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:

(a) Run smoothly on straight roads. (However, there's overshoot at curves.)
(b) Don't overshoot on curves. (However, there's wondering on straight roads.)

(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 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.

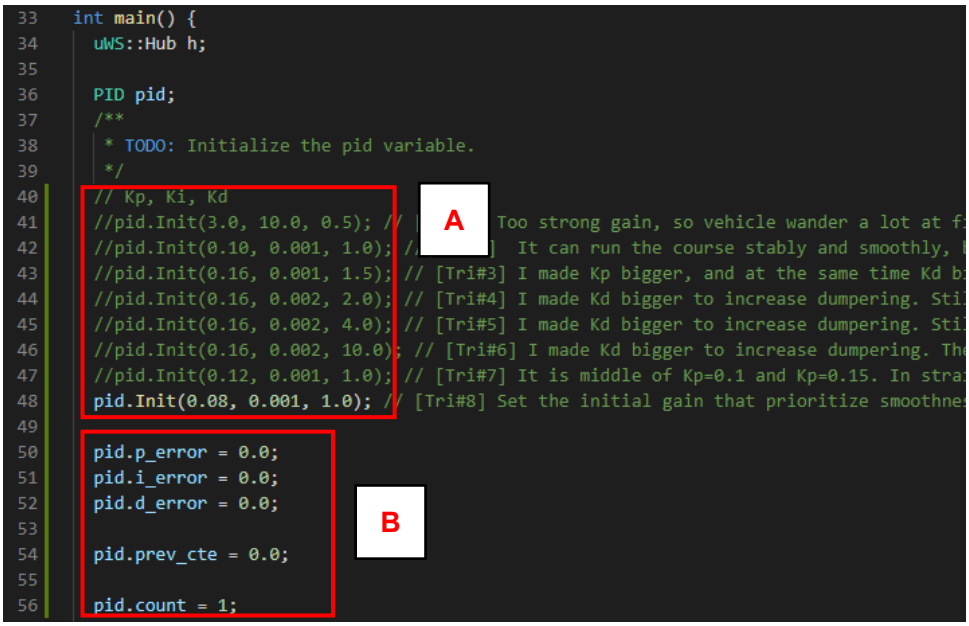
- **Writeup_of_Lesson13.pdf**: This file
- **src**
  - **main.cpp**: Main script to calculate steering angle by using PID control and communicate with Simulator.
  - **PID.cpp**: Script used to create functions. I only used Initialization function.
  - **PID.h**: Header file of "PID.cpp".

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.

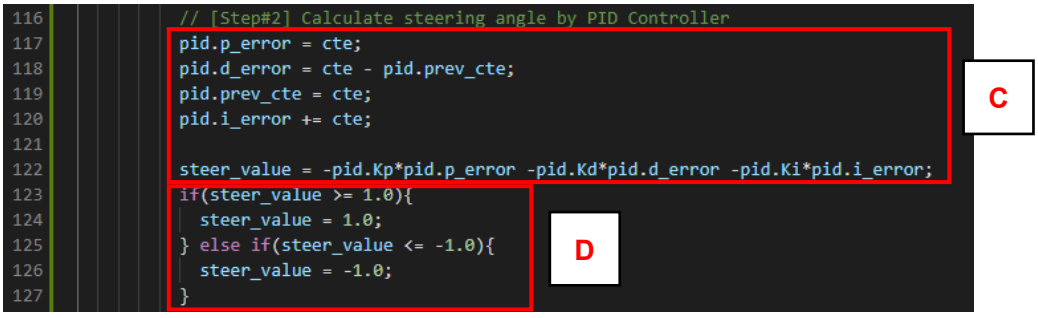
I also initialize other variables. **(B)**




```
int main() {
  uWS::Hub h;

  PID pid;
  /**
   * TODO: Initialize the pid variable.
   */
  // Kp, Ki, Kd
  //pid.Init(3.0, 10.0, 0.5); // Too strong gain, so vehicle wander a lot at f
  //pid.Init(0.10, 0.001, 1.0); // It can run the course stably and smoothly,
  //pid.Init(0.16, 0.001, 1.5); // [Tri#3] I made Kp bigger, and at the same time Kd b
  //pid.Init(0.16, 0.002, 2.0); // [Tri#4] I made Kd bigger to increase dumpering. Sti
  //pid.Init(0.16, 0.002, 4.0); // [Tri#5] I made Kd bigger to increase dumpering. Sti
  //pid.Init(0.16, 0.002, 10.0); // [Tri#6] I made Kd bigger to increase dumpering. The
  //pid.Init(0.12, 0.001, 1.0); // [Tri#7] It is middle of Kp=0.1 and Kp=0.15. In stra
  pid.Init(0.08, 0.001, 1.0); // [Tri#8] Set the initial gain that prioritize smoothne

  pid.p_error = 0.0;
  pid.i_error = 0.0;
  pid.d_error = 0.0;

  pid.prev_cte = 0.0;

  pid.count = 1;
```

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)**




```
          // [Step#2] Calculate steering angle by PID Controller
          pid.p_error = cte;
          pid.d_error = cte - pid.prev_cte;
          pid.prev_cte = cte;
          pid.i_error += cte;

          steer_value = -pid.Kp*pid.p_error -pid.Kd*pid.d_error -pid.Ki*pid.i_error;
          if(steer_value >= 1.0){
            steer_value = 1.0;
          } else if(steer_value <= -1.0){
            steer_value = -1.0;
          }
```

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

```
// Kp, Ki, Kd
//pid.Init(3.0, 10.0, 0.5); // [Tri#1] Too strong gain, so vehicle wander a lot at first!
```

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.

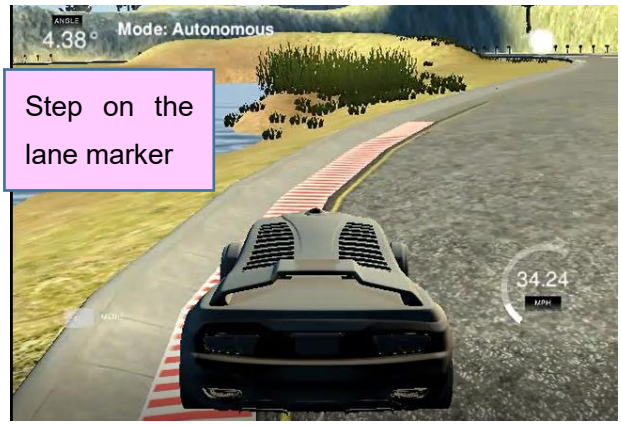
I tried a couple of sets of gains, and finally find the 2 sets of gains that meet the following each specification:

(a) Run smoothly on straight roads.

```
pid.Init(0.08, 0.001, 1.0); // [Tri#8]
```

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

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



(b) Don't overshoot on curves.

```
//pid.Init(0.16, 0.002, 2.0); // [Tri#4]
```

This set of gains is good on curves as below.

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

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.

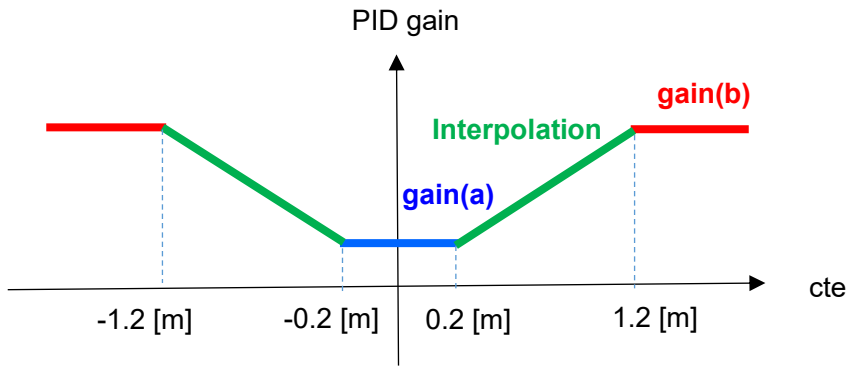
[I gain]
The bigger I gain is, the less summation of error. Especially it's necessary to decrease error if the system has systematic bias such as steering drift. It's also important when I set small P gain around cte is near to zero (explained in the next page) because small P gain cannot decrease summation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



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)**.

```
11  void PID::Init(double Kp_, double Ki_, double Kd_) {
12    /**
13     * TODO: Initialize PID coefficients (and errors, if needed)
14     */
15    this->Kp = Kp_;
16    this->Ki = Ki_;
17    this->Kd = Kd_;
18
19    this->Kp_min = Kp_;
20    this->Ki_min = Ki_;
21    this->Kd_min = Kd_;
22
23  }
```

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.

```
92              pid.Kp_max = 0.16;
93              pid.Ki_max = 0.002;
94              pid.Kd_max = 2.0;
95
96              // Change the gain proportionally according to the deviation: cte.
97              // If cte is smaller than 0.2[m], min gain. If cte is bigger than 1.2[m], max gain.
98              double cte_min = 0.2;
99              double cte_max = 1.2;
```

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

```
101             pid.Kp = (pid.Kp_max - pid.Kp_min)/(cte_max - cte_min) * (std::abs(cte) - cte_min) + pid.Kp_min;
102             pid.Ki = (pid.Ki_max - pid.Ki_min)/(cte_max - cte_min) * (std::abs(cte) - cte_min) + pid.Ki_min;
103             pid.Kd = (pid.Kd_max - pid.Kd_min)/(cte_max - cte_min) * (std::abs(cte) - cte_min) + pid.Kd_min;
104
105             if(std::abs(cte) < cte_min){
106               pid.Kp = pid.Kp_min;
107               pid.Ki = pid.Ki_min;
108               pid.Kd = pid.Kd_min;
109             }
110             if(std::abs(cte) > cte_max){
111               pid.Kp = pid.Kp_max;
112               pid.Ki = pid.Ki_max;
113               pid.Kd = pid.Kd_max;
114             }
```

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)**.