Hyperparameter Tuning for the Steering Controller

Manual hyperparameter tuning was used exclusively on the PID controller. Using a PID controller for steering input presented an interesting challenge because the external disturbances were fairly random - the curvature in the road did not follow a linear pattern. The common metrics for characterizing a PID controller, like impulse response, damping behavior, offset handling, etc, are not particularly useful when there are not many straight paths in the road.

Tuning P parameter: Sensitivity to deviations in CTE

In tuning the controller, I first considered the proportional term. I believed the controller must be able to react quickly to an upcoming turn, but at the same time it should avoid being hypersensitive to deviations from the center of the road. I noticed that the CTE would max out at ~3.0 assuming the vehicle was on-road, so I figured that the steering input should hit upper and lower bounds (i.e. 1 and -1) when the CTE hit a max. To accomplish this, I obtained the initial *P* value by dividing the steering limit by the max CTE value, and this gave 1.0/3.0 = 0.33. When I ran this configuration (with *I* = 1.0 and *D* = 1.0), I noticed the controller was too sensitive to small deviations in the CTE. From there, I decreased *P* to smaller values until I felt it was appropriately responding to disturbances in the CTE value.

Tuning D parameter: Damping properties

Next, I considered the derivative term, *D*. In testing the proportional term, it was obvious that the system was not damped enough since it would oscillate several terms even on straighter paths. I simply increased *D* by relatively large amounts until the controller was able to properly damp the oscillations. I did hit a point where damping was successful in quickly stabilizing the system, but the controller did not perform so well when the CTE changed quickly like it did in sharp turns. From there, I decreased *D* until the oscillations were tolerable while the system was still performing well through turns. I settled on *D* = 3.2

Now that the system was making it through most of the track, I decided to increase the speed limit, and I quickly saw that more damping was needed as the vehicle was entering turns at a higher velocity (and the CTE would grow much faster in this case).

To make the controller more robust to changes in the speed limit, I introduced a 'fudge factor' that was proportional to the selected speed limit. I arrived at

$$D = 2.8 + (speed\_limit/100.0)$$

This did a fairly good job at making sure the damping was appropriate between ~30mph and ~75mph.

Tuning I parameter: **Correction for gentle curves**

There are no real straight-aways in the test course, so there is not really a chance to see if some kind of steering input 'offset' is necessary to keep the vehicle centered. There are so many turns and they appear so frequently that I decided to tune the *I* parameter to assist the steering input when handling gentle curves.

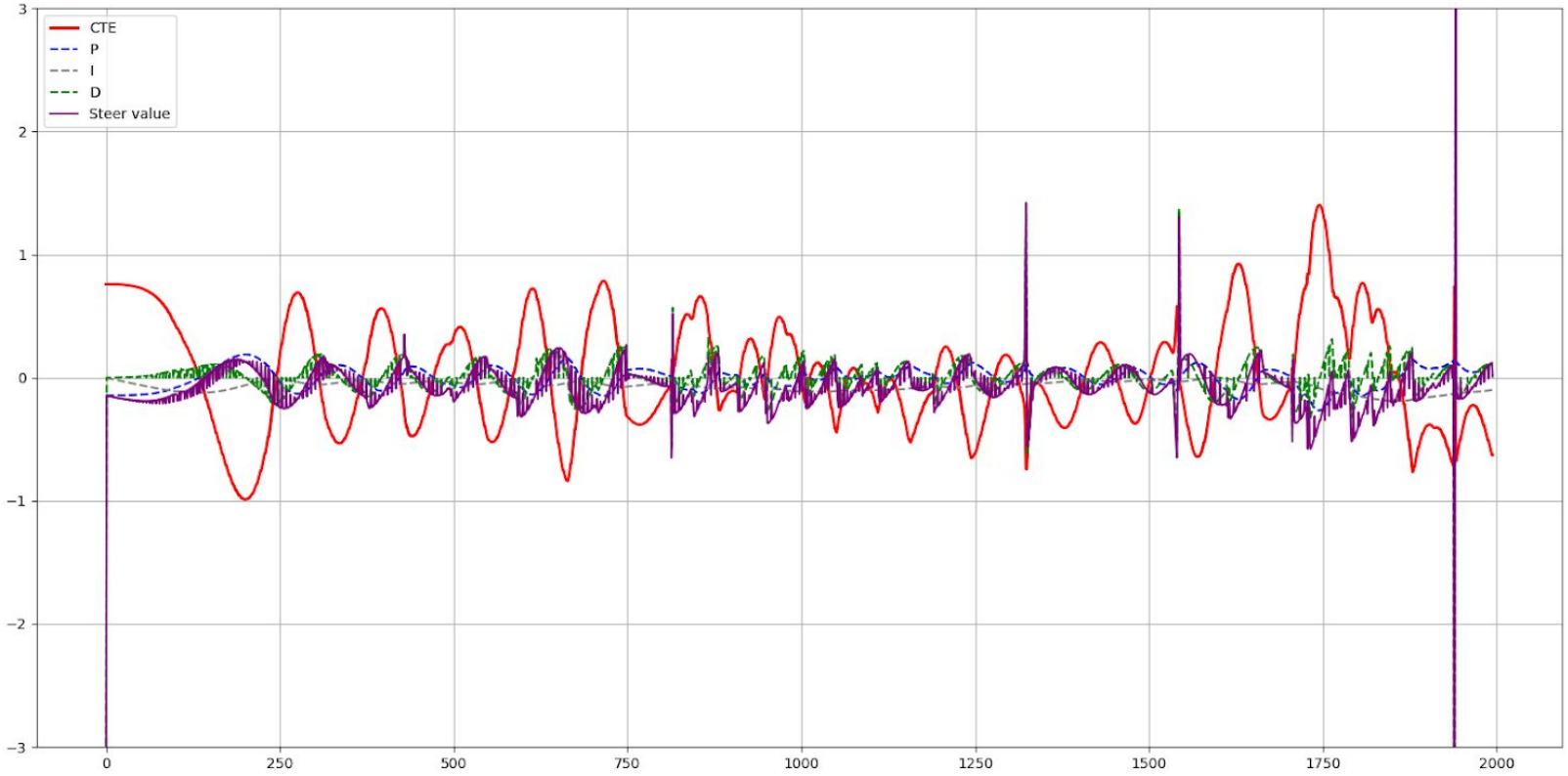
I noticed that most of the gentler curves in the track only required about +/- 0.25 steering input to keep the vehicle centered on the road. I thought of the gentle curve as kind of an 'offset', since to keep centered on a curve with a constant large turning radius the controller must keep the steering wheel slightly turned for a long duration of time. I figured that after about 1000 time steps, the controller should be able to handle a necessary steering input offset of that magnitude (+/- 0.25). This approach requires that the input from the *I* term is capped at a desired usable value, and this also has the added benefit of eliminating integral windup. Since I was thinking in time steps, I had to roughly convert the sum total of the successive CTE values to a distinct number of time steps. Ideally, the CTE would be between -0.2 and 0.2 on average, so I estimated that after 1000 time steps with an error CTE in each time step, the sum total of the CTE up to that point would be equal to 200. To implement this, I included the following login in my controller (i_error is the sum total of CTE):

```
if (i_error >= 200 || i_error <= -200) {
        i_error -= cte; // subtract cte from i_error if it pushes it out of bounds
}
```

Using the fact that the sum total of the error was capped at +/- 200, this put the *I* value to 0.25/200 = 0.00125. From there, I adjusted *I* until the performance was at its best.

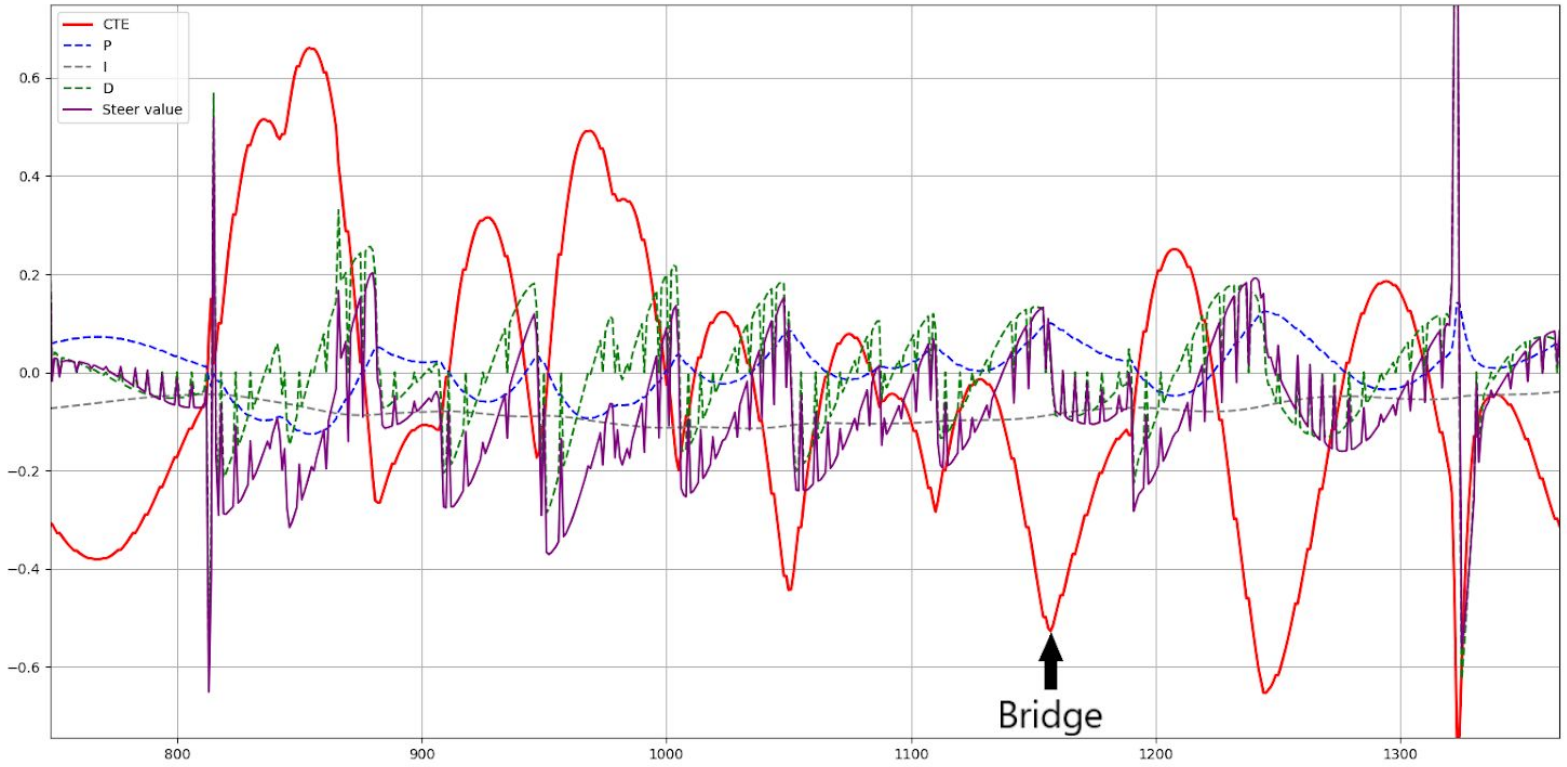
P, I, D: Behavior summary

To analyze how each parameter was affecting the steering input based on the CTE, I plotted the CTE values versus the steering input, and also included each discrete P, I, and D term contribution to the steering input. The end result is that while there are definitely oscillations in the CTE value, it is mostly contained to be < 0.8.



Investigating the behavior when crossing the bridge (the only real straight away) illuminates how each term contributes to the steering input. The bridge is reached at around time stamp 1140, and the there is a gentle left-bending curve leading to up to the

bridge. Inside the gentle curve, from points 820 to 1140, the controller keeps the vehicle on the inside of the turn, evidenced by the value of CTE being mostly positive here. This is good behavior, since it counteracts the vehicle's tendency to veer to the right of the road in a left turn. The *I* term contribution increases to slightly to -0.1 to pull the vehicle away from the left side of the road and back to center. The *D* term contribution seems a bit over-sensitive, but it seems like this is necessary when the curvature of the road is so dynamic.



As the vehicle pulls out of the turn right before the bridge, the CTE is kept to small values of +/0.4. There is an impulse in the CTE value right as the vehicle enters the bridge. I suppose this is meant to simulate sensor noise as the vehicle moves from one road type (asphalt road) to another (bridge with physical curbs on either side). The response to the impulse represents some unwanted behavior since the vehicle begins to swerve from side to side a bit. However, this is the necessary tradeoff if the controller

is capable of responding to sharp turns at high speeds - it must respond fast, but it has no discretion and will respond fast to sensor noise. The same kind of sensor noise impulse happens at ~1800.

Throttle Controller

Ideally, I wanted to introduce a throttle controller that would brake the vehicle as it was entering a turn. I tried using a 'dumb' controller to cut the throttle by 10% whenever the steering input value was above a threshold value, but the behavior was clunky and not very effective. Next, I decided to use a full PID controller for the throttle input.

In order to brake the vehicle as it entered a turn, I introduced an offset to the set point value for the throttle PID controller. Initially, the set point was equal to the difference between the current speed and the speed limit: *speed - speed_limit*. This naturally worked to get the vehicle to the speed limit, but it didn't really introduce any disturbances to the throttle. I figured that the disturbances would have to come from the CTE values, which means that as the magnitude of the CTE increased, the throttle would decrease to more effectively turn the vehicle. The average CTE value is an order of magnitude smaller than the speed limit value, so in the end I multiplied it by 15 and added it to the set point value like so:

$$pid\_throttle.UpdateError(speed - speed\_limit + 15 * cte);$$

This implementation produced the best behavior and I was able to set the speed limit to 60 mph.