

## Lab 2: PID Control

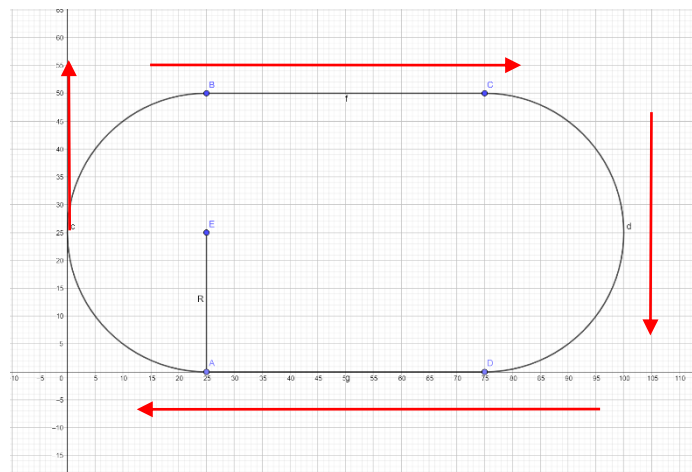
You are given three Python programs:

1) “PIDControl Straightline Twiddle.py” is the Python program from Udacity course Lesson 15: PID Control, “21. Parameter Optimization (solution)”. It considers a straight-line reference trajectory, using function `twiddle()` to automatically tune controller gains, incl. `params[0]` (Proportional), `params[1]` (Derivative) and `params[2]` (Integral):

<https://classroom.udacity.com/courses/cs373/lessons/48743150/concepts/f9fe06f9-9b1c-40b1-b9ad-312ca92be287>

2) “PIDControl Racetrack.py” is based on the Python program from Lesson 16: Problem Set 5, “4. Quiz: Racetrack control”. It considers a circular clockwise reference trajectory, with start position at (0,25).

<https://classroom.udacity.com/courses/cs373/lessons/48721468/concepts/487015300923>

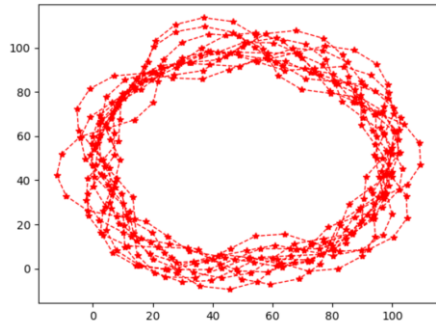


3) “PIDControl Racetrack Twiddle.py” adds the function `twiddle()` to PIDControl Racetrack.py to tune controller gains `params[]` automatically.

**Task 1:** (The first two programs are for your reference only.) Start from PIDControl Racetrack Twiddle.py, change the track shape to a round circle with radius 50 (so you need to modify the function `cte()` for computing cross-track error), and modify `twiddle()` to tune:

- 1) P controller, with  $K_i = K_d = 0$ ;
- 2) PI controller, with  $K_d = 0$ ;
- 3) PD controller, with  $K_i = 0$ ;
- 4) PID controller.

**Task 2:** Continue from your code in Task 1. The car speed is set to 1.0 in the original program. Setting a higher speed may cause the car to lose control, as shown in the following visualization of the actual trajectory for speed of 10.0, obtained with `plt.plot()` (you may or may not get the exact same result).



To keep control, one can either increase track length by increasing circle radius, or reduce timestep size  $dt$  (increase sampling rate). Let's take the latter approach. (Refer to L6 Control, p. 34 "For General  $dt$ " for how to add the variable  $dt$ .) Tune a PID controller with `twiddle()` for `speed=10.0`. Try different timestep sizes  $dt$  until you achieve low tracking error. (A good heuristic is to reduce  $dt$  proportional to the increasing speed, to keep the distance traveled in each timestep constant.)

(The total number of simulation timesteps is  $2N$ . We compute the average error and record the trajectory for the last  $N$  timesteps, in order to skip initial transient dynamics in the first  $N$  timesteps and focus on the steady state performance. We suggest setting  $N=1000$  to traverse the track multiple cycles.)

### Submit a report and corresponding Python programs to Canvas:

#### For Task 1:

For each of the 4 controllers (P, PI, PD, PID) with default speed 1.0:

1. The final controller gains and average error.
2. Visualization of actual trajectory for the last  $N$  timesteps.
3. Python programs named "PControl Racetrack-YourLastName.py", "PIControl Racetrack-YourLastName.py", "PDControl Racetrack-YourLastName.py", "PIDControl Racetrack-YourLastName.py",

#### For Task 2:

For PID controller with speed 10.0, and for (a) timestep 1.0, (b) your final  $dt$  that achieves low tracking error:

1. The final controller gains and average error.
2. Visualization of actual trajectory for the last  $N$  timesteps.
3. Python programs named "PIDControl Racetrack-TS1-YourLastName.py", "PIDControl Racetrack-TSdt-YourLastName.py".

#### Programming Tips:

One way of implementing the P, PD, PI control is by initializing the array `dp` as follows:

PID: `dp = [1.0, 1.0, 1.0]`

PD: `dp = [1.0, 1.0, 0]`

P: `dp = [1.0, 0, 0]`

P: `dp = [1.0, 0, 1.0]`

Another way is to initialize array `p` to 0:

```
p = [0.0, 0.0, 0.0]
then use different loop indices when updating array p:
PID: for i in range(len(p))
PD: for i in range(len(p)-1)
P: for i in range(len(p)-2)
PI: for i in range(len(p))
    if i==1 continue
```