PID CONTROLLER PROJECT

15-04-2020

OVERVIEW

1. Project Description

In this project we will implement a PID Controller in C++ to maneuver the vehicle around the lake race track.

2. Project Scope

- The car will try to be in lane.
- No tire will leave the drivable portion of the track
- The car will not pop up onto ledges or roll over any surfaces that would otherwise be considered unsafe (if humans were in the vehicle)

3. High-Level Requirements

The new system must include the following:

- cmake >= 3.5
- All OSes: [click here for installation instructions] (https://cmake.org/install/)
- make >= 4.1
- Linux: make is installed by default on most Linux distros
- Mac: [install Xcode command line tools to get make] (https://developer.apple.com/xcode/features/)
- Windows: [Click here for installation instructions] (http://gnuwin32.sourceforge.net/packages/make.htm)
- Gcc / g++ >= 5.4
- Linux: gcc / g++ is installed by default on most Linux distros
- Mac: same deal as make - [install Xcode command line tools] ((https://developer.apple.com/xcode/features/)
- Windows: recommend using [MinGW] (http://www.mingw.org/)
- [uWebSockets] (https://github.com/uWebSockets/uWebSockets)
- Run either `install-mac.sh` or `install-ubuntu.sh`.
- If you install from source, checkout to commit `e94b6e1`, i.e.
- git clone https://github.com/uWebSockets/uWebSockets
- cd uWebSockets
- git checkout e94b6e1

- To setup the Environment on Windows - here.
- To setup the Environment on Linux - here.
- To setup the Environment on Mac - here.

4. Implementation Strategy

4.1 P Controller **main.cpp**

P Controller stands for 'Proportional Controller'. This helps us set steering angle proportional to CTE (cross track error) by some factor of tau.

$$Steering\ angle = -\tau_p * CTE$$

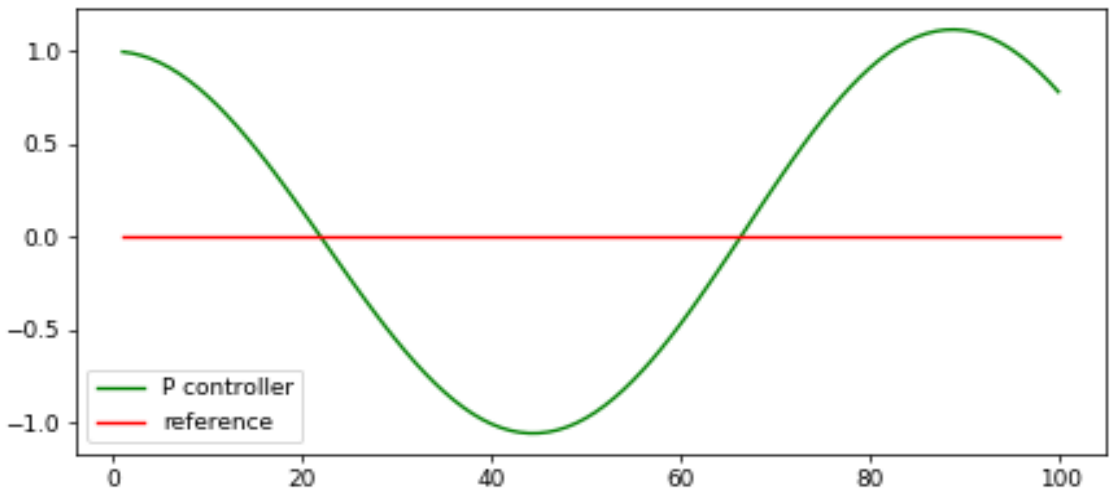
In other words, if a car is moving forward, the proportional controller sets the steering to a value that causes it to steer near the lane center i.e. (CTE). The negative sign indicates that it needs to steer towards the lane center, when it starts to move in opposite direction.

Ex - Steer left when car moved right from the lane center, and right when car moved left from the center.

To visually describe, car will be oscillating around the lane center, and

Drawback:

1. The car overshoots every time.



Output: Refer to the Video: **P_Controller.mkv** in Results folder

4.2 PD Controller **main.cpp**

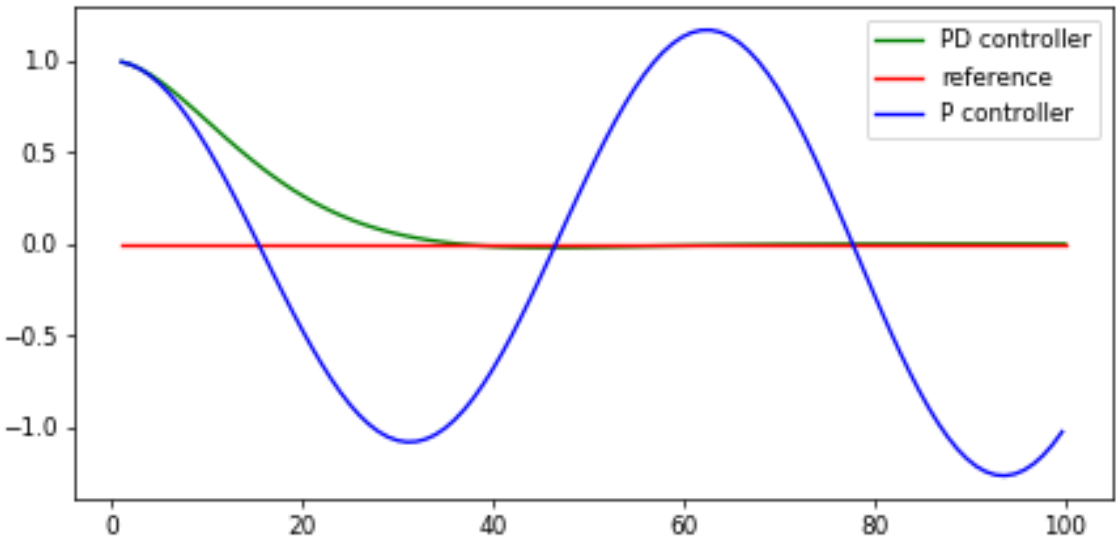
Here, D stands for 'Derivative'. In PD controller, the Steering angle also depends on the temporal derivative of Cross Track Error (CTE). It helps in counter steering the vehicle, when it starts to overshoot.

What this means is that, when the car approaches to the lane center, the D controller counteracts and steers the vehicle away from the lane center, causing the vehicle to approach the lane center gracefully and smoothly.

$$Steering\ Angle = -\tau_p * CTE - \tau_D * \Delta CTE$$

Drawback:

1. Both P and D cannot solve the systematic bias issue. In this case, if there is any bias in the system, the PD controller produces a very large Cross Track Error, and never converges to the lane center.



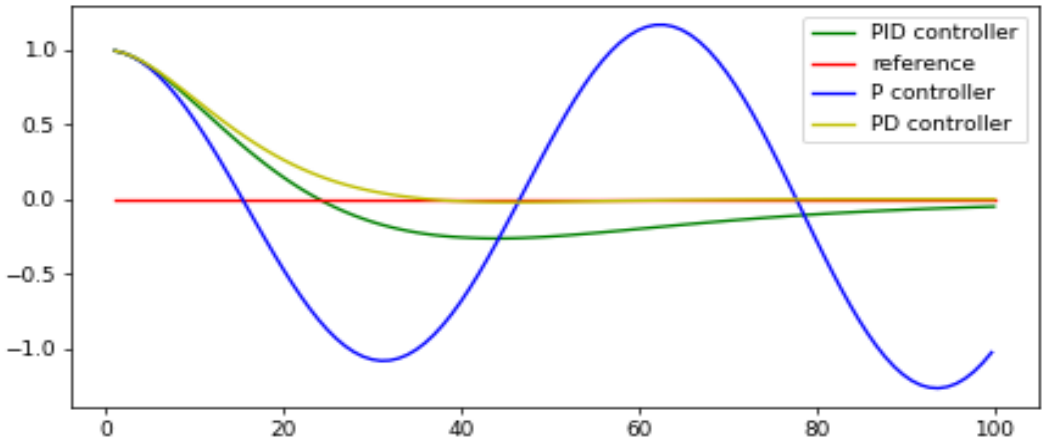
Output: Refer to the Video: **PD_Controller.mkv** in Results folder

4.3 PID Controller main.cpp

Here I stands for 'Integral'. Integral component counteracts with the bias reflected in CTE, so that there is a steady decrease in CTE and the vehicle reaches the Lane center. In other words, it adjusts the slight overshoot that was caused by bias.

A PID controller depends on all three components above,

$$Steering\ Angle = -\tau_p * CTE - \tau_D * \Delta CTE - \tau_I * \sum CTE$$



Output: Refer to the Video: **PID_Controller.mkv** in Results folder

5. PID Initialization main.cpp

To Implement PID Controller, we need to estimate Initial values of P, I and D.

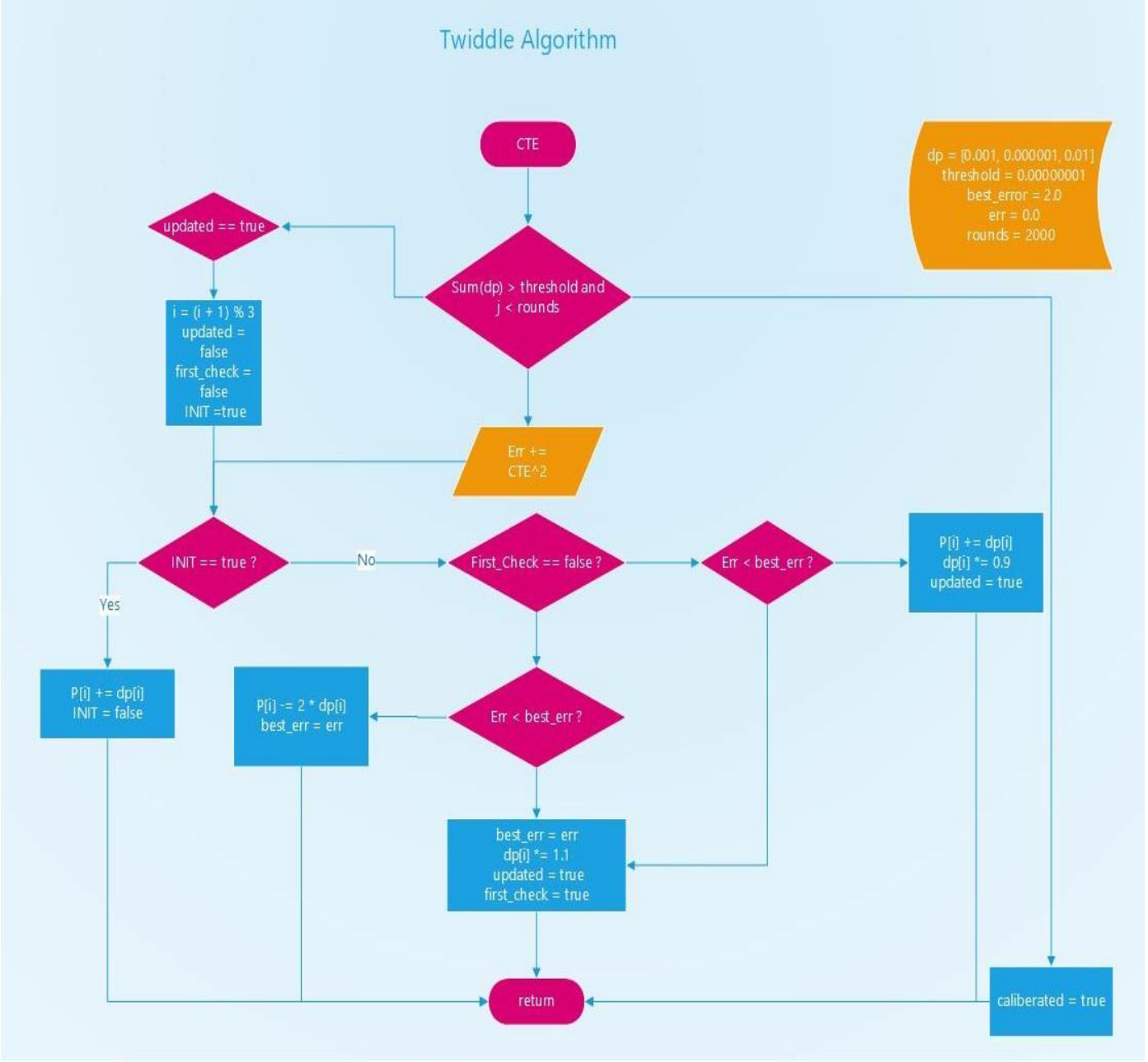
To estimate the initial value of P, I ran simulator with only P values keeping I and D at 0.

After many simulator runs, checking with values [1, 0.5, 0.1, 0.05], P = 0.05 showed less oscillations along the lane center

Similarly estimating D, with P = 0.05, with range of values [1, 0.8, 0.6, 0.5], D = 0.5 proved to stabilize the oscillations efficiently.

Estimating I was difficult. With P = 0.05, D = 0.5, running simulator with range of values [0.001, 0.005, 0.0001, 0.0005], I = 0.00005 showed much better results to negate the bias error.

5.1 Twiddle Implementation PID.cpp [68:129]



The Final Value of PID after tweaking with twiddle is: {0.061, 0.000057, 0.57}

Output: Refer to the Video: **Final_result.mkv** in Results folder