Project 3 – PID control

In the course of my studies and work life I had to tune some PID controllers working in real life. Usually experienced control engineers (I am not one) tune their PID controllers by a heuristic approach (by hand) or using e.g. rules of Ziegler/Nichols to estimate controller parameters if they can properly measure step response times. Of course it is not always possible to apply a step response to all systems (e.g. start of a very heavy turbine) which might become unstable and very dangerous. So ins some cases its necessary to rely on calculations before applying load on a system → but there are very highly specialized and experienced control engineers working on such systems.

I will start to hand tune the parameters for the steering and velocity PID and see how this works out. Then I might try to implement the parameter optimization method to further increase the found parameters using them as initial values. But as I will state later on, imho the simulation setup is not providing the proper tools to do that.

Spawning of the car

Because the car, when it spawns in the world, drops from a height, I introduced a spawn counter, which should prevent the controllers to start working until the car hits the ground. This helped to get rid of the noise at the beginning of the simulation introduced by the drop.

Starting on my local machine

I again start my work on my local machine, because I am more comfortable working in my Eclipse Editor rather then online in the web browser and to have something to start with on the online workspace – saving some GPU time. Knowing that the tuning will not be applicable, because apparently my old laptop has issues with time dependent calculations, when running running a heavy simulation.

Steering - P parameter

I am starting with the P parameter, which theoretically at least should be able to keep the car on the road. I expected to see the car overshooting and osculating around the CTE, but the oscillation got worse and the car eventually crashed.

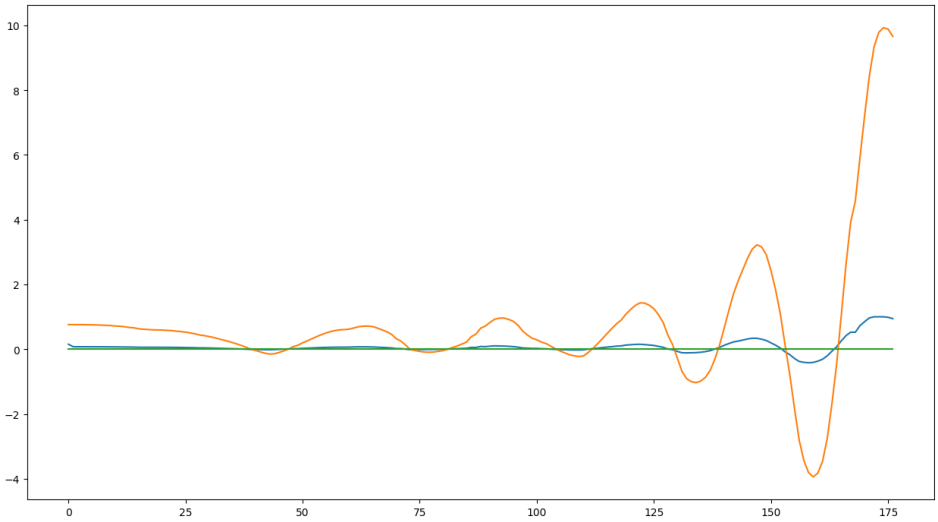

Illustration 1: P 0.1 D 0.0 I 0.0 - throttle pos 0.3, cte (orange), steering angle (blue), zero error (green)

So the P parameter is too high and overshooting. I tried to reduce the parameter by factor of 10 and then find my way into the middle. The controller was not able to reduce the error with this small P value as expected outputting almost no steering angle to the wheels (blue). The car got from the track in the first corner.

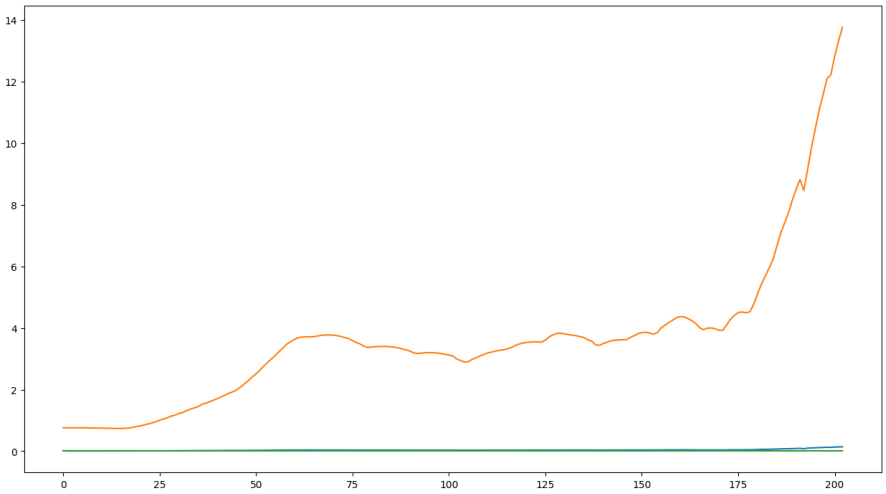

Illustration 2: P 0.01 D 0.0 I 0.0 - throttle pos 0.3, cte (orange), steering angle (blue), zero error (green)

Next I used a value in the middle of the two previous attempts. Again it started to oscillate around a what it seem to be a bias zero value, but then overshooting badly.

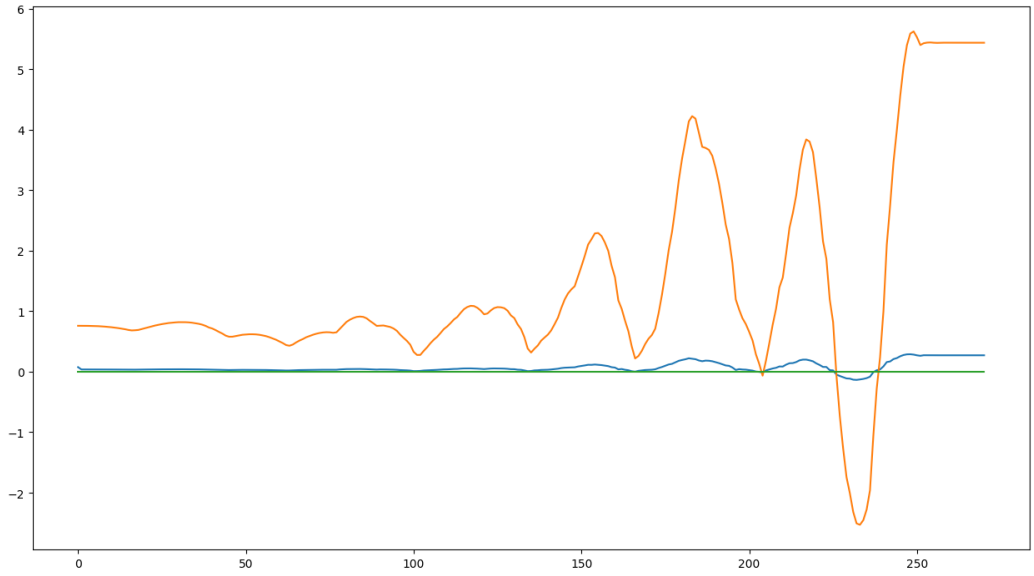

Illustration 3: P 0.05 D 0.0 I 0.0 - throttle pos 0.3, cte (orange), steering angle (blue), zero error (green)

I give it a one last shot on my local machine, then I switch to the online workspace. I have the feeling that my control application gets the cte error of the car way to late, resulting in a delayed reaction on the error, which causes the system to get instable. Again reducing the P parameter to the middle of the lower part of my search interval. I am doing a binary search here to estimate the parameter. And the result again was not very satisfying.

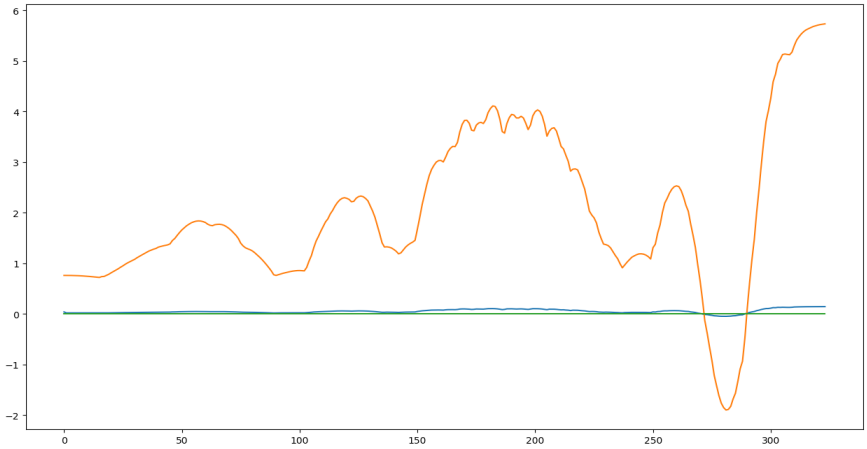

Illustration 4: P 0.025 D 0.0 I 0.0 - throttle pos 0.3, cte (orange), steering angle (blue), zero error (green)

Dead time

I am no control engineer but I think I have quite a long dead time between simulation response and control output. Which leads to this oscillating and even worse instable behavior of the system. At least my steering outputs seem to arrive late into the simulation.

Speed

To be able to control the speed properly I tuned another PID controller, like it was proposed by the code. Again I started with the P parameter until I saw a good response and continued to reduce the systematic bias by increasing the I parameter. I first clipped the throttle position to be positive, I felt that braking should be controlled for itself, because it has entirely different time behavior. Later I allowed the controller also to brake because the steering needed the the slow speeds to react.

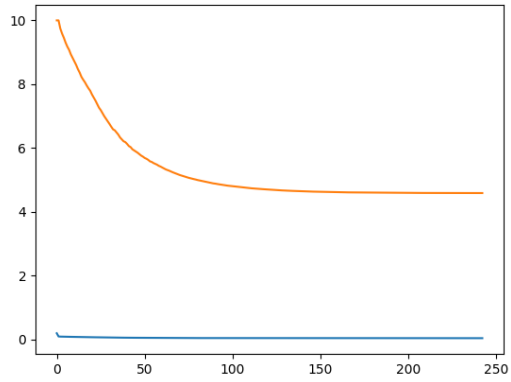

Illustration 5: P 0.01 I 0.0 P 0.0 - target speed 10 mph- orange speed error – blue throttle pos

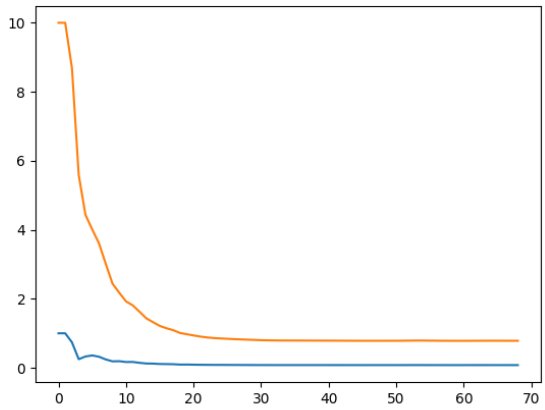

Illustration 6: P 0.1 D 0.0 I 0.0 - target speed 10

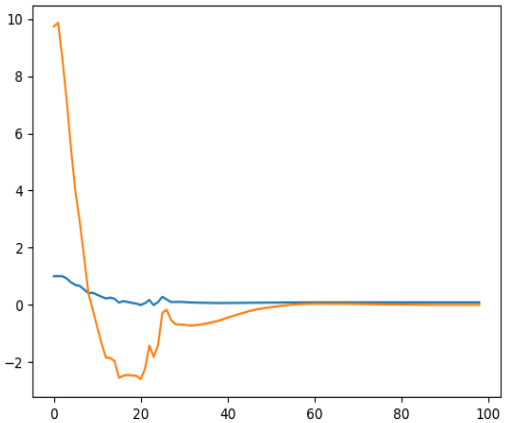

Illustration 7: P 0.1 D 0.0 I 0.1 - target speed 10mph - overshooting

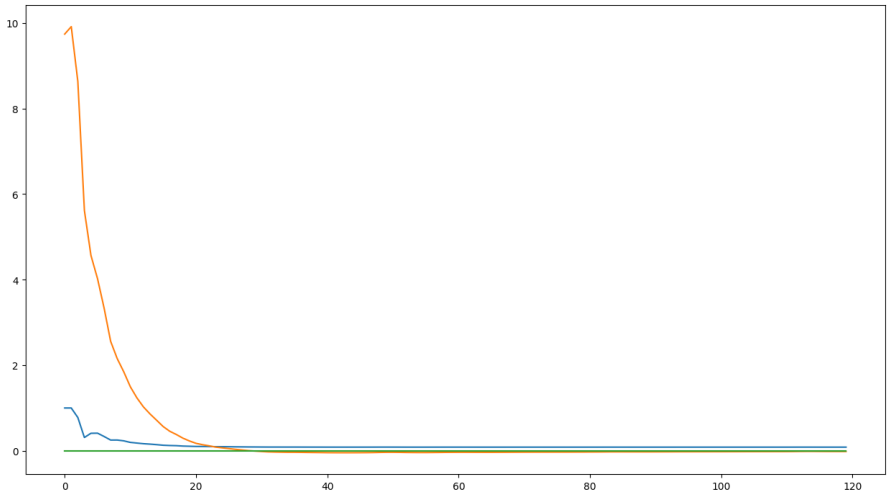

Illustration 8: P 0.1 I 0.015 D 0.0 - target speed 10 mph - good enough

I read that some of my fellow students coupled the speed to the set steering angle, reducing the speed when the car has to steer sharply into the right direction. This makes sense, because the car then has “more time” for steering while traveling distance according to its kinematic restrictions (of the bicycle model? It is not mentioned what the simulation is using).

Final result local machine

With the speed controller implemented and linearly coupled to the output of the steering controller, I gave it a final shot on my local machine before switching to the on line workspace. At lest the car passed an entire lap without crashing, even it oscillated badly around the center of the track.

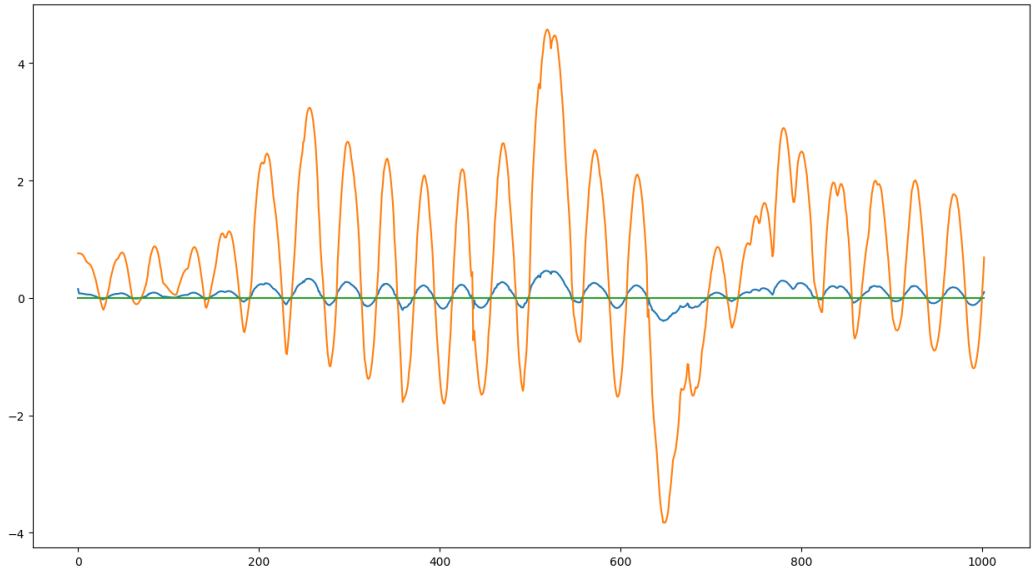

Illustration 9: P 0.1 I 0.0 D 0.0 - target speed 20 mph

Switching to on-line workspace

To not to repeat my last error to just tune parameters of the simulation on my local machine, I moved onto the online workspace and tried the tuning in the online simulation which runs much smoother then the simulation on my local machine.

I had luck, the parameters I tuned worked even better on the online simulation. It seems that on my local machine again my controller application gets the data later then running on the workspace, which leads to a more oscillating behavior.

Nevertheless I started over to tune the parameters. Again I increased P parameter until the car stayed on the road and tended to over steer a little bit. This worked so much nicer than on my local machine. It never started to get instable compared to my first tries on my local machine. I managed to keep the car on the road only using the P parameter.

D Parameter

The D parameter incorporates the rate of change of the error into the control output. As faster the error changes, as higher the D part output will be.

I am wondering, is the cycle time of the simulator also 0.02 seconds like in the Path Planning Project? There is nothing mentioned about time in the project instructions.

For the D parameter we need to calculate the change of error over elapsed time. In the example python code, this was assumed to be 1, but in the simulator this is certainly not the case.

It supposed to be d_part = (cte_t - cte_t-1)/dt

What is dt in this project or how can we estimate it? Should we use something like gettimeofday() or clock() to estimate this time? Does the simulation report back the elapsed time?

I looked into several forks of fellow students to see how they coped with this problem. I was surprised that in all of the forks I looked into the time wasn’t even considered to calculate the D value of the controller.

Thats maybe because in the course example the delta t time was 1 (1 cycle) and the division by 1 was neglected in the example code. This was mentioned in one of the videos.

But I think it is really necessary to consider the time using the simulation, so I assumed that the simulation runs with a cycle time of 0.02. Also the kinematic equations of the car movement in the simulator need to have a time constant to calculate proper velocity and acceleration for the simulation.

I am not sure if this is the right cycle time and I also do not know, how much cycles pass until the simulation return to the controller application.

After the car seemed to do a good job staying on the road with only P portion, I tried to reduce the over steering by increasing the D parameter. This did a good job, but the car always oscillates along straights, so it still tend to over steer even with a high D value. Therefore I tried to decrease P, this helped on the straights but the car did not manage to take the sharp turns a the end of the track anymore. So I kept the tunings from before, accepting that it might be a compromise between this two behaviors.

The performance looking at the car subjectively is really good. It is keeping the car quite in the middle of the road, even in the sharp corners. Though I do not quite understand in every area of the track what the plots of my cte and pid outputs are telling me, I think I can see some bias of the cte (orange) at the beginning of the track (which is a fairly straight road).

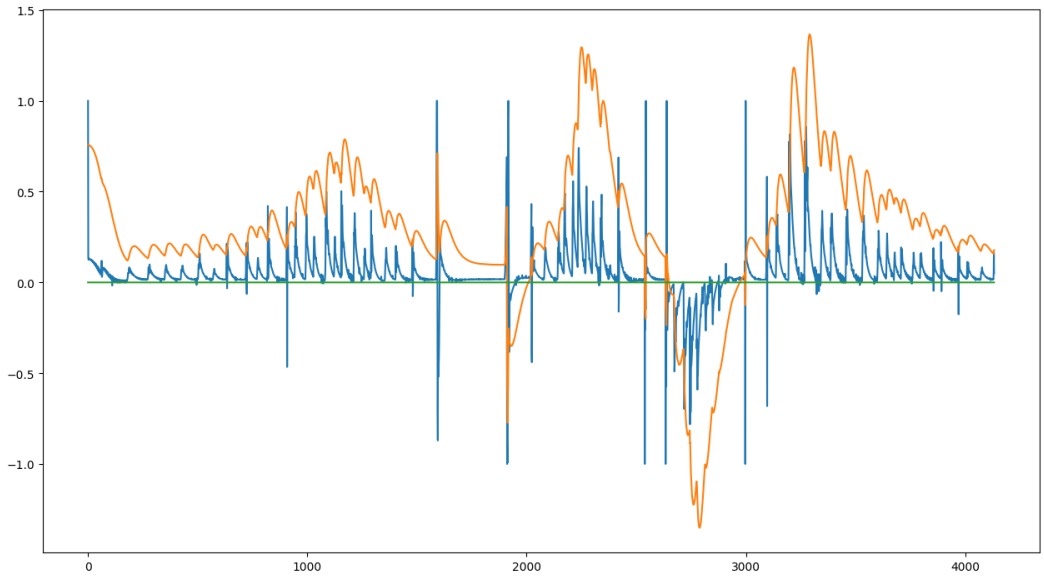

Illustration 10: P 0.18 D 0.0 I 0.0 - speed 20mph, cte (orange), steering angle (blue), zero error (green)

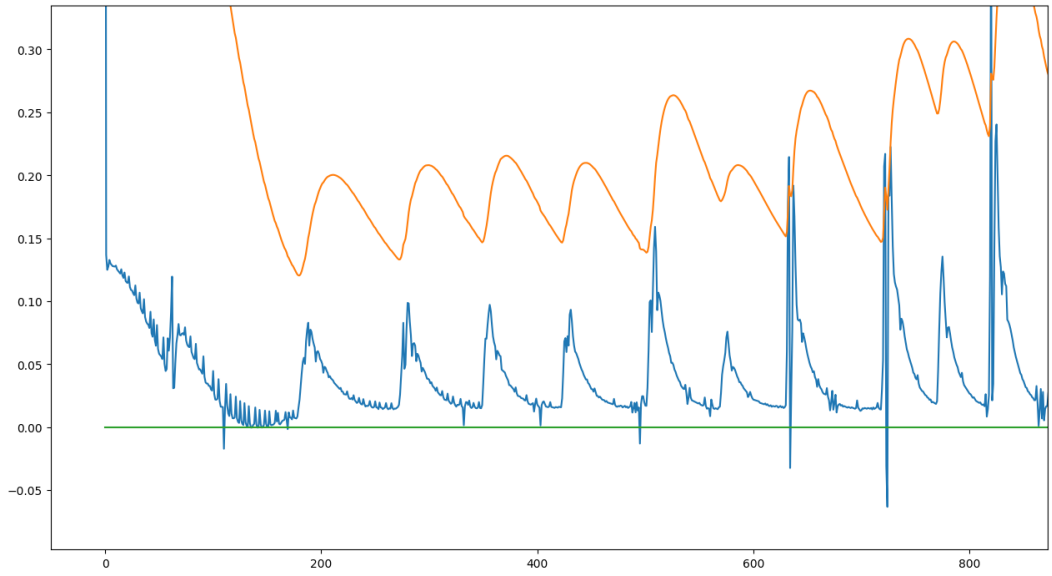

Illustration 11: P 0.18 D 0.0 I 0.0 - speed 20mph, cte (orange), steering angle (blue), zero error (green) - Detail first straight part of the road

I increased the I parameter so that that the bias might get smaller. It now seems to overshoot at the beginning of the track.

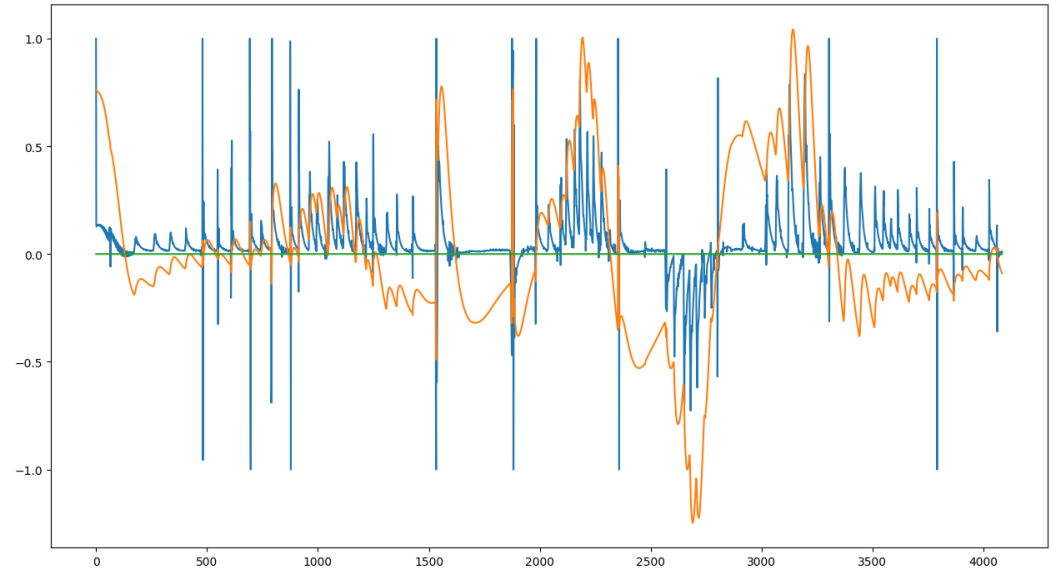

Illustration 12: P 0.18 D 0.02 I 0.01 - speed 20mph, cte (orange), steering angle (blue), zero error (green)

Reducing the I parameter so that it does not overshoot, showed that on the first part of the road, the car is osculating around the CTE, which is nice. I tweaked the the P and D parameter also a little bit. Increased the D Parameter slightly hoping to get rid of the small oscillations around the CTE, but it did not help very much.

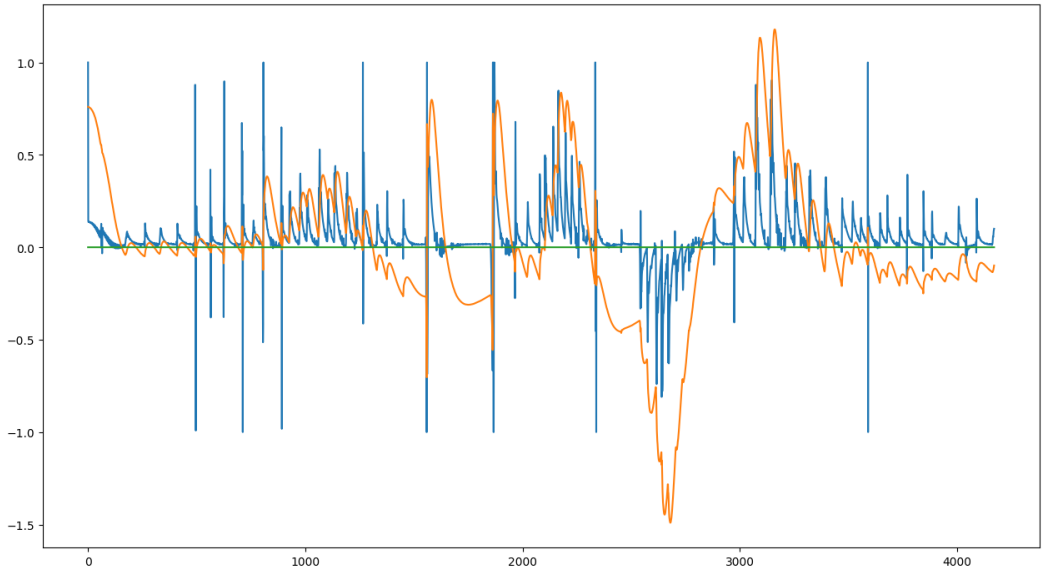

Illustration 13: P 0.185 D 0.025 I 0.005 - speed 20mph, cte (orange), steering angle (blue), zero error (green) - final tune

There sometimes sudden changes in the CTE which causes the D part of the controller to peak out. While the car is driving it can be seen on sudden left/right steerings of the front wheels.

These sudden changed seem to be also in the CTE data sometimes, especially at the bridge part the value seems to jump. Also I do not know if my assumption using delta t of 0.02 are right to calculate the D part of the control output.

If this where a real car this could be dangerous and also could not be executed by the steering system or would damage it. I would want to get rid of these -1 +1 peaks.

Recording a video

For final submission I installed record my Desktop on the server and recorded a video of the run. Having the video on my local machine, the speed of the car seemed quite slow. :) Looking at the laggy VNC Desktop transmission, this seemed much faster. Anyway better to drive safe and slow then fast and dangerous. I do not want to tune on and proceed to get the project done, because I am past due.

https://youtu.be/P4WsAvGLax8

Conclusion:

As I already stated in my course feedback: The course actually entirely blends out how to tune the parameters in a car itself. The Twiddle algorithm is nice, but I can ran this parameter optimization a hundred times in this simulation, but how should we do this in reality in the car / or in simulation where we do not have the proper mechanisms to start/stop/restart the simulation.

In reality:
Do I need a straight and flat test area, where I follow an e.g. straight RTK-GPS path and let Twiddle work as long as the car does not drive me in circles anymore?

Sebastian Thrun talked about sitting in the car for month doing tunings in his introduction video. It would be great, if we could get more insight what methods are used in the car. Real hands-on experience is the most valuable information we could get. Not everyone has the possibility to jump into a car in the desert and start to tune a PID steering controller, so we only can learn form the few ones who did this already.

In Simulation:
As I already mentioned we do not have the proper start/stop/restart mechanisms to start the simulation over and over again to pass optimization runs. Also like in the last path planning project it would be nice to have special cases to test like:
- a straight road to start with,
- circle round left,
- circle round right and some clothoid shaped curves with constantly increasing curvature (like the road are built, at least in good old Europe).

A possibility to restart the simulation from the application would be nice, to trigger multiple optimization runs on the different scenarios. I think running the optimization on different track part would yield in different parameters, so I do not think optimizing parameters over one entire lap would yield in optimal parameters.

I take with me what I learned about the Twiddle optimization in the course, but I not going to implement this for this project submitting it with my hand tuned result of the controller. I this is OK for the depth of the course leaving out A LOT OF control theory behind the scenes.

It could have at least explain the fundamentals of a control system with Input/Output/Feedback:
https://en.wikipedia.org/wiki/Control_system

Pure Pursuit Controller

To follow a trajectory I rather would use the pure pursuit controller:

https://core.ac.uk/download/pdf/4406862.pdf

We used this one in one project our ours controlling a car and it was relatively easy to tune the system to give a good performance. For tuning of the “reactiveness” of the system we only had to set the following points nearer or further away on the trajectory to follow. Of course sharp turns tended to be cut by the vehicle if the following point on the trajectory was too far away.