1 Getting started

Last week we worked on using an evolutionary algorithm to solve linear regression. This week we will use a different evolutionary algorithm to optimize the PID controller from the previous lab report. In that lab report you used the Twiddle algorithm for optimization. This week we will replace Twiddle with an evolutionary algorithm. Again, your lab report on evolutionary robotics will include results from all three weeks, so keep detailed notes!

1.1 The problem of control

As you are (hopefully!) aware, the PID controller in our example dealt with the problem of minimizing the deviation from an optimal track given only the deviation at one timepoint and only the steering angle as output.

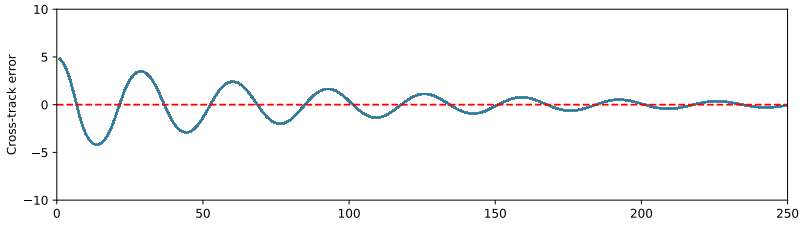

Figure 1: Visualization of cross-track error (CTE) for an unoptimized PID controller. Oscillations around the optimal track decrease over time, but an optimized PID controller will have a smaller overall CTE.

To minimize total cross-track error, we need to find optimal values for the P-, I-, and D-term for our controller. We can use Twiddle for this, but–of course–also an evolutionary algorithm.

2 Your evolutionary algorithm

You will work with the PID controller that you implemented in week 8. You should make sure that your P, I, and D terms are not hard-coded but can be passed as parameters to your PID controller when you instantiate the PID class.

2.1 The error landscape

Investigate the error landscape, similar to what you did last week. Is it smooth and convex? Is this well-suited to regular optimization algorithms? How about evolutionary algorithms? Discuss this in terms of local minima and convexity.

Of course, because we have three parameters this is more difficult to visualize. Show three surface plots, one for each combination of two parameters while keeping one fixed.

2.2 Implementing your algorithm

Implement an evolutionary algorithm to optimize your PID controller. This week we will be manipulating selection strategies as discussed in the lecture. Create two functions, one called `optimize_evo()`, and the

other called `optimize_twiddle()`. You can use the implementation you used in the PID assignment for the `optimize_twiddle()` function. In your implementation of the evolutionary algorithm you should work with a fixed population size of 100.

2.3 What to investigate

First, start with a simple *truncation selection*, similar to last week's assignment. Manipulate the number of individuals you copy each generation, let's say 2, 5, 10, and 20. Plot the mean number of generations needed for optimization. Do this with and without elitism. Plot both lines in different colors in a figure.

Investigate different mutation levels, defined as the spread of the normally distributed values you add to the genotypes. Visualize your results.

Now implement the *roulette wheel method*. Compare the speed of convergence to truncation selection in number of generations.

Now implement the *tournament-based selection*. Compare the speed of convergence to truncation selection and the roulette wheel method in number of generations.

Finally, compare these methods to Twiddle in terms of wall time. Which would you prefer to optimize your PID controller? Explain your argument!

2.4 Submission

You will submit your lab report after the third week of evolutionary robotics. It will consist of your work from all three weeks. Submit your code to a separate assignment that is able to accept .py files. Instructions will be given after the third week.

Good luck with this week's assignment, and remember: we are here to help you if you get stuck.