Control and Trajectory Tracking for Autonomous Vehicles

Author: Joao Silva, joao.p.silva@mercedes-benz.com
Date: 2022-12-06

Project Scope

The goal of this project is to design and implement a PID controller class, which is used twice through the project. These two controllers generate the commands to feed to the vehicle driving outputs: one command is given as the steering angle to rotate the steering wheel, and therefore the vehicle. The other command is fed to either the accelerator or the brake pedal as the throttle percentage, given either from -100% (full brake) to 100% (full accelerate). The PID controller is firstly implemented, and then tunned in order to properly control the vehicle to follow a trajectory given by the Behavior planner. Different tunning coefficients are experimented with, as well as some strategies to make the controllers inputs more robust and improve vehicle-driving dynamics.

PID Controller Class

The PID controller consists of the C++ class which includes the following methods:

Init(): Initializes member variables of the PID Class, including: sets the PID coefficients k_P, k_I and k_D; sets the PID output min and max thresholds enabling a saturation for robustness of output; resets other class member variables to default vales;

UpdateError(): updates the error for cycle i. The instant error e_i is given as input, and the accumulated error is integrated over time using the cycle-to-cycle delta time ΔT_{i-1}^{i}

UpdateDeltaTime(): updates the cycle i delta time, which is the time difference between the current and last cycle ΔT_{i-1}^{i}.

TotalError(): Computes the cycle total error. This would be the output of the PID controller, meaning the total error shall be given as in formula at cycle i:

$$E_i = k_P \cdot e_i + k_I \sum_{j=0}^{i} e_j \Delta T_{j-1}^{j} + k_D \cdot \frac{(e_i - e_{i-1})}{\Delta T_{i-1}^{i}}$$

Notes towards PID Controller Implementation:

As convention, we decided to define error in the relation of the difference between Target and Measured variable of interest. The integrated error is statically stored, so that from cycle-to-cycle we just have to update the error increment. The delta time is protected to avoid unreasonable control output if small time is passed (mostly to protect for mathematical errors in a division close to zero).

Throttle Control

To implement throttle control, meaning longitudinal motion, we opted for a throttle input not only correlated to the error associated with the target speed, but also to the error associated with the target longitudinal position at an adequate distance. The reason for that is that if we used only the speed of the target, it could mean that ideally we would be moving at the target speed but possibly before/further than the expected longitudinal trajectory point at a given time point. We could also similarly use only the position target, but that could mean that we would reach that target slower/faster than the expected trajectory speed.

We thus have two input errors on the throttle PID controller, representing two physical quantities measured in m and m/s. They need to be related. We opted for a relation which favors more the speed control, given by:

$$e_{throttle} = e_{speed} + 0.8 \cdot e_{pos_{lon}}$$

The formula assumes position and speed in the same order of magnitude, which is the case because we deal with speed/position in the range of $\pm 5m/s$ or $\pm 5m$.

For the throttle PID controller using $e_{throttle}$, we opted for the following coefficients:

$$k_P = 0.12; \;\; k_I = 0.005; \;\; k_D = 0.02$$

Steering Control

To implement steering control, meaning lateral motion, we opted for a strategy also similar to the one of throttle control, meaning we used a steering input not only correlated to the error associated with the target trajectory angle, but also to the error associated with the target lateral position at an adequate distance. The reason for that is that if we used only the angle of the target trajectory, it could mean that ideally we would be moving at the target trajectory orientation but possibly with a lateral deviation concerning trajectory point at a given time point. We could also similarly use only the position target, but that could mean that we would reach that position with an incorrect orientation.

We thus have two input errors on the steering PID controller, representing two physical quantities measured in rad and m. They need to be then related. We opted for a relation which favors more the yaw/orientation control, given by:

$$e_{steer} = e_{yaw} + 0.4 \cdot e_{pos_{lat}}$$

The formula assumes position and angle in the same order of magnitude, which is the case because we deal with angle/position in the range of $\pm 2rad$ or $\pm 2m$.

For the throttle PID controller using e_{steer} , we opted for the following coefficients:

$$k_P = 0.02; \;\; k_I = 0.006; \;\; k_D = 0.2$$

Notes towards Throttle and Steering Control

As mentioned earlier, we opted to use longitudinal trajectory point as input. Additionally, we made a strict enforcement that forces that if the vehicle is ahead of the generated trajectory (all longitudinal target values are negative), meaning vehicle doesn't know where to move further, then the vehicle should stop. In this case, we forced the vehicle to wait until an ahead trajectory comes (eventually fully stopping) by faking an input that corresponds to an error composed in $0m$ longitudinal position error and a velocity error equal to the negative of current speed (meaning speed target is $0m/s$).

Also for steering, if the vehicle is ahead of the generated trajectory then it doesn't know where to go and in this case, it assumes that it will drive to a full stop. Therefore, it simply uses null steering, meaning keep driving straight until reaching stop.

Another point is that it appears the $i = 0$ cycle input velocity seems to be wrongly estimated. For that we overwrite this input and use it with $0m/s$.

Finally, we transform the trajectory points to vehicle coordinates, firstly by discarding vehicle orientation, but then also using it to compute the PID error inputs.

PID Efficiency evaluation and Tunning

After implementing the controller and deciding on the input error strategy, we moved towards tunning the coefficients. We will consider PID efficient as the faster and steadier we will reach a zero error. Firstly, we started with longitudinal control, then lateral control, and finally a fine-tunning for the mix of both. We will show only the tunning of the throttle controller, as for the steering controller would be similar.

Firstly, in the following Figures, we can see the effect of the increase in the propositional coefficient and the effect on throttle error. The bigger the k_P, the bigger the response throttle to the same input error, and the sooner the vehicle will start moving. Since there are drag and other acceleration effects, which are not modeled, that does not mean that we reach a desired speed faster, because the earlier we get to the goal the less the vehicle accelerates, and so the speed increase rate slows down.

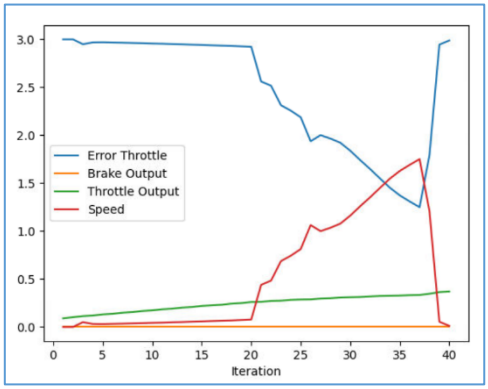

Figure 1 - Small proportional coefficient

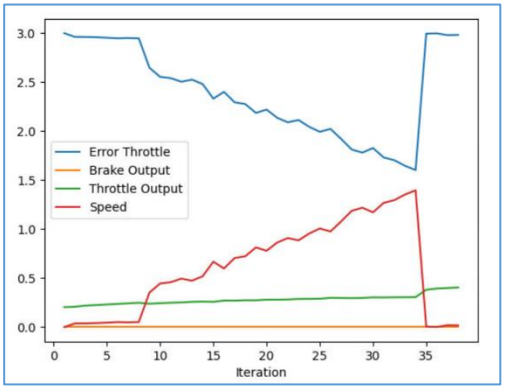

Figure 2 - Larger proportional coefficient

Secondly, to compensate for the drag and other acceleration dynamics, the integral coefficient k_P is increased. This makes the initial response much faster, as the initial struggle to start vehicle moving contributes severely to increase integral error. Nevertheless, we have to be careful not to increase the coefficient too much and overshoot, as seen in Figure 4. To counteract this, we can tune the derivative coefficient, which should smoothen the curve.

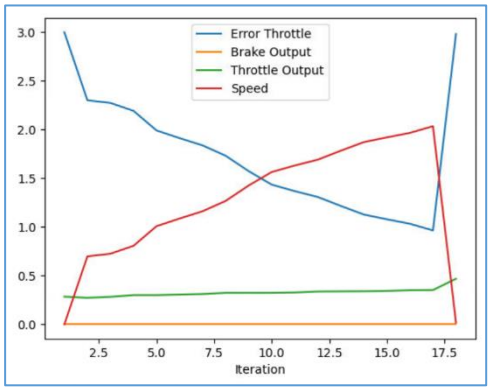

Figure 3 – Acceptable integral coefficient

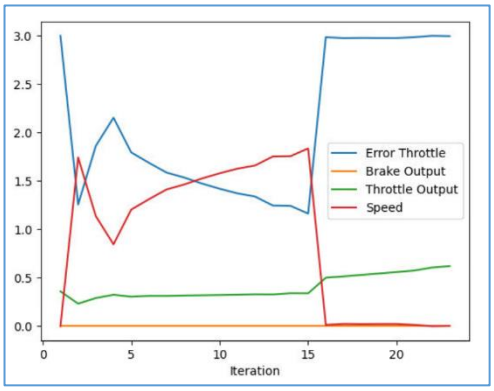

Figure 4 - Larger integral coefficient

Thirdly, with initial throttle and steering coefficients tunned, we can see an initial coefficient set that is not optimal. In fact, it leads to a crash as seen in Figure 7.

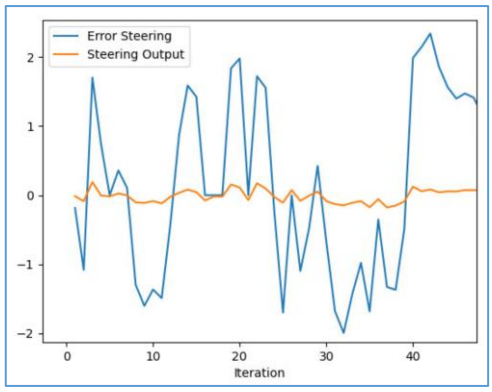

Figure 5 – Steering output leading to a crash

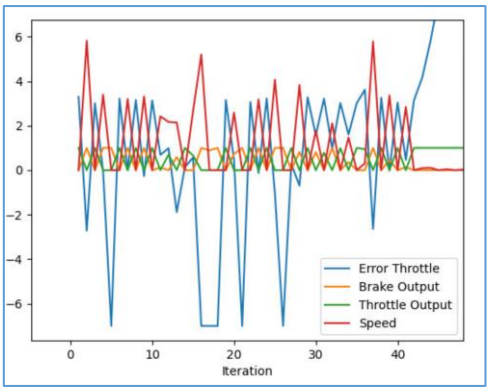

Figure 6 – Throttle output leading to a crash

Figure 7 - Vehicle crashes into a tree

Finally, we fine-tune together both the throttle and steering coefficients. This leads to a vehicle behavior that, although is not perfect, is optimal in the sense that can properly follow the given trajectory, not leave the road or hit any obstacle along it.

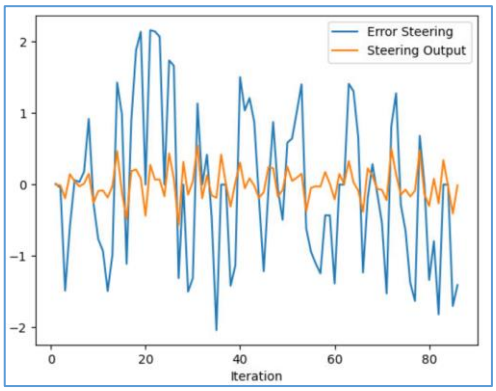

Figure 8 – Steering output with optimal coefficients

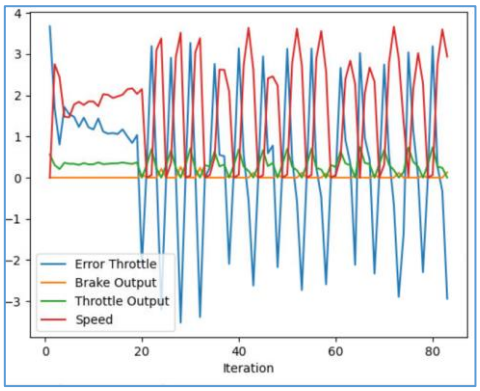

Figure 9 – Throttle output with optimal coefficients

Figure 10 - Vehicle side by side with another vehicle obstacle

Figure 11 – Vehicle reaches end of road and stops

Although the coefficients are not perfect, they are optimal in the mentioned sense. Nevertheless, the coefficients could also be further optimized, even automatically tunned. In order to do so, one has to define a cost function, and then automatically experiment what would be the cost of running a given trajectory with a given set of coefficients. The coefficients yielding better results will be optimal in that sense, but then again they would be biased to a given trajectory. Furthermore, one could experiment with a multitude of standardized trajectories, and combined the costs until finding which coefficients are better for each trajectory.

PID as a Model free Controller

PID is a simple 3-coefficients model and independent of the system it is meant to control. This means, the vehicle dynamics or even the trajectory kinematics information are not being used at all. An example of such is that, as we mentioned earlier, the trajectory has a position component but also a velocity component, and this can be combined. So since the PID only takes one single input, it needs to be tricked into thinking it has more (without even knowing it). Furthermore, the action envelope of the vehicle is quite broad, and we might want to have different coefficients depending on speed for example. Or, as another example, a car accelerating dynamics are non-linear, as starting from stand-still is different than just cruising and adapting slightly the speed.

On the other hand, simplicity is also not to be ignored, and PID provides for an easy to implement, straight-forward model that could potentially be used in several very different systems and achieve sufficiently good results, with less effort and just some tunning activities.

PID Controller improvements

The PID is a simple controller but there are many ways one could improve it. As mentioned already, one is to try to combine different inputs (position/speed, position/direction). Another would be to automatically tune the PID coefficients as mentioned earlier. Ultimately, one could even try different coefficient for different driving dynamics, example for start-stop, for mid-speeds and for high-speeds.