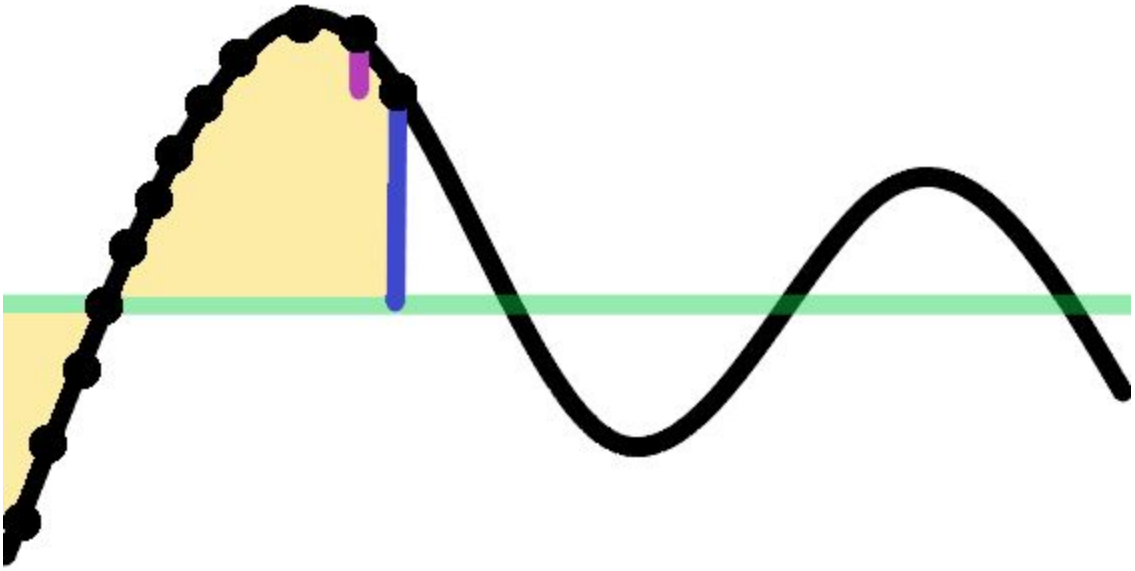
Meredith Meyer

PID Control

I utilized the principles of PID control in order to maneuver the car around the track in simulation. PID control involves setting a controllable feature in order to minimize the error in the outcome. In this situation, the controllable feature is the steering angle of the vehicle, and the outcome is the vehicle's position on the track. PID control consists of 3 components of error: the error itself (proportional[P]), the change in the error from the previous sample (derivative [D]), and the accumulation of error (integral [I]).



The image above illustrates the components of error. If the green line represents the target outcome, and the black line represents the actual output, then at the time represented by the last shown black point on the curve: error is shown as the blue line, derivative error is the pink line, and integral error is the yellow space. A PID controller uses 3 constants, one for each component of error, as factors in setting the next control value. Therefore, at this point in time, the controller would set the output (steering angle) as:

$$K_p * P_{error} + K_i * I_{error} + K_d * D_{error}$$

Setting the K values (the constants), is done through a process called tuning. For this project, I used a combination of manual tuning and twiddle, as discussed in the lesson. For manual tuning, I took the baseline steering values of [-1, 1] to logically decide on starting magnitudes. Since integral error is the largest value, K_i should be the smallest value. The next largest error would likely be proportional error, so K_p is the middle value, and that leaves K_d as the largest value. Using simple factors of 10, I started with K_p=0.01, K_i=0.001, and K_d=0.1. However, in my manner of manual tuning, I started first with a P controller, then made it a PD controller, and finally a PID controller. This allowed me to manually tune each one of the parameters at a time. My self-tuning method was sort of a manual twiddle: my first comparison was multiplying the parameter by 10, and if that was not an improvement, I would divide the

original parameter by 10. Then I decreased the magnitude of the change until I found values that were reasonable estimates. The values I found were: K_p=0.05, K_i=0.001, and K_d=1.

Once I had this starting point, I incorporating twiddle to optimize the parameters further. I started with my aforementioned values, and dp values of {0.01, 0.001, 0.1}, about a factor of 10 below the starting values. The process of twiddle looks like:

- Get the current error
- Increment the parameter & get the new error:
    - If new error is better:
        - Increment the parameter by a slightly increased increment
    - If new error is not better:
        - Set the parameter to 1 increment below what it was last
        - Get a new error
            - If better error:
                - Increment the parameter by a slightly increased increment
            - If worse:
                - Go back to the middle value
                - Decrease the increment amount & increment

Where all of these steps are looped for each parameter. Twiddle helps to hone in on the optimized value of each parameter. Through utilizing twiddle, resetting my starting values, and running twiddle again, I was able to optimize the K parameters. The final output was:

Best average error= 0.047818
Best PID values: 0.0718455, 0.00449649, 1.4344
Sum DP: 0.000991926

and the car was able to drive around the track continuously without leaving the driveable portion of the track.