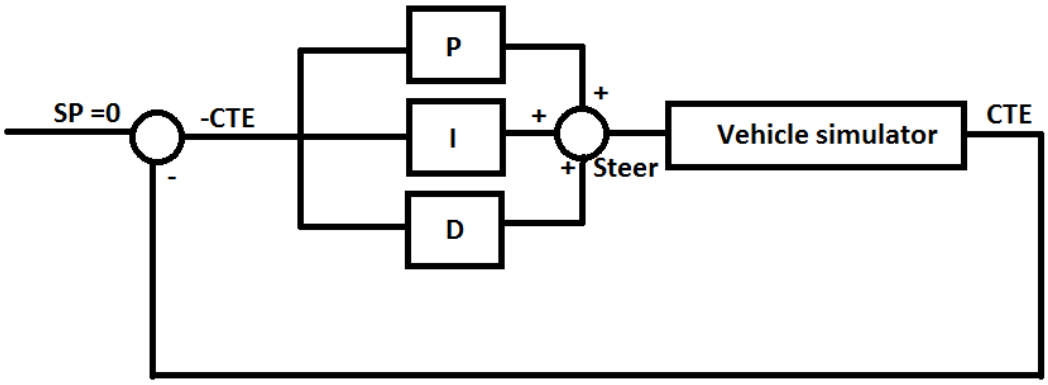
PID Project

The project implements a PID controller and an algorithm for tuning the Kp, Ki and Kd parameters. For this reason, this write-up is splitted in two chapters: **PID Implementation** and **Tuning**.

PID Implementation

The first thing I did was to map the PID block diagram to our project resulting in the diagram bellow:



PID Controller for self driving car

The set point (SP) is 0, because the set point is represented by the CTE(Cross Track Error) when driving exactly along the desired trajectory.

Usually a conducted process outputs a measurable variable from where the error is computed but here the vehicle simulator outputs directly the error (CTE).

The output of the PID is the control variable, in our case the steer value. This will be the input to the process.

In our case, the vehicle simulator runs the process.

The full PID controller is implemented in PID.cpp, in function *double PID::RunPID(double cte)*.

Here I considered each P, I and D component separate. The Proportional component is represented by the –CTE multiplied by the parameter Kp. The P component offers a simple way to steer the car but in a continuous oscillation. The Derivative component is represented by the negative difference of the current CTE and the previous iteration CTE multiplied by the Kd component. The Integral component is an error cumulative element and it is represented by the negative sum of the previous CTE values multiplied by its own Ki parameter.

The sum of the output of each component is the output of the PID controller.

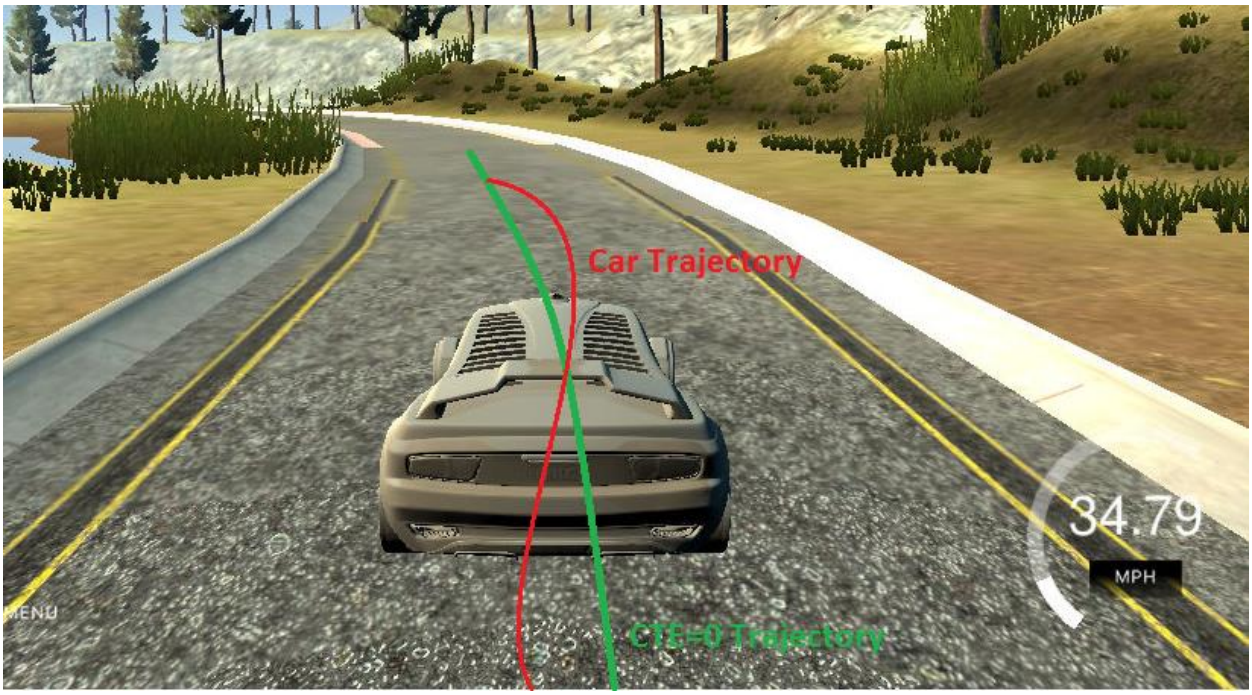
Tuning

For the parameters tuning I constructed a simple algorithm to determine Kp, Ki and Kd parameter. The observations that lead to the development of the tuning algo started from the following:

- If I set a big Kp I see huge amplitude oscillations of the vehicle right from the start.
- If I set a small Kd I see fast oscillations of the vehicle right from the start.
- Now if I set the opposite for Kp and Kd I have problems steering later on the track when tight curves arrive.
- Ki seems not to matter so much. Just leave it to zero.

So now I can choose some values for Kp and Kd that seems decent for the starting of the lap (somehow the straight line from the first portion of the track). These value are Kp = 1 and Kd = 2, leaving Ki = 0;

With these parameters, the vehicle trajectory resembles a clear sin wave around an imaginary CTE=0 line.



Knowing that Kp parameter is responsible for the amplitude of the sin wave and the Kd for the frequency I implemented the function for auto-adjusting down these 2 parameters in:

void adjustP(double cte);

void adjustD(double cte);

Function *adjustP()* reduces Kp proportional with the CTE and *adjustD()* just reduces the Kd with a predefined value when the sin wave completes too fast. The values used as thresholds in these functions are determined empirical, the purpose of this auto-adjustment being in obtaining a smooth sin wave with low amplitude and small frequency.

The auto-adjustment is done online by running the simulator and the software with the macro PARAM_ADJ defined as 1 in main.cpp. Once I get the desired values for Kp and Kd I define the macro PARAM_ADJ as 0 and re-compile the software. If desired you can leave the macro PARAM_ADJ as 1 and the software will stop auto-adjusting once the vehicle had done one complete lap (1400 lap increments) and run with Kp and Kd adjusted.

The delivered software was compiled without auto-adjustment enabled (#define PARAM_ADJ 0) and with the Kp, Ki and Kd already set to the values obtain before with auto-adjustment.

The final values for Kp, Ki and Kd after online adjustment are:

Kp = 0.179

Ki = 0

Kd = 1.5