# 1 Files

Please make sure that you use the Updated CMaker file as I added a new file to the project in order to accommodate the twiddler functionality.

Extra files:

```
CMakeLists.txt (updated only)
Twiddle.cpp
Twiddle.h
```

I made the empirical tuning of the PID controller by way of with trial and error and with the help of the included Twiddle class.

# 2 PID Control Reflection

A PID controller may be a good choice when the control specifications are well defined and the magnitude and frequency of disturbances are known:

- The required time for recovery
- Allowable overshoot
- Required stability

The main advantage of this method is its relative simplicity and straight forward applicability in programmable controllers.
I've implemented a simple speed adjustment functionality in order to make the car drive faster when safe:

```
if (steer_value < abs(0.3) && cte < 0.2) {throttle +=
0.08; }
```

```
if (steer_value > abs(0.5) || cte > 0.6) {throttle -=
0.4; }
if (throttle < 0.1 ) {throttle = 0.1;}
```

## Components

```
Pout = Kp*e(t) + Kd*e(t) + Ki*e(t)
```

***Kp - proportional gain*:**

The control is dependent only on the present error term (cte).

***Ki - integral gain*:**

It represets the past of the error term, as it summarizes the error gains collected along the way.

***Kd - derivative gain*:**

It represents the *future* of the system in a sense that it captures the only the most recent change of the error term predicting where the process is heading (though sensitive to high frequency noise).

## My Solution

The aim is to find a combination of gains that renders the vehicle's response appropriate, thus avoid overshoots, underdamping and overdamping. My solution is a control characterized by the relative overweight of derivative term, as I found it most appropriate after a few iterations. It kicks in all the more when the cte deteriorates quickly, i.e. in a turn especially at high speed.
I made my own twiddler class in order to tune the values programmatically, though it provided only limited assistance. The

solution is a simplification of the functionality presented in class:

```
if (avgCte < prevAvgCte)
    {
        K += delta;
    } else {
        delta = -delta/2.0;
        K += delta;
    }
    prevAvgCte = avgCte;
```

It will converge slower if it misses the right direction at first, but the code is cleaner, hence easier to verify.

The method has its clear shortcomings in general:

- It will be caught up in local minimums
- The K's are interdependent, many re-iterations are necessary
- It looks only at the deterioration or improvement of the cumulative error, and it can drive the car in the ditch easily (as it did...)

## Limitations if PID control

The PID model is blind in a sense as it does not know where the vehicle is heading or what its momentum vector is so it cannot choose a more favorable set of gains. It is a feedback closed loop control system with constant parameters without taking the real process into account: it is always lagging (as opposed to on-time or upfront), reactive, linear and prone to oscillation.

## Possible Improvements:

Cascade control with an inner and an outer loop would be an easy way. The outer loop would set the actual set-point of the physical system, for

example pulling the set trajectory towards the inner side of the curvature of the road. The inner loop would be the actual PID controller controlling the car. (new parameters required).

The system could be analysed with one of the known PID control loop verification methods, for controllability, robustness, and limitations. (Ziegler–Nichols method, Laplace form analysis, etc.)

Gain scheduling could be used to pick the most favorable gains for different situations (e.g. highway or a inner city roads).

All of the above would though require the inclusion of some new observable variables in the control model.

PID control does not take vehicle model into account in any way, at least the followings would help a great deal (a such system would hardly be a mere PID control, though). Here are some parameters to include in the model to start with:

- The speed of the vehicle
- Yaw rate
- Steering angle
- Mass of the vehicle (or rather a mass distribution)
- target trajectory