

Udacity SDC Term 2 project 4.

The design of PID is basically same as what was being described in the online class material. I used the array `K[3]` to store my P, I, and D components. The P (proportional) component is coming directly from cte. The D (derivative) component is basically the delta of cte. The calculation of D is simply the current cte subtract the previous cte. And the I (Integral) is sum of all cte since time zero. To smooth out the derivative noise, I have implemented a low pass filter to smooth out the derivative. I used the std vector to store the previous D component and take the average. Here is the implementation detail

```
void PID::UpdateError(double cte) {
    double diff = cte - error[0];

    lowpass.push_back(diff);

    if (lowpass.size() > 3){
        lowpass.erase(lowpass.begin());
    }

    error[1] = std::accumulate( lowpass.begin(), lowpass.end(),
0.0)/lowpass.size();
    error[0] = cte;
    error[2] += cte;

    std::cout<<"K value: "<<K[0]<<" : "<<K[1]<<" : "<<K[2]<<std::endl;

    if ( is_twiddle ){
        twiddle();
    }
}
```

I basically keep a buffer of 3 and take the average of it for the D component. Standard library vector and accumulate functions are used to get the average. For the P component, I assign cte directly into my `error[0]` variable. And for the I component, I add cte into `error[2]`.

I did not try to fine tune the hyper parameters for the PID coefficient. I use twiddle algorithm to fine out the PID coefficient. I used 0 for the default values in the `init()` function for my PID coefficient. When the default values are used, twiddle algorithms will automatically trigger. The twiddle algorithms collect a data samples of 1000. Algorithms is simply and work exactly the way how the online class material described. However, to speed up thing a little, I did not use square errors. I do double square (error to the power of fourth). Here is the code that do that:

```
double square = error[0] * error[0];
square *= square; // square it again to speed up the computation a
```

```

little
    err_squ += square;

    // Let's see if we can speed up the twiddle process by checking if
    err_squ is already larger than best_err
    if (state == 1 && err_squ > best_err){
        K[index] -= 2*dp[index];
        state = 2;
        iteration = 0;
        error[2] = 0;
        return;
    }
    else if (state == 2 && err_squ > best_err){
        K[index] += dp[index];
        dp[index] *=0.9;
        index = (index+1) % 3;
        state = 0;
        iteration = 0;
        error[2] = 0;
        return;
    }
}

```

So if the cte input is 4, and I take it to the power of four, it will be 64, This will speed up the rate of err_squ increase its value and we can discover earlier that it is larger than the best error.

After using twiddle, I found the PID coefficient to be P for 0.137611, I = 0 and D = 2.89388. Since I = 0, I did not bother implement integral wind-up.

When I executed the twiddle algorithms, all PID coefficient are initialize to zero. And the car can only go straight. The first parameter P then initialize to one from the twiddle algorithms, the car start running like a "S" shape. So if cte is some value and P is some coefficient, and the output determine how much steering is needed, then the P value is simply implying the ratio of the cte to the steering angle. As I observe the training process of the twiddle algorithm from the simulator, I found that "D" has to increase to a particular value for the car to run smoothly. This coefficient is the ratio of the difference of the cte value to the steering. This is also the derivative term of the PID control. And as I think about it more carefully, it made sense that if cte remain no change, my "D" component will be zero and that mean I am completely depended on the "P" component. This make a lot of sense that if I have to steering at a particular angle because the road is curving at a particular angle, I do want my car to preserve at some angle for some time. If the car is moving straight, cte will become smaller over time until my steering overshoot. And I do want to reduce my steering angle as I am getting a smaller cte. For the "I" coefficient, the way I think about it is if my steering is off alignment, then I want to steering to the opposite direction little more over time.