```
pid.Init(0.3, 0.0, 20.0);

pid.UpdateError(cte);
steer_value = pid.TotalError();
steer_value = std::min(std::max(steer_value, -1.0), 1.0); //normalize the steer_value in
range [-1,1]

msgJson["throttle"] = 0.3;
```

round 1: simplest PID w/o fine tuning, the car drives. And oscilate faster and faster.

```
pid.Init(0.3,0.0,20.0);
throttlePid.Init(1.0,0.0001,0.2);


          steer_value = pid.TotalError(cte);
          steer_value = std::min(std::max(steer_value, -1.0), 1.0); //normalize the
steer_value in range [-1,1]

          json msgJson;
          msgJson["steering_angle"] = steer_value;

          throttle_value = throttlePid.TotalError(cte);
          double throttle = 1 - fmin(1, fabs(throttle_value)); // normalize throttle to
[0, 1]

          if(speed > 40){
              msgJson["throttle"] = throttle - 0.1;
          } else{
              msgJson["throttle"] = throttle;
          }

void PID::UpdateError(double cte) {

    this -> p_error = cte;
    this -> i_error += cte;
    this -> d_error = cte - p_error;
}

double PID::TotalError(double cte) {

    this->UpdateError(cte);
    return -1.0 * (Kp * p_error + Ki * i_error + Kd * d_error);

}
```

round 2: this is my 2nd trial, throttle is also controlled with PID regulator.   i.e. vehicle speed is correlated to cte input.

```
// update Steering angle error function.
steerPid.UpdateError(cte);
// calculate PID controller for steering angle
steer_value = steerPid.GetValue();
// Normalization
steer_value = std::min(std::max(steer_value, -1.0), 1.0); //normalize the steer_value in
range [-1,1]

// link steer_value to target speed, so any veering will leads to slower speed.
double target_speed = 60.0*(1.-std::fabs(steer_value))+20;
// calculate the delta with respect to target speed and update the error function
throttlePid.UpdateError(speed - target_speed);
// calcualte PID controller for throttle
```

```
throttle_value = throttlePid.GetValue();
// Normalization
throttle_value = std::min(std::max(throttle_value, -1.0), 1.0); //normalize the throttle
in range [-1,1]

void PID::UpdateError(double cte)
{
    // build error update function, meanwhile to ensure correct initial d_error
    if (!has_prev_cte_) {
        prev_cte_ = cte;
        has_prev_cte_ = true;
    }

    this -> p_error = cte;
    this -> i_error += cte;
    this -> d_error = cte - prev_cte_;
   ...
}
```

in my third trial, I added the target speed, so the PID will control it like we did for CTE. And the driving performance is further improved.
Meanwhile I add on a refined d_error initial term.

This is my final model, base on which to tune the PID para. Separately.

As the throttle and steering angle is already decoupled.
It did not bother much to tune the throttle itself, I left the I In 0, and tune the P,D controllers instead.

The tricky part is the PID setup for steering.

Here are the steps I used:

1. Set all gains to 0; -> the car drives straight forward.

2. set the initial P gain at 5; -> the car oscillates in the high frequency.

3. tune the P gain until the car oscillates steadly;

4. set the initial D gain to 10; -> the car run out a whole lap in an damped oscillation.

5. tune the D gain until the car drives fast while curving w/o too much vibrations.

6. Iterate step 3 and 5, to set the last stable values.

7. Increase the I gain a bit with a small value, so the oscillation can reach my expectations in visual sense. Here I have to balance between a quicker response and oscillations of overshoot.

I found that a good model is generally a good base to tune the parameters. Whereas not to set up the parameters without a good model. i.e. model contributes more in this case. Thus in quite a wider numerical ranges my PID controllers can work.

In my last attempt, I added the twiddle module, but not focus on too much efforts.
I have some issue with my twiddle model and only part of it works.
But still it showed me some local optimal values into 7-digit decimal.
Actually I did to pay much loads of time into it since the manual method works better.

This is my issue in the discussion forum, in case of a galance:
https://discussions.udacity.com/t/a-better-tuning-process-flow/361045/6

I hope I clearly stated how P,I,D each component works in above 7 steps process so to fulfill the rubric requirements.