**Summary of the P I D control components**

As the name indicates, PID algorithm consists of three basic components including proportional (P), integral (I) and derivative (D). In the PID model, the P accounts for present values of the error. The control output will be proportional to the error. D accounts for possible future trends of the error, based on its current rate of change. The derivative response is proportional to the rate of change of the process variable. If the process variable is increasing too fast, the derivative component causes the strength of output to decrease to reduce the overshoot. I accounts for the sum of the past of the error. Small errors over long term will cause the integral component to increase slowly over time unless the error is zero.

I tested the (P, I, D coefficients) by assigning different values to them in the simulator. The result is summarized as below.

**Setting P only**

P. I. D = (1, 0, 0), the car was running OK at the beginning, but swung a little bit. In every swing the error was becoming larger and larger and finally the turn overshot and the car ran off the road.

P. I. D = (0.1, 0, 0) Similar to the behavior when PID is (1, 0, 0)

P. I. D = (10, 0, 0) the car swung drastically at the beginning, and run off the road very soon.

The result indicate that the model is lacking the D and I control component, which could prevent the overshot and the error getting larger and larger gradually.

**Setting P and D**

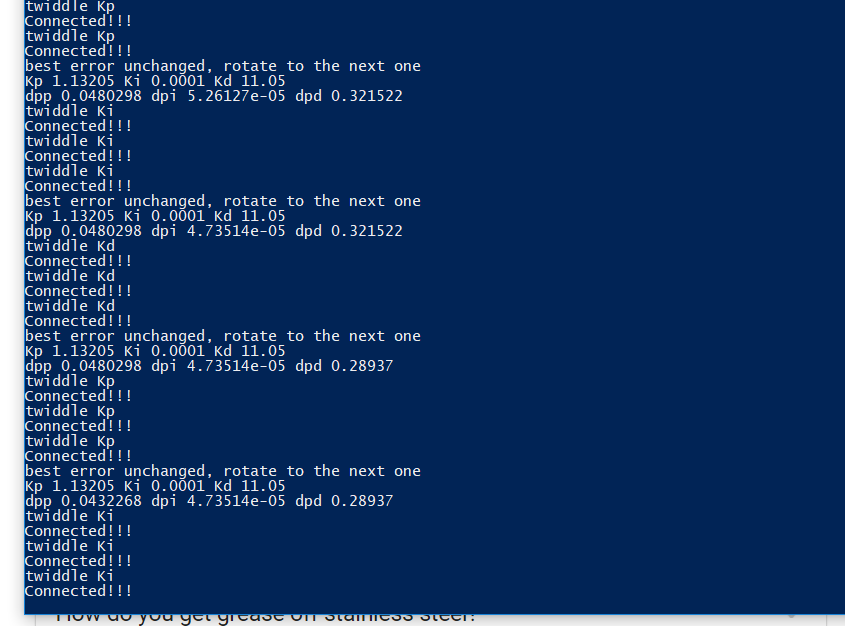
The component D was assigned after the P, because the D is responsible for the short-term overshot problem, while I will play more role in the long-term run. The P was set to 1 at beginning.

P. I. D = (1, 0, 1) the car ran better than without D component at the beginning, but swung still happened finally, and the car finally run off the road. Maybe the D component should be adjusted to higher number to prevent the overshot.

P. I. D = (1, 0, 10) the car was running well on the road, and finish the first lap easily. Sometimes some sudden small turning could still be observed, the basically the overshot was controlled very well. So the (1, 0, 10) should be a combination I can use to twiddle the model. The component I, apparently, should be a small number which will play the role when the error accumulation is large in a long term.

**Twiddle the model**

Next, I use the twiddle process to tune the final hyperparameters PID. My twiddle process includes a “single\_twiddle” function which tune the single parameter, and a “twiddle” function which rotate the turn of the three parameters. A rotor was set in the twiddle function in order to adjust the three coefficients one by one. In every cycle of twiddle, the car run in the simulator for 2000 steps, and the error is recorded. Once a smaller error was achieved, the best error would be changed and the coefficient would be adjusted. The initial value of PID feed into the twiddle is (1, 0, 10), while the dpp, dpi and dpd feed into the twiddle is (0.05, 0.0001, 0.5). After around 45 min of twiddling process, the final coefficients achieved was (1.13205, 0.0001, 11.05) for P, I and D as the print screen shows. I used these final values for the PID model, and disabled the twiddle function, but they could still be found in the main.cpp and PID.cpp



void PID::twiddle(){

step\_num = 0;

p\_error = 0;

i\_error = 0;

d\_error = 0;

total\_error = 0;

if (err < best\_err && (switch\_I == true || switch\_II == true)){

best\_err = err;

std::cout << "The best error is " << best\_err << std::endl;

switch(rotor){

case 0: {dpp \*= 1.1;

std::cout << "The dpp \* 1.1" << std::endl;

break;

}

case 1: {dpi \*= 1.1;

std::cout << "The dpi \* 1.1" << std::endl;

break;

}

case 2: {dpd \*= 1.1;

std::cout << "The dpd \* 1.1" << std::endl;

break;

}

}

std::cout << "Kp " << Kp << " Ki " << Ki << " Kd " << Kd <<std::endl;

std::cout << "dpp " << dpp << " dpi " << dpi << " dpd " << dpd <<std::endl;

rotor = (rotor + 1)%3;

switch\_I = false;

switch\_II = false;

}

else if(switch\_I == false && switch\_II == false){

rotor = (rotor + 1)%3;

std::cout << "best error unchanged, rotate to the next one" << std::endl;

std::cout << "Kp " << Kp << " Ki " << Ki << " Kd " << Kd <<std::endl;

std::cout << "dpp " << dpp << " dpi " << dpi << " dpd " << dpd <<std::endl;

}

//std::cout<< "rotor " << rotor << " switch I " << switch\_I << " switch\_II " << switch\_II << std::endl;

switch(rotor){

case 0: {single\_twiddle(Kp, dpp);

std::cout << "twiddle Kp" << std::endl;

break;

}

case 1: {single\_twiddle(Ki, dpi);

std::cout << "twiddle Ki" << std::endl;

break;

}

case 2: {single\_twiddle(Kd, dpd);

std::cout << "twiddle Kd" << std::endl;

break;

}

}

}

void PID::single\_twiddle(double &K, double &dp){

if (switch\_I == false && switch\_II == false){

K += dp;

switch\_I = true;

}

else{

if (switch\_I == true && switch\_II ==false){

K -= dp\*2;

switch\_II = true;

}

else{

K += dp;

dp \*= 0.9;

switch\_I = false;

switch\_II = false;

}

}

}