PID controller

Daisuke Miyazaki

Summery

My vehicle successfully drives a lap over the whole track without leaving its tires off the roads, rollover nor any crash. Here is my walk through of how I implemented the code and the reasonings.

What Is PID controller?

PID controller is a control software of an interested object. Given the reference trajectory, this controller can navigate an object properly as to follow a given trajectory. *P*, *I* and *D* refer to different mathematical equations and are used for its optimal trajectory to converge to the reference trajectory.

Here is the entire equation of PID controller.

$$\alpha = -\mathrm{tau_p} * cte - tau_i * \Sigma\, cte - tau_d * \frac{d}{dt} cte$$ [1]

Alpha is a steering angle. When a vehicle receives the value, it automatically steers by the amount to get closer to the reference trajectory.

P stands for proportional control, and mathematically it writes as $\mathrm{tau_p} * cte$. As the cross track error or AKA cte to the reference trajectory increases, this value increases its trajectory proportional to the cte.

I stands for integral control, and mathematically it writes as $tau_i * \Sigma\, cte$. If a vehicle converges to a trajectory which sits afar from the reference trajectory by a certain margin, this value comes into play. Combined with the integral cte – in another word the history of the cte from the beginning, this shortens the distance between the current apparent trajectory and the reference trajectory, thereby converging the current trajectory to the reference

[1] In my code, Kp, Ki and Kd express $\mathrm{tau_p}$, $\mathrm{tau_i}$ and $\mathrm{tau_d}$ respectively.

trajectory.

D stands for differential control, and mathematically it writes as $tau_d * \frac{d}{dt} cte$. If my current trajectory is converging to the reference trajectory, it'd be ideal to slowly decay the ratio of the change and smoothly converge to the reference trajectory without overshooting. By taking the derivative of my current trajectory, it decreases the absolute value of alpha as my current trajectory gets closer to the reference trajectory.

Software pipeline

Here is my walk through of how I implemented PID controller for this project.

Firstly, I build a PID instance to run the vehicle around the track.

In this project, the vehicle traverses the track by adjusting its steering angle. Throughout the whole track, an optimal set of 3 values – tau_p, tau_i and tau_d – should not only navigate the vehicle properly but also satisfy smooth driving. So essentially, after building the pipeline to drive the vehicle around the track, the task is to fine tune these hyperparameters. For convenience, I used argV function to manually assign the 3 values when I run the simulator. Afterwards, I assigned 3 values to PID::Init and initialize them which go on in the onMessage loop.

Next, it's time to consider the ctes in the equation.

The first member of the equation is the product of tau_p and cte. Here, cte is the current value of cte.
The second member of the equation is the product of tau_i and integral cte(i_error). Here integral cte is a combination of the previous value of cte and the current value of cte.
The third member of the equation is the product of tau_d and differential cte(d_error). Here differential cte is a difference between the previous value of cte and the current value of cte.

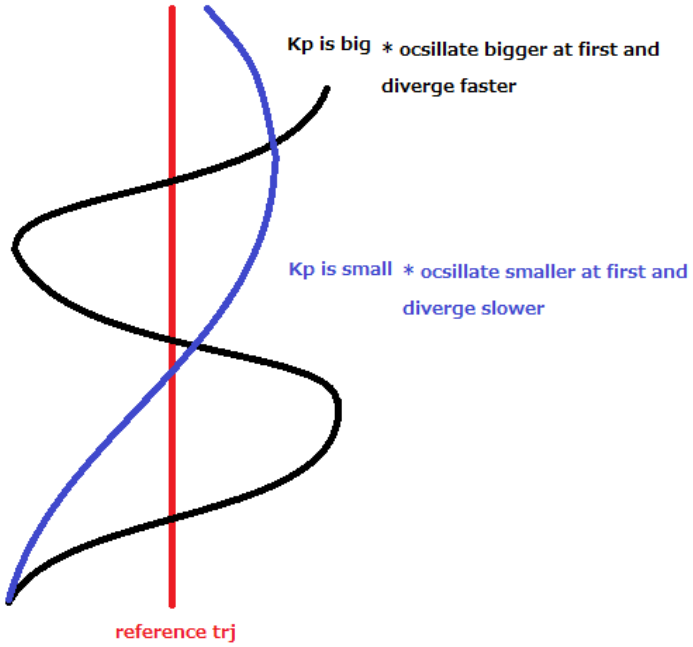
Updating the 3 values, I yielded alpha and steer the vehicle by its value. Its range varies, so I set up the maximum steering value to 1 and the minimum to -1. If the alpha value goes out of the range, my program instead assigns -1 or 1 to it depending on its corresponding positive or negative sign.

Result values of Kp, Ki and Kd

The vehicle starts away from the center of the road. So as the vehicle moves, the current value of cte should decrease to converge to the reference trajectory.

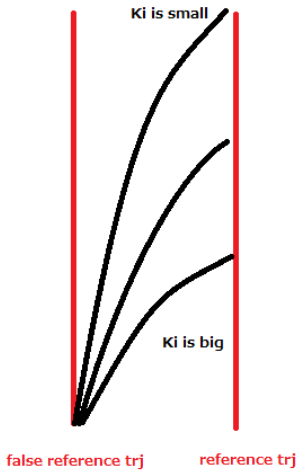
P value

When P value is big, the vehicle oscillates large at first and diverges faster. On the other hand, when P value is small, the vehicle also oscillates small at first and diverges slower.



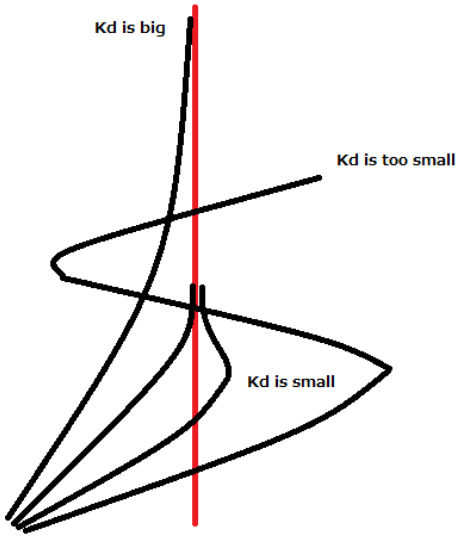
I value

When Ki value is small, my current trajectory gradually converges to the reference trajectory, overshoots a bit, and oscillates around the reference trajectory by a small margin. On the other hand, when Ki value is big, the trajectory rapidly converges to the reference trajectory, overshoots more, and oscillates around the reference trajectory by a big margin. Without I value, the trajectory will converge to a false trajectory when the vehicle has a start steering angle opposite to the direction to the reference trajectory.



D value

Kd value resists to the change of my current trajectory. The bigger Kd is, the less the trajectory oscillates around the reference trajectory. Meanwhile, the smaller Kd is, the more the trajectory oscillates around the reference trajectory. With a certain value of D value, the trajectory converges to the reference trajectory. When the Kd is too small, this D doesn't play a role in alpha and the estimated trajectory will be winding if the original estimated trajectory without I is so.



With this, I obtained the appropriate set of the order of the hyperparameters.

Tuning Hyperparameters: approximate estimate

The goal of this tuning is to find an optimal set of parameters to decrease ***the sum of cte*** as much as possible. This means that the vehicle remains close to the reference trajectory throughout the reference trajectory.

I defined this sum of cte as *total err*, which goes as below:

$$\text{total err} = \frac{\Sigma\ \text{cte}^2}{\text{mid to the end of timestamps}}$$

As it's possible for the set of parameters to have a wide range of local minimum, I needed to estimate the set of values first, and then use Twiddle algorithm to fine tune the set values.

In the lesson from Sebastian about the algorithm, he gave an example set of parameters, which was

Kp 0.2 ***Ki***: 0.004 ***Kd*** 3.0

which corresponds to *Kp,Ki*, and *Kd* values. Then I manually tried a couple sets of values within the same orders and drove the vehicle with them. I ended up obtaining this set of value (in the same order as above)

Kp: 0.31, ***Ki***: 0.0015, ***Kd***: 7.8

With this, I used the twiddle algorithm to fine tune the parameters (explanation in the next section). As a result, I obtained

Kp: 0.149962 ***Ki***: 0.0012375 ***Kd***: 5.655

And with this set of hyperparameters, the vehicle could successfully drive around the whole truck without leaving its tires off the roads, rollover nor any crash.

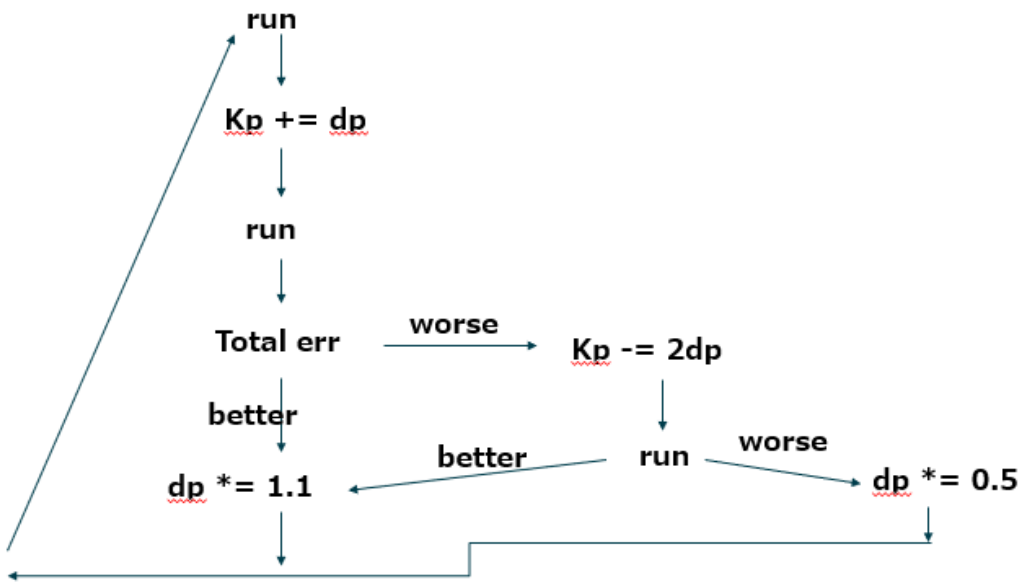
Tuning hyperparameters: Twiddle

OnMessage loop yields a cte value on every loop. So, I created a timestamp which counts 1

per loop. With roughly 1500 timestamps, the vehicle reaches the bridge in the truck.
Due to the mathematical equation of the PID controller, the width of the vehicle oscillation around the reference trajectory decreases over time.

In Twiddle, I change *Kp, Ki* and *Kd* parameters by a small margin- from now on I call this margin as *dp*. This should also be about the same or 1 lower order as *Kp, Ki* and *Kd* so the tuned values remain at about the same order. After changing the values individually, I run the vehicle again within the range of timestamps and see if *total err* decrease.

Ideally, I want to drive the vehicle over the whole truck and fine tune the parameters, though this takes long time to converge to the minimum *total err*. Instead, I halted using Twiddle by 1500 timestamps, as cte there is already small enough to keep going.



Twiddle algorithm

The change of the hyperparameters are either addition or subtraction with *dp*. When I increase *Kp*, for example, by a small margin and *total err* worsens, I instead decrease Kp by 2 times of *dp*(I call the new *Kp* value as *Kp'*). After driving the vehicle with this set of parameters, *total err* either improves or worsens.

In former case, I renew *Kp* parameter with *Kp'*. Then I iterate the exact same order with 1.1 times bigger *dp*, as this converges faster to the local minima.

In latter case, I also renew *Kp* parameters as *Kp'*. But then I iterate the exact same order with 0.5 times smaller *dp*, as the local minima now lay within the range of ± smaller dp.

Over iterating the process above, the set of hyperparameters converges to its local minimum. For comparisons, I keep the best value of *total err.* When all the *dp* values decrease to less than 10 % of its original values – which is also 10 % of its *Kp, Ki* and *Kd* values, I stopped running twiddle by returning false flag. In main.cpp, twiddle only activates when its flag is true.

Conclusion

As a result, I obtained the set of hyperparameters above and the vehicle was successfully able to drive the whole truck without leaving its tires off the roads, rollover nor any crash.

I also attached a video of the vehicle driving around the truck successfully, so please refer to it as confirmation as well.