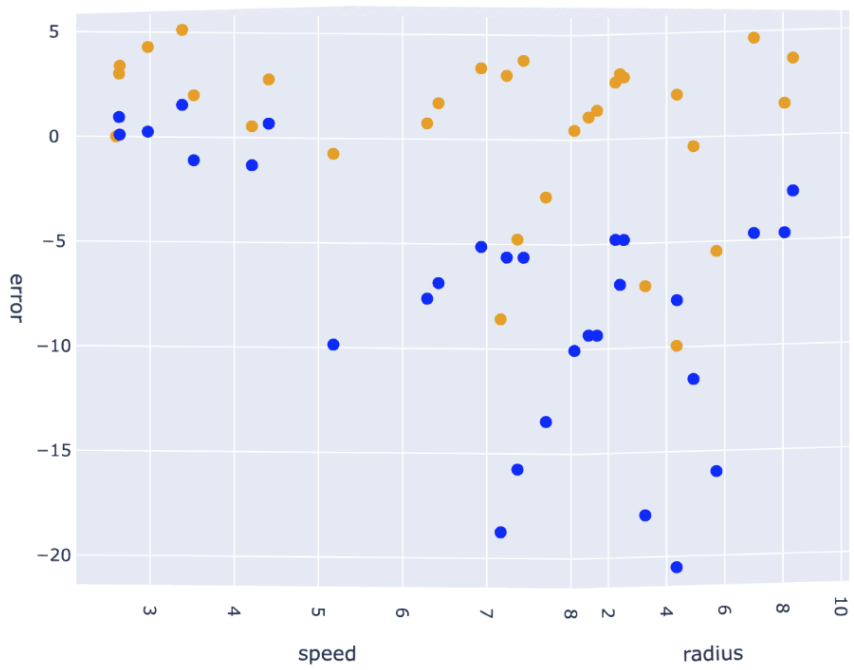

Figure 13: Evaluation of the new and old models

The points are overall much closer to 0, and the largest error is now of "only" 10 units.
It is definitely not perfect though:

- an underestimation of 10 units means that the car will still under turn pretty heavily in some cases
- the fact that there are some positive error means that the car will over turn in some cases, which could lead to instability

This new model is not perfect, but it's better. When testing it in real life, we've confirmed that the car is now able to perform some U-turns that it was not able to do with the previous model.

3.3.4 Next steps

A first and relatively next step is to optimize the regions (bounds and associated coefficients) to minimize the error on the *training* set.
Another next step is to record more datapoints to: either improve the regions-based model, or be able to revisit some of the other models like the k-nearest-neighbors regression.