Lab Report
Assignment 2: PID controllers

Ryan Sleeuwaegen (s3122166) & Sirvan Kus (s3338827) &
Sjouk Ketwaru (s3287297)

April 28, 2023

1 Introduction

This report describes our project we made to implement a PID controller. The project was a simulation of a car driving on a road. The main idea is for the car to move to the center of the road as fast as possible and then stay on the center. With the help of the CTE (cross track error) it is possible to do this, in this context CTE is the distance perpendicular to the center of the road. Now the goal is to minimize it by looking for the best inputs for the car using the twiddle algorithm.

2 How do P, PD, and PID controllers work?

P, PD and PID are three different kinds of controllers.

P-controller:
A P-controller has only proportional (P) control, the correction is proportional to the error (difference between the set point and measurement), In this case the measurement is the CTE (cross-track error), and the set point would be center of the car. The P-controller is flawed in the sense that it is sensitive to overshoot and just makes a sine function on the road.

The formula for our P-controller:

$$\alpha = \text{-}t_P\text{CTE}$$

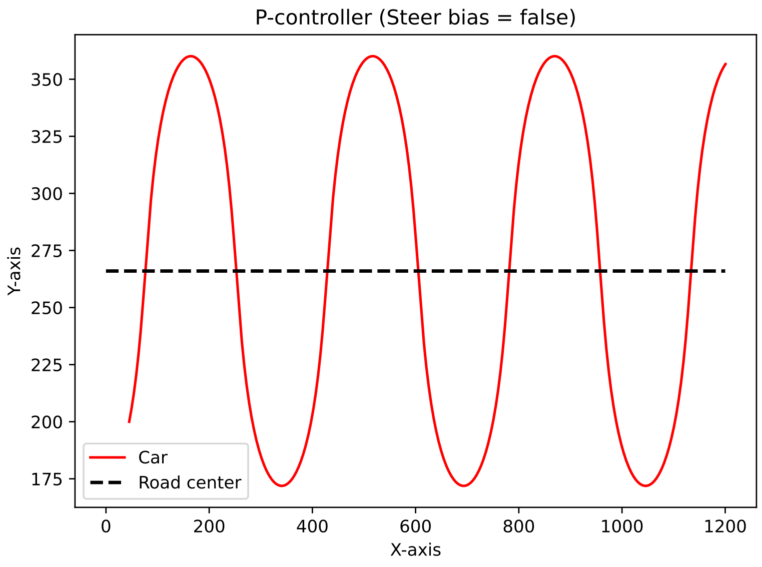

Figure 1: The simulation ran only with a p-controller, showing a sine function.

PD-controller:
To correct the flaw of overshooting from the P-controller, we get the derivative in time too. This results in the Proportional Derivative controller. The problem of a PD-controller is that it is incredibly sensitive too systematic bias.

The formula for our PD-controller:

$$\alpha = \text{-}t_P\text{CTE - }t_D\frac{d}{dt}\text{ CTE}$$

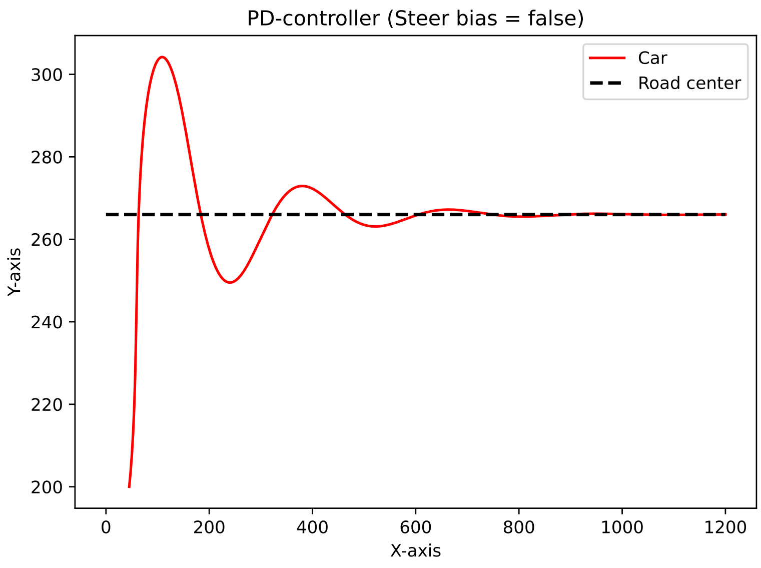

Figure 2

PID-controllers:
To correct the flaw of the PD-controller of systematic bias, we get the Integral too. This results in the Proportional Integral derivative controller. Now the correction is proportional to the error, derivative of error in time, and accumulated error over time.

The formula for our PID-controller:

$$\alpha = \text{-}t_P\text{CTE - }t_D\frac{d}{dt}\text{ CTE - }t_I\sum\text{CTE}$$

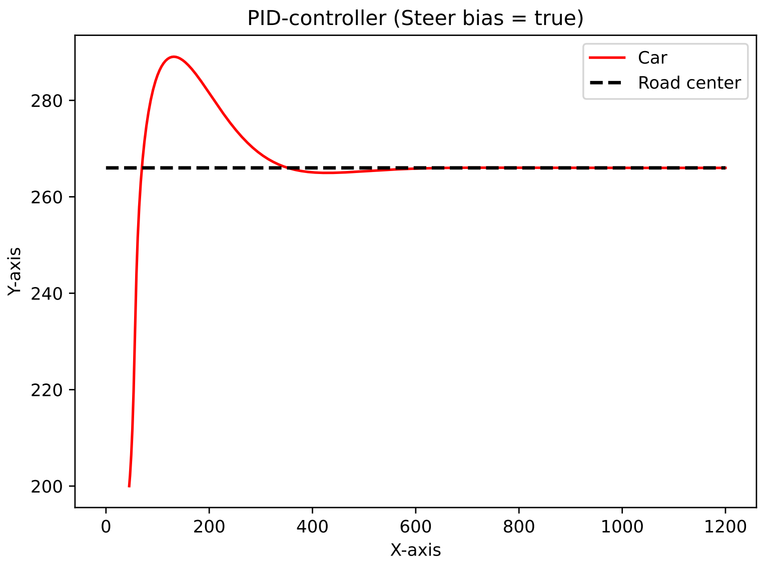

Figure 3

3 The Twiddle algorithm

The first step in the Twiddle algorithm is setting the tolerance (how low should the sum of change minimum be before we stop the program). Then we set a list called "P" with each of the values from PID, first one is Proportional, second one is Integral, last one is Derivative, these are set to the best values we got during manual testing. Then we set another list with the same structure however these are set to 1 for each variable of the PID, this list is called dp, namely delta p, how much we should change the values of p to get to the best values for the PID. (the variable it = 0, is just used for counting the number of iterations already ran.). Lastly we set the best error, which for the first run is just the current p (the best values we got during manual testing).

After all the variables we get to the while loop which stops if the sum of dp is smaller then a given tolerance (so the code stops when a certain tolerance is reached). In that loop there is a for loop which goes the length of p (so for the values P, I, D). In other words the for loop starts at P from PID then I from PID then D from PID. Each iteration in that for loop does the following: first it adds the dp at the ith place to the p at the ith place. After that it runs the simulation while clearing all needed parameters to see how well the simulation does with these new values (CTE_Total to calculate the best error without also counting the previous one, and player_car to make the car start at the correct position. Thereafter we get the CTE_total from the function that runs the simulator and changes the PID values, each run.

If the error is lower then the best_error we make the current error the best error and make the dp at the ith place 1,1 times bigger to see if we can make a bigger jump. If the current error is not smaller then the best_error we then take p at the ith position and remove 2*dp at the ith position from it to make the parameters smaller because the value was too big because it gave a bigger error then the best error. Hereafter we do a test run to see how these new parameters do, whilst still reseting the needed parameters CTE_total for the correct error calculation and player_car to start the car in the correct place. From there on we compare the current error again to the best error, if the current error is smaller then the best_error we set the best_error to the current error, and we do the dp at the ith place * 1,1 so we can make a bigger jump the next time since the current version is working out. If the current error is bigger then the best error then the list p at the ith place has gotten too small so we add dp at the ith place to it, to see if that value may work out. then we do dp at the ith place *0,9 in case it doesnt work out so we can try a smaller value again.

4 Experimental manipulations of the PID parameters

Manipulating the P-parameter value:
Looking at the figure 4, we can see that there is a significant change when we use a higher P-value. In the left graph (with a lower P-value) the car does not ever reach the center of the road, and already diverts from it again. In the right graph (with a higher P-value) we can see that the car overshoots a little.

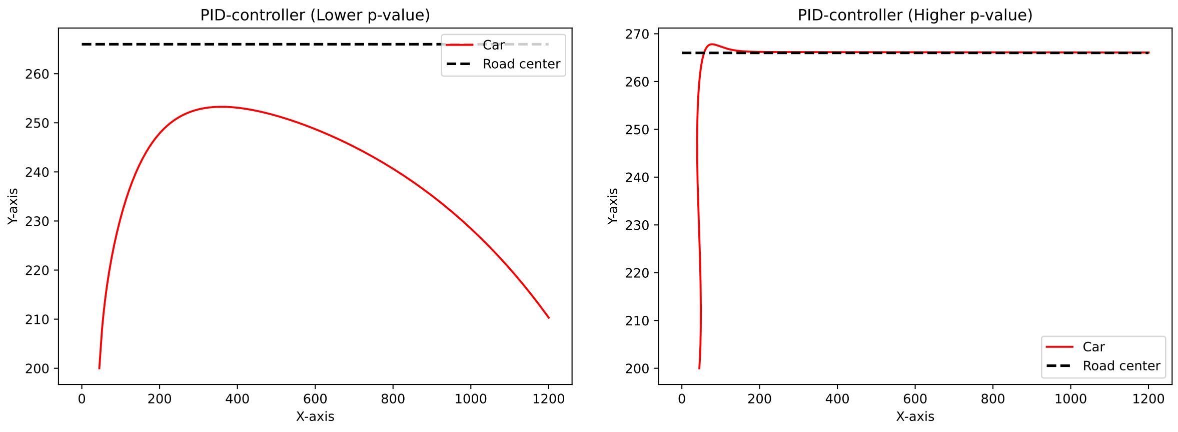

Figure 4

Manipulating the I-parameter value:
Looking at figure 5, we can see that when the I-value is too low (left graph), the car overshoots a little, but then goes to the center at a very low pace. On the other hand, when the I-value is too high (right graph) we can see that the car does not even reach the center and starts diverting from it.

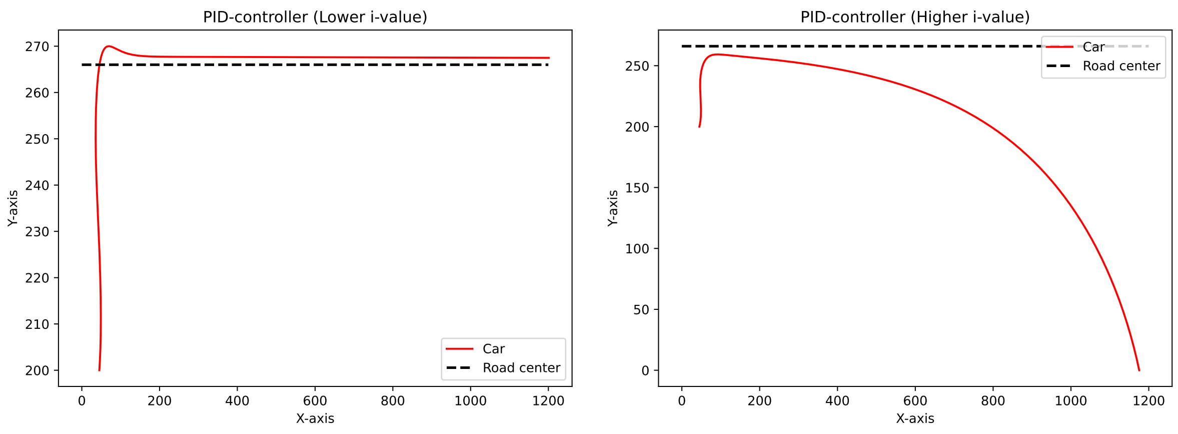

Figure 5

Manipulating the D-parameter value:
Looking at figure 6, we can see that when we use a lower D-value (left graph) the car overshoots a bit more. When you look closely you can see that the car swirls just a tiny bit around the center of the road. If the D-value is higher (right graph), the car does not get to the center, and slowly moves away from it.

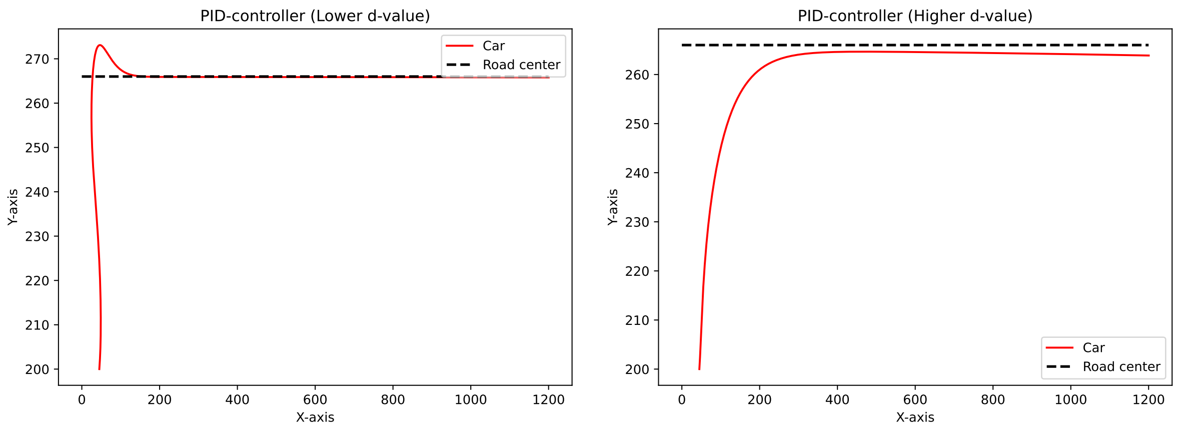

Figure 6

5 Optimal solution using Twiddle

In figure 7 you can see a visualization of our optimal solution after having our parameters optimized using Twiddle.

The results after Twiddle are:
P = 0,1408...
I = 0,0001...
D = 5,7228...
Best error (Total CTE) = 2683,3468...

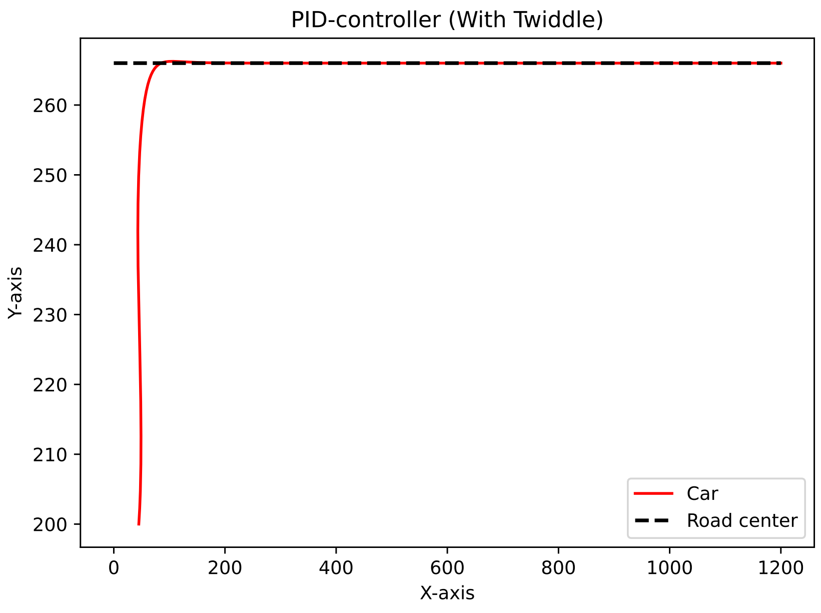

Figure 7