**PID-Controller**

**Write up** 1st submit:September 13th, Kenta Kumazaki

**1. Goals**

In this project my goal is to design the PID Controller to control the autonomous vehicle that run on  
the Lake Course (same course as the project: Behavior Planning).   
The vehicle should run smoothly and don’t overshoot on curves.

**2. Steps**

The steps of this project are the following:

1. **Find some good Gains**

At first, design the basic PID controller by referring Lesson 12.

And I tried some sets of P gain, I gain and D gain, then find the 2 sets of gains that meet the following each specification:

1. Run smoothly on straight roads. (However, there’s overshoot at curves.)
2. Don’t overshoot on curves. (However, there’s wondering on straight roads.)
3. **Combine good Gains**

It’s difficult to have one optimal set of gains, so I take advantage of each set of gains (a) (b) above.  
To do so, I dynamically change gains according to the following way:

* If the absolute value of Cross Track Error (variable: cte) is smaller than some value, I use the set of   
  gains (a) above. If cte is small, there’s low risk of lane deviation.
* If the absolute value of cte is bigger than some value, I use the set of gains (b) above.

On curves I set higher priority to avoid overshoot than running smoothly.

* Between the range of these 2 points, I use interpolation to gradually change the set of gains.

**3. Submission**

**(1) GitHub**

<https://github.com/kkumazaki/Self-Drivig-Car_Project8_PID-Control.git>

**(2) Directory**

I cloned the basic repository from Udacity <https://github.com/udacity/CarND-Path-Planning-Project>   
and added/modified the following files.

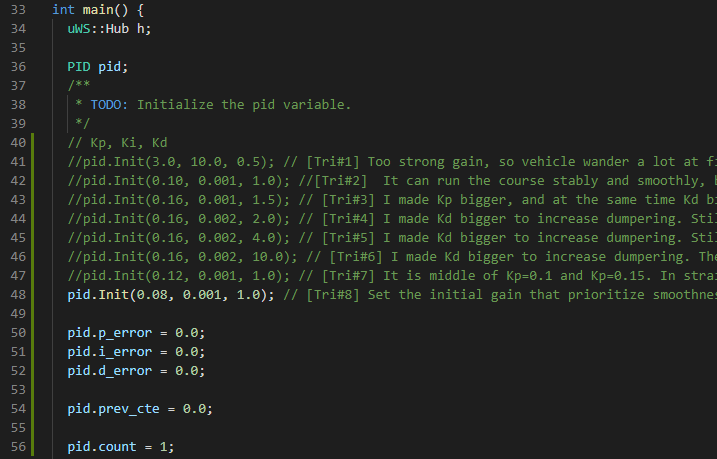
* **Writeup\_of\_Lesson13.pdf**: This file
* **src**
  + **main.cpp**: Main script to calculate steering angle by using PID control and communicate with Simulator.
  + **PID.cpp**: Script used to create functions. I only used Initialization function.
  + **PID.h**:Header file of “PID.cpp”.

**4. Reflection**

**(1) Find some good Gains**

At first, I initialize gains of PID controller. **(A)** I tried a couple of sets of gains here. I will explain it later.

I also initialize other variables. **(B)**

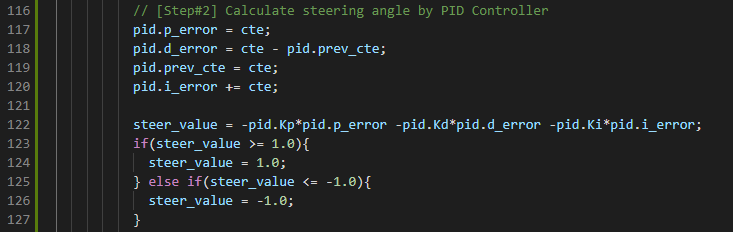


**B**

**A**

Then I designed the PID controller to calculate steering angle according to Lesson 12. **(C)**

I also set the limitation of steering angle from -1.0 to +1.0. **(D)**



**D**

**C**

At first there was no clue, so I randomly set the gains as below.



The first Cross Track Error (variable: cte) is about 0.7, so steering angle became the minimum: -1 and the vehicle moved too fast to left and it overshoots a lot.

I found that P gain and I gain are too large.

I tried a couple of sets of gains, and finally find the 2 sets of gains that meet the following each specification:

**(a) Run smoothly on straight roads.**



This set of gains is good at straight roads as below.

-0.10 degree

-0.76 degree

However, there’s **overshoot** on curves as below.



Step on the lane marker

**(b) Don’t overshoot on curves.**



This set of gains is good on curves as below.



Trace the   
track well

However, there’s wondering on straight roads as below. **(-3.73 degree is large at straight road)**

+0.66 degree

-3.73 degree

I learned the effect each of the P, I, D gain had in my implementation as below:

[P gain]

The bigger P gain is, the faster the error decrease.

However, if it’s too big, the error overshoots and the vehicle wanders and becomes unstable.

[D gain]

The bigger D gain is, the more damping effect to the steering angle. If we’d like to increase P gain to get faster response, at the same time we should make D gain bigger to some amount to avoid overshoot or hunting.

However, if D gain is too big, the error overshoots and the vehicle wanders and becomes unstable.

[I gain]

The bigger I gain is, the less summation of error. Especially it’s necessary to decrease error if the system has systematic bias such as steering drift. It’s also important when I set small P gain around cte is near to zero (explained in the next page) because small P gain cannot decrease summation of error.

However, if I gain is too big, the error overshoots and the vehicle wanders and becomes unstable.

**(2) Combine good Gains**

It’s difficult to have one optimal set of gains, so I take advantage of each set of gains (a), (b) above.

To do so, I dynamically change gains according to the following way:

* If the absolute value of Cross Track Error (variable: cte) is smaller than 0.2 [m],   
  I use the set of **gains (a)**. If cte is small, there’s low risk of lane deviation, so I set higher priority on running smoothly than avoid overshoot.
* If the absolute value of cte is bigger than 1.2 [m], I use the set of **gains (b)**.  
  On curves I set higher priority to avoid overshoot than running smoothly.
* Between the range of these 2 points, I use **interpolation** to gradually change the set of gains.

PID gain

**gain(b)**

cte

0.2 [m]

1.2 [m]

-1.2 [m]

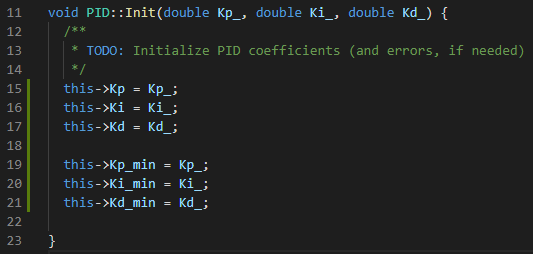
-0.2 [m]

**gain(a)**

**Interpolation**

The codes of this method is following.

First of all in PID.cpp, I set Kp\_min, Ki\_min, Kd\_min in the function init(). They are **gain (a)**.



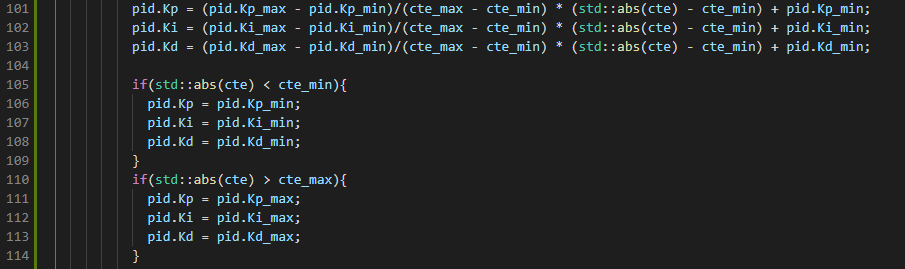
Then I come back to main.cpp.

I set Kp\_max, Ki\_max, Kd\_max according to the **gain (b)**.

I also set the end points of gain (a) and gain (b), which are cte\_min and cte\_max.



Finally I calculate the PID gains by using **interpolation** as below.



**5. Result**

I ran the Simulator with PID controller, and I achieved the Project Rubric as below:

* The vehicle must successfully drive a lap around the track.

No tire may leave the drivable portion of the track surface.   
The car may not pop up onto ledges or roll over any surfaces that would otherwise be considered   
unsafe (if humans were in the vehicle).

It can run on curves without overshoot as below, thanks to the big **gain (b)**.



It can run smoothly on straight roads as below, thanks to the small **gain (a)**.

-0.77 degree

+0.56 degree