Self-driving cars program - project 8: PID controller

Francesco Boi

Contents

1 Content of the project **1**

2 Project goals **1**

3 Instruction **2**

4 Parameters tuning **2**
4.1 Kp, Kd, Ki 2
4.2 PID class 2

5 Twiddle algorithm **2**
5.1 dKp, dKd, dKi 2
5.2 Twiddle algorithm and state machine 3
5.3 Automatically reset 3

6 Increase throttle **3**

7 Next **3**

1 Content of the project

Here is the content of the project:

- *writeup.pdf* (this file): report of the project;
- *writeup.tex*: source tex file;
- *src*: folder containing the C++ source code of the project;
- *README.md*: file giving a general description of the project.

2 Project goals

The goal of this project is to tune the parameters of a PID controller that automatically drives the car in the simulator.

3 Instruction

Instructions to compile and run the code are included in the README.md file.

4 Parameters tuning

The first set of parameters has been found manually by using the PID class and the standard throttle. The methods and variables implemented in the class are straightforward. One thing to notice is that in the `main.cpp`, when passing the cte error to calculate the total error, its third power is used. This has the effect to penalise more large cte and less cte.

4.1 Kp, Kd, Ki

The PID generates a control based on three components of the error. The three components are the error itself, its time derivative (i.e. the difference between the current error and the one at the previous error) and the integral component (i.e., the sum of all the errors or alternatively the sum in the last period). Each component is multiplied by a constant, respectively `Kp`, `Kd` and `Ki`.

4.2 PID class

Each time a new error is received, the method `PID::UpdateError` is called. This updates the three components described previously that in the class are called respectively `PID::d_error`, `PID::p_error` and `PID::i_error`. To prevent the `i_error` term to continuously increasing, a counter variable has been defined.

The control signal is generated by the function which sums up each error component taken with opposite sign, multiplied by the proper constant. The `i_error` is divided by the `counter` variable to get its average.

5 Twiddle algorithm

To fine tune the parameters the twiddle algorithm has been implemented. This was done by declaring a new class. To keep the functionalities of the PID, inheritance has been used: The `Twiddle` class has been defined as a child class of `PID` and the `Init` and `TotalError` methods overridden. For this reason these were declared as virtual in the parent class.

5.1 dKp, dKd, dKi

These variables represent the amount to be added or subtracted when updating `Kp`, `Kd` and `Ki`. The algorithm is considered tuned when their sum is less than a given quantity and this check is implemented in the `checkTuningCondition`.

5.2 Twiddle algorithm and state machine

The steps of the twiddle algorithm is implemented in the `TotalError` method by using a state machine. The following states are defined:

- `initialising`: at first the algorithm is let run to for the given number of iterations to get a first error;
- `increment_tuning`: this state corresponds to the case where the next constant needs to be incremented by the corresponding `dK` variable;
- `increment_tuning`: this state corresponds to the case where the next constant needs to be decremented by the corresponding `dK` variable;
- `increment_tuning`: after the algorithm has run, the three parameters are fixed and need not changed.

For each state the algorithm runs for a given number of steps eventually change to the next state.

After the first state represented by `initialising` has determined the the first error, the `state` is set to `increment_tuning` and the first `K` variable is increased (the algorithm starts with `Kp`). If the obtained error is less than the previously obtained ones, then the next `K` is going to be updated at the next iteration, otherwise the algorithm tries to decrement the current `K`. If neither of the two improves the error, then for the moment the current `K` value is kept, the amount for the increment and decrement is reduced by 10% and the next variable is going to be updated.

At the end of each iteration, the next parameter to be updated and the corresponding update amount is obtained by using an instance variable named `paramToBeUpdated` and the methods `getK`, `getdK`, `setK` and `setdK` allow to get and set these values.

5.3 Automatically reset

The `Twiddle` class has a `uWS::WebSocket<uWS::SERVER> *ws` variable used to reset the run. The run is reset if the speed is close to 0 (for example when the car gets stacked) or when the `current_error` gets bigger than the `best_error` before completing all the steps. This is done by sending a proper message through the websocket. The instance method to do that is `reset`.

6 Increase throttle

After a the Twiddle found some good values, the throttle was increased and the algorithm rerun. Locally, I have managed to run with a throttle of 0.7.

7 Next

The next thing to be done is to create a PID for the throttle.