2021 IEEE Intelligent Transportation Systems Conference (ITSC)
Indianapolis, USA. September 19-21, 2021

Low-level Controllers Play a Decisive Role in the String Stability of Adaptive Cruise Control

Hao Zhou[1], Anye Zhou[1], Tienan Li[2], Danjue Chen[2], Srinivas Peeta[1], Jorge Laval[*1]

***Abstract*—Current commercial adaptive cruise control (ACC) systems consist of an upper-level planner controller that decides the optimal trajectory that should be followed, and a low-level controller in charge of sending the gas/break signals to the mechanical system to actually move the vehicle. We find that the low-level controller has a significant impact on the string stability (SS) even if the planner is string stable: (i) a slow controller deteriorates the SS, (ii) slow controllers are common as they arise from insufficient control gains, from a "weak" gas/brake system or both, and (iii) the integral term in a slow controller causes undesired overshooting which affects the SS. Accordingly, we suggest tuning up the proportional/feedforward gain and ensuring the gas/brake is not "weak". The study results are validated both numerically and empirically with data from commercial cars.**

***Index Terms*—string stability, low-level controller, ACC**

I. Introduction

With the development of vehicle automation, adaptive cruise control (ACC) systems are now widely available on commercial vehicles around the world. To be beneficial in terms of traffic efficiency, ACC systems are expected to achieve string stability (SS) to ensure that small perturbations do not amplify along a platoon of vehicles [1], [2], i.e., speed fluctuations should be dampened rather than amplified by the follower [3], [4]. A commercial ACC system is typically comprised of an upper-level planner and a low-level controller. The planner uses sensor data to estimate the kinematic information of preceding vehicles (e.g., spacing, speed) and outputs a desired speed/acceleration. Then, the low-level controller is in charge of actually moving the vehicle towards the desired speed/acceleration target.

In recent years, the SS of ACC systems has drawn increasing attention from the traffic flow community, in the context of fast developing self-driving technologies and their potential impact on traffic flow efficiency. Since the ACC modules on commercial car models are "black boxes", researchers can only collect empirical driving data of the ACC systems, and retrofit them with controllers or car-following (CF) models to investigate the SS using simulation [5], [6].

*Cooresponding author

This work is supported by the NSF Grant CMMI 1826162, CMMI 1826003, and 1932921.

[1]H. Zhou, A. Zhou, J. Laval, and S. Peeta are with the School of Civil and Environmental Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA (e-mail: zhouhao, azhou46, peeta@gatech.edu, jorge.laval@ce.gatech.edu).

[2]T. Li and D. Chen are with the Department of Civil and Environmental Engineering, University of Massachusetts, Lowell, MA 01854, USA (e-mail: tienan_li@student.uml.edu, danjue_chen@uml.edu)

The SS of ACC systems has been studied since the turn of the millennium in the control area [7]–[9]. There are a plethora of studies working towards the design of string stable ACC or cooperative adaptive cruise control (CACC) systems. Most studies focus on deriving the theoretical SS condition of a simplified upper-level planner, and some studies validate SS performance using real-car experiments [10], [11]. However, the impacts of low-level controllers have been consistently neglected. This might be because most cars used for SS experiments are retrofitted, possibly use different hardware compared to commercial vehicles and operate on customized platform such as the robot operating system, rather than commercial ACC systems.

Existing studies have identified that most commercial ACC systems are string-unstable [12]. In particular, [9] conjectures that string-unstable ACCs are likely due to the vehicle actuators driven by low-level controller cannot perfectly follow the upper-level planner. However, an explicit analysis of the impact is missing. In addition, [11], [13] recognize the existence and potential impact of low-level controllers but still treated them as black-boxes. So far the impact of low-level controller has not been studied in detail, and pertinent experiments using real cars are still missing.

Although empirical ACC data has been collected and used to retrofit commercial ACC vehicles, to the best of our knowledge, no commercial ACC algorithms have been thoroughly investigated and evaluated. To fill the gaps related to evaluating low-level controller and investigating commercial ACC algorithms, this paper dives into the open-source commercial ACC algorithms in Openpilot [14], a self-driving software published by Comma.ai, one of the leading self-driving technology companies with more than 40 million driven miles [14]. The after-market ACC device of Comma.ai is called Comma Two, which connects and overwrites the stock ACC modules on commercial cars with its own self-driving software Openpilot. The Openpilot software is not only open-source, but also allows users to customize their own ACC algorithms and test it on real cars, which is aligned with the objective of this study, and enables us to highlight the differences between commercial ACC system in the real world and the simplified controller/CF models in the literature. Correspondingly, with an emphasis on the impact of the low-level controller, this study analytically explores the impact mechanisms of low-level controller, and leverages the software (Openpilot) and hardware (Comma Two) from Comma.ai to validate our findings through simulations and

2021 IEEE International Intelligent Transportation Systems Conference (ITSC) | 978-1-7281-9142-3/21/$31.00 ©2021 IEEE | DOI: 10.1109/ITSC48978.2021.9564679

real-car tests.

The remainder of the paper is organized as follows: section II introduces the commercial ACC algorithms used in Openpilot; section III presents the detailed mechanism of the impact of the low-level controller on SS; section IV showcases the simulation results; section V validates the major findings using real drives on commercial cars running Openpilot; section VI concludes the paper and discusses future directions.

II. Background

A. Pipeline of the longitudinal control in Openpilot

Fig. 1 shows the pipeline of the longitudinal control of self-driving cars in Openpilot. The longitudinal control starts with a planner running at 20hz (equal to the radar sampling rate) which outputs a target speed, v_{target}, or target acceleration, a_{target}, based on certain type of (car-following) CF models. The target value is then passed on to the low-level controller. Notice that the low-level controller runs at the 100 hz, which is required by the gas/brake system of a modern car. Hence every planning period would consist of 5 consecutive control steps. For the low-level controller, the logic is as follows: i) the speed or acceleration setpoint, v_{pid} and a_{pid}, for each of the 5 control steps is first computed based on v_{target} or a_{target} ; ii) then a proportional-integral (PI) or proportional-integral-feedforward (PIF) controller is designed to achieve v_{pid} or a_{pid} by outputting the $control$ input; iii) the control input is then fed to a processor that maps it to the actuator command, which is a gas or brake percent gb for the car. We refer to this process as the $compute\_gb$ function. If we view the $control$ variable as the desired acceleration, then the $compute\_gb$ function is a mapping from an acceleration to a gas/brake percentage. Conversely, we can call the mapping from gb to the true accelerations the $gb2accel$ function which largely depends on the engine and braking performances of a car. The gas/brake percentage is then applied on the ego vehicle to obtain the true acceleration a_{ego}, speed v_{ego}, and position x_{ego}.

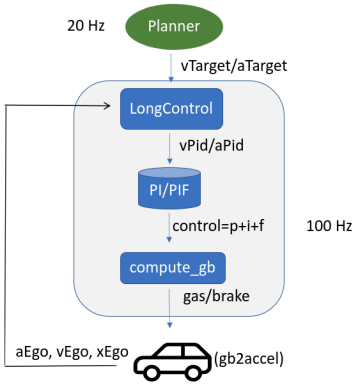


Fig. 1: Pipeline for longitudinal control.

B. The Upper-level planner

1) The linear planner: As mentioned previously, the planner can be a linear controller, or a CF model in traffic domain, or even a model predictive controller. Here we start with a regular linear-controller type planner with constant time headway policy (CTH), which is common on commercial ACC systems. Given a desired time headway τ, and the jam spacing δ, the desired spacing, s_{des}, from the front bumper of the ego vehicle to the rear bumper of the lead vehicle, is calculated as:

$$s_{des} = \delta + \tau \cdot v_{lead} \tag{1}$$

Then the planner outputs a target speed v_{target} given by (2) below to reduce the error between the true spacing s_{ego} and the desired spacing s_{des} by a rate k_v. Note that in practice, k_v is a linear function of v_{ego}, rather than a constant as in most existing literature [5], [15].

$$v_{target} = (s_{ego} - s_{des}) \cdot k_v + v_{lead} \tag{2}$$

The actual v_{target} will be further constrained by acceleration limits from multiple sources such as comfortableness, the vehicle dynamics, maximum speed difference from the leader, and the impact of curves.

2) MPC planner: For a regular commercial vehicle, a professional solver (for example Acado [16]) is needed for an MPC planner to compile and run in real time. Consequently, the MPC controller follows a predefined optimization framework required by the solver. The formulation of the optimization objective and the reference trajectory are two key components for an MPC problem, while how to solve it is out of scope for this paper. Now we will introduce these two key components.

In Openpilot the objective function $C(\hat{t})$ for the longitudinal MPC at the planning time $\hat{t}$ is defined as a weighted sum of four sub-costs, time to collision, spacing, acceleration and jerk:

$$\begin{aligned} C(\hat{t}) = \sum_{\hat{t} \le k \le \hat{t}+T_{mpc}} & w_{ttc} C_{ttc}(k) + w_{dist} C_{dist}(k) \\ & + w_{accel} C_{accel}(k) + w_{jerk} C_{jerk}(k) \end{aligned} \tag{3}$$

where T_{mpc} is the optimization horizon, the tunable weights applied here are $w_{ttc} = 5$, $w_{dist} = 0.1$, $w_{accel} = 10$, $w_{jerk} = 20$. The four sub-costs are defined as: $C_{ttc} = \exp\{\frac{s_{des}-s_{ego}}{(\sqrt{v_{ego}+0.5}+0.1)/0.3}\}-1$, $C_{dist} = \frac{s_{des}-s_{ego}}{0.05 v_{ego}+0.5}$, $C_{accel} = a_{ego}(0.1 v_{ego}+1)$, and $C_{jerk} = j_{ego}(0.1 v_{ego}+1)$, where j_{ego} is the jerk of ego vehicle (i.e., derivative of acceleration), the desired spacing s_{des} for the MPC controller is defined as:

$$s_{des} = v_{ego} \cdot \tau - (v_{lead} - v_{ego}) \cdot \tau + \frac{v_{ego}^2 - v_{lead}^2}{2g} \tag{4}$$

where g is the constant of gravity.

The reference trajectory for the MPC is the predicted lead vehicle trajectory estimated from a simple dynamics model in (5), which assumes a decaying acceleration of the lead vehicle with time parameter τ_{lead} (1.5s) and dt_p (0.2s)

denotes the step size along the MPC prediction horizon T_{mpc} (2.0s).

$$\begin{aligned} a_{lead} &\leftarrow a_{lead}(t_0)\exp(-\tau_{lead}\hat{t}^2/2) \\ x_{lead} &\leftarrow x_{lead} + v_{lead}\cdot dt_p \\ v_{lead} &\leftarrow v_{lead} + a_{lead}\cdot dt_p \\ \hat{t} &\leftarrow \hat{t} + dt_p \end{aligned} \tag{5}$$

By solving the MPC problem (6) at each time step $\hat{t}$, we can obtain the a_{target} for the low-level controller.

$$\begin{aligned} \min_{a_{target}} \quad & C(\hat{t}) \\ \text{s.t.} \quad & \text{Point-mass dynamics [17] for the ego car} \\ & \text{Predicted dynamics for the lead car in (5)} \\ & v_{target} \geq 0 \end{aligned} \tag{6}$$

It is worth noting that the MPC planner can also use the predicted ego vehicle trajectory as the reference, which is more straightforward but probably needs a vision-based machine learning model. Detailed discussion can be found in [18], [19].

Recall that a linear controller outputs a desired speed value v_{target}, which is all a low-level PI controller needs. However, it is a different case for a low-level PIF controller since it needs both v_{pid} and a_{pid}. Accordingly, except for the MPC solver, the MPC planner involves two more variables $v_{start}(\hat{t})$ and $a_{start}(\hat{t})$, which are updated using the following rule:

$$a_{start} \leftarrow a_{start} + d\hat{t}/dt_p(a_{target} - a_{start}) \tag{7}$$

$$v_{start} \leftarrow v_{start} + d\hat{t}(a_{target} + a_{start})/2 \tag{8}$$

where $d\hat{t}$ denotes the time step for the planner to update, and T_p is the planner step. The usage of v_{start} and a_{start} will be introduced shortly.

C. Longitudinal low-level controllers

As shown in Fig.1, for each step in the low-level loop, the processor $LongControl$ adopts the most recent upper-level planning variables (v_{target} or a_{target}, v_{start} and a_{start}) updated at 20hz and calculates setpoints v_{pid} or a_{pid} for each low-level iteration running at 100 hz. Simultaneously the PI or PIF uses v_{pid} or a_{pid} to output the $control$ variable and feeds it to gas/brake system. For the low-level control loop, it is important to understand how $LongControl$ calculates v_{pid} or a_{pid}. The detailed algorithm is introduced as follows.

If the planner gives a target speed v_{target}, the low-level controller is usually a PI controller. Let $\hat{t}$ denote the planning time where the sensor and planner get updated every 0.05s, and t is the low-level control time with step of 0.01. The algorithm to compute v_{pid} can be found in Openpilot 0.3.5 and the pseudo-code is shown in Algorithm 1 below.

In short, the current $v_{pid}(t)$ in a PI controller updates itself by moving towards the given $v_{target}(\hat{t})$ at a maximum rate bounded by some constraints including the acceleration limits.

Algorithm 1 Algorithm for v_{pid} with a linear planner

Input: v_{target}, $v_{pid}(t)$, $v_{ego}(t)$, a_{max} and a_{min}
Output: $v_{pid}(t+1)$
Initialisation: $v_{pid}(0) = v_{ego}(0)$
constant: overshoot allowance $oa = 2.0$
Start loop to compute $v_{pid}(t)$ **for** $t \in [\hat{t}, t\hat{+}1]$**:**
if $v_{pid} > v_{ego} + oa$ and $v_{target} < v_{pid}$ **then**
  $v_{pid} \leftarrow max(v_{target}, v_{ego} + oa)$
else if $v_{pid} < v_{ego} - oa$ and $v_{target} > v_{pid}$ **then**
  $v_{pid} \leftarrow min(v_{target}, v_{ego} - oa)$
end if
if $v_{target} > v_{pid} + a_{max}\cdot dt$ **then**
  $v_{pid} \leftarrow v_{pid} + a_{max}\cdot dt$
else if $v_{target} < v_{pid} + a_{min}\cdot dt$ **then**
  $v_{pid} \leftarrow v_{pid} + a_{min}\cdot dt$
else
  $v_{pid} \leftarrow v_{target}$
end if

If the upper-planner outputs the desired acceleration a_{target}, then the low-level control loop usually outputs the acceleration setpoint a_{pid} as well as v_{pid}. A feedforward term $k_f \cdot a_{pid}$ is added to form a PIF controller. In Algorithm 2, we show the algorithm for computing a_{pid} and v_{pid} using the a_{target}, v_{start} and a_{start} from a MPC longitudinal planner. Recall the two surrogate variable v_{start} and a_{start} are updated in the MPC planning loop.

Algorithm 2 Algorithm for a_{pid}, v_{pid} with an MPC planner

Input: most recent a_{target}, a_{start}, v_{start}
Output: a_{pid} and v_{pid}
Reset: $v_{start}(0) = v_{ego}(0)$ and $a_{start}(0) = a_{ego}(0)$
Low-level loop for $v_{pid}(t), a_{pid}(t)$ **for** $t \in \{\hat{t}, \hat{t} + 0.01...\hat{t} + d\hat{t}\}$**:**
$dt = t - \hat{t}$
$a_{pid}(t) = a_{start} + dt(a_{target} - a_{start})/dt_p$
$v_{pid}(t) = v_{start} + dt(a_{pid}(t) + a_{start})/2$

Then we introduce how PI/PIF calculate the $control$ input. Define the true speed as v_{ego} at each control time t, the speed error as $e = v_{pid} - v_{ego}$, the formulation of PI/PIF $control$ input is expressed as:

$$control = k_p \cdot e + k_i \cdot \int_0^t e(\tau)d\tau + k_f \cdot a_{pid} \tag{9}$$

where k_f, k_i and k_f correspond to the control gain for proportional(P), integral(I) and feedforward(F) terms. Note that they can all be constants or speed-dependent parameters.

The $compute\_gb$ function then views the $control$ input as a desired acceleration and maps it to the desired gas/brake percentage gb:

$$gb = compute\_gb(control, v_{ego}) \tag{10}$$

Note that brake is negative, and gb ranges from [-1,1]. The gas/brake commands are sent to the control interface of the car, where the dynamics between the gb and the produced acceleration can be represented as the $gb2accel$ function:

$$a_{ego} = gb2accel(gb) \tag{11}$$

We will later show for real cars the $gb2accel$ can be well approximated by a simple scaling function, for example, $a_{ego} = 3gb$, and accordingly the $compute\_gb$ function should share the same scale factor in the denominator, e.g. $gb = a_{ego}/3$. This can be achieved with the help of ACC module on recent car models. Advanced ACC control interfaces can accept scaled accelerations, or gb equivalently, as the input and execute it using some unknown model. For these type of cars, one can view the gb as a scaled acceleration. We consider $gb2accel$ and $compute\_gb$ functions are 'perfect' if they just use a single scale factor and match each other.

III. Methodology

This section investigates the impact of low-level controllers through analytical derivation. Correspondingly, we study the two types of controllers corresponding to linear/MPC planners.

Assuming the $compute\_gb$ and $gb2accel$ functions are "perfect" (i.e., the desired acceleration requested by the $control$ will approximately equal to the true acceleration a_{ego} responses from the car), we have:

$$a_{ego} \approx control = k_p \cdot e + k_i \cdot \int_0^t e(\tau)d\tau + k_f \cdot a_{pid} \tag{12}$$

Then the whole control loop can be explicitly modeled and we can study the underlying interactions between low-level controller and the planner.

A. PI low-level controller and its impact on SS

Consider that the proportional gain has a dominant effect, and in discrete-time implementation the integral gain and derivative gain can all be converted into proportional gain, we omit the integral and derivative term for now:

$$a_{ego} = control \approx k_p(v_{pid} - v_{ego}) \tag{13}$$

For a linear planner as (1) and (2), it satisfies:

$$\begin{aligned} \dot{s}_{lead} &= v_{lead} - v_{ego} = v_{rel} \\ \dot{s}_{des} &= \tau \dot{v}_{lead} \\ \dot{v}_{lead} &= a_{lead} \end{aligned} \tag{14}$$

Substituting (14) into (2), we have:

$$\dot{v}_{target} = k_v(v_{lead} - v_{ego}) + (1 - k_v\tau)a_{lead} \tag{15}$$

To investigate the impact on SS, we evaluate the maximum changes in the ego vehicle target speed, namely Δv_{target}, which can be calculated by tracking (15) from the time T_1 when the ego car starts to react to the lead vehicle, to the time $T_1 + \Delta T$ when the ego car starts to reduce the overshoot/undershoot (see Fig.2(e)):

$$\begin{aligned} \Delta v_{target} &= v_{target}(T_1 + \Delta T) - v_{target}(T_1) \\ &= \sum_{T_1 \le \hat{t} \le T_1 + \Delta T} k_v(v_{lead}(\hat{t}) - v_{ego}(\hat{t}))dt \\ &+ \sum_{T_1 \le \hat{t} \le T_1 + \Delta T} (1 - k_v\tau)a_{lead}(\hat{t})dt \\ &= k_v \sum_{T_1 \le \hat{t} \le T_1 + \Delta T} v_{rel}(\hat{t}) + (1 - k_v\tau)\overline{a}_{lead}\Delta T \end{aligned} \tag{16}$$

where $\overline{a}_{lead}$ is the average acceleration for the lead vehicle during ΔT. Since v_{lead} and a_{lead} do not change with the follower, the magnitude of the speed change for the ego vehicle will only depend on v_{rel}. Also note that $1 - k_v\tau > 0$ for rational values of the gain k_v and desired time headway τ.

Then we subtract the lead vehicle speed change during ΔT, to yield the SS index of linear planner I_{linear}:

$$\begin{aligned} I_{linear} &= (|\Delta v_{target}| - |\Delta v_{lead}|)/|\Delta v_{lead}| \\ &= \frac{k_v}{|\Delta v_{lead}|}\Big(\sum_{T_1 \le \hat{t} \le T_1 + \Delta T} |v_{rel}(\hat{t})| - \tau|\overline{a}_{lead}|\Delta T\Big) \end{aligned} \tag{17}$$

which can be applied to both acceleration and deceleration cases. A smaller I_{linear} is desirable as it corresponds to better SS. If $I_{linear} > 0$ the speed change is amplified from leader to follower, yielding a string unstable case. Although (17) does not directly involve low-level variables, the impact of the low-level controller can be described as follows: for a fixed planner, a slow-response controller moves the follower more slowly, which further makes the relative speed v_{rel} larger, and also takes longer ΔT to stabilize. A faster controller can track the planned speed faster, reduces the speed amplification Δv_{target} and thus improves SS.

The SS index (17) also sheds some light to the impact of the planner. Note that k_v, i.e., how fast the planner react to spacing, and the desired headway τ can affect SS as well. If the planner is more sensitive to spacing change (i.e., with a larger k_v), the ACC system is more likely to be string unstable (i.e., larger Δv_{target} will make the planned trajectory harder to track). For the impact of time headway τ, (17) indicates larger headway can benefit the SS, which is consistent with the recent empirical finding [20] from a few commercial ACC systems. Remarkably, the theoretical analysis of string stability in the frequency domain also achieves the similar findings, i.e. the larger time headway can reduce the $\infty-$norm and $2-$norm of the string stability transfer function [10], [17], [21].

B. PIF low-level controller and its impact on SS

As stated before, a PIF controller works with an MPC planner. Recall Algorithm 2 uses a_{target}, v_{start} and a_{start} to calculate the low-level setpoints $vPid$ and $aPid$.

In (8) we can assume the initial a_{start} equals to zero, thus the a_{start} sequence can be simplified using only a_{target} from the MPC planner, which leads to:

$$a_{start}(\hat{t}) = \sum_{n=0}^{\lfloor \hat{t}/d\hat{t} \rfloor - 1} \frac{d\hat{t}}{dt_p}(1 - d\hat{t}/dt_p)^{\lfloor \hat{t}/d\hat{t} \rfloor - 1 - n} a_{target}(nd\hat{t}) \tag{18}$$

where $\lfloor \hat{t}/d\hat{t} \rfloor$ is a floor function to calculate index for the planning step of $\hat{t}$ starting from zero. Substitute (18) into (8), the changes of v_{start} can be described using a_{target}:

$$2/d\hat{t} \cdot \Delta v_{start} = \sum_{T_1 \le \hat{t} \le T_1 + \Delta T} a_{target}(\hat{t}) + \sum_{n=0}^{\lfloor \hat{t}/d\hat{t} \rfloor} d\hat{t}/dt_p(1 - d\hat{t}/dt_p)^{\lfloor \hat{t}/d\hat{t} \rfloor - 1 - n} a_{target}(nd\hat{t}) \tag{19}$$

Correspondingly, the SS index of the MPC planner I_{mpc} can be derived as:

$$\begin{aligned} I_{mpc} &= (|\Delta v_{start}| - |\Delta v_{lead}|)/|\Delta v_{lead}| \\ &= -1 + \sum_{T_1 \le \hat{t} \le T_1 + \Delta T} |a_{target}(\hat{t})| \frac{d\hat{t}}{2|\Delta v_{lead}|} \\ &\quad + \sum_{n=0}^{\lfloor \hat{t}/d\hat{t} \rfloor} \frac{d\hat{t}(1 - d\hat{t}/dt_p)^{\lfloor \hat{t}/d\hat{t} \rfloor - 1 - n}}{dt_p |\Delta v_{lead}|} |a_{target}(nd\hat{t})| \end{aligned} \tag{20}$$

Similarly, smaller I_{mpc} means better SS. Here for an MPC controller, a slow low-level controller will trigger the planner to produce larger desired acceleration $|a_{target}(nd\hat{t})|$ and thus increase the I_{mpc}, which means the SS is undermined.

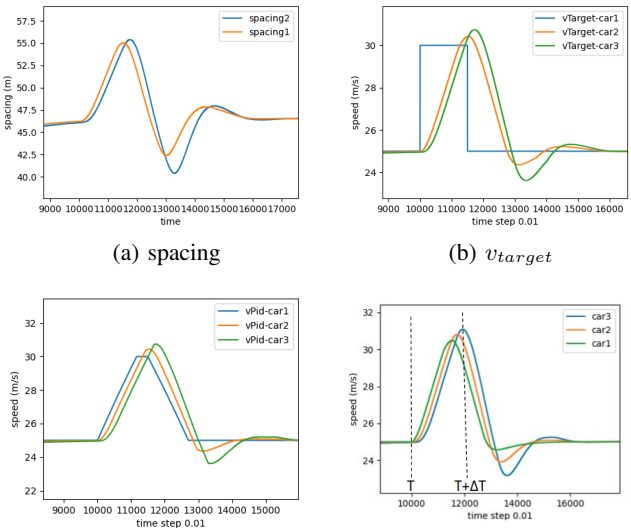
In this section we sought to derive two indexes to measure the SS for the linear+PI and the MPC+PIF ACC systems. The mathematical derivations in (17) and (20) do not have explicit low-level variables yet, but they suggest the same finding; that a fast controller (which enables faster decay of tracking error) can help the SS and vice versa. Note that (17) and (20) also indicate the SS is the outcome of the interactions and coupling effect between the planner and the low-level controller. Note that our indexes are essentially the same as the well-known transfer function used in control theory, but are derived in the time domain rather than the frequency domain. A strict proof is omitted due to space constraints.

IV. Simulation Results

This section reinforces the theoretic findings through numerical simulations running on the same planner and low-level control algorithms. To simulate the actuator system, we assume $gb = \frac{1}{3}control$ and $a_{ego} = 3gb$. This assumption will be justified by data from real cars in the next section.

A. Actuator performance

First we investigate the impact of the gas/brake system (the $gb2accel$ function). While the $compute\_gb$ implies $gb = \frac{1}{3}control$, the true accelerations induced by the gb values are unknown, which could be larger or smaller than $control$. The unknown relationship from gas/brake to true acceleration largely depends on each vehicle's engine and braking performance. While the dynamics might be complex, we can specify whether the actuator is overshooting if $a_{ego} > control$, or undershooting if $a_{ego} < control$. We present the impact of an undershooting actuator and an overshooting actuator on SS of a linear+PI ACC system. The simulation results are shown in Fig.2 and Fig.3, respectively.



(a) spacing (b) v_{target}

(c) v_{pid} (d) speed

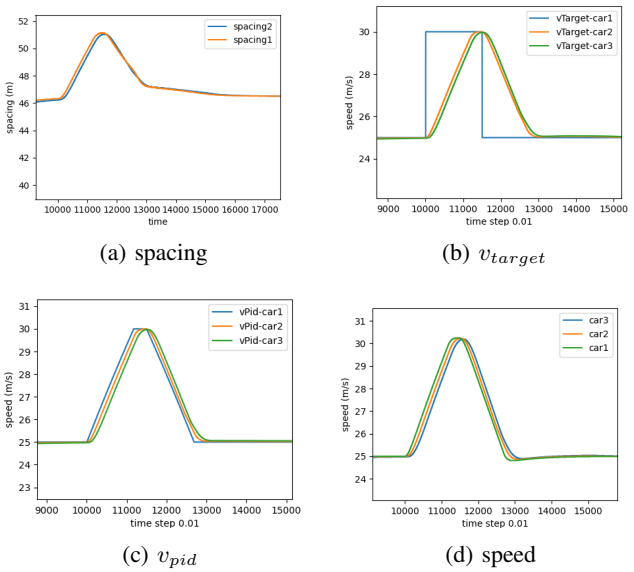
Fig. 2: The impact of an undershooting actuator on SS: the order of the figures is intended to show the cause and effect, but also is a loop.

The overshooting actuator (see Fig.3(e)) achieves significantly better SS than the undershooting actuator (see Fig.2(d)). Also, for the time ΔT that the follower needs to stabilize, Fig.2 and Fig.3 suggest that the slow controller needs longer time. Both findings are consistent with our interpretation of the mathematical derivations (17) and (20).

B. Control gains

Nest, we investigate the impact of the control gains, and assume the actuator system is not undershooting or overshooting. In simulation, it means $gb = control/scale$ and $a_{ego} = scale * gb$ share the same $scale$ parameter. This assumption will be verified with data from real cars.

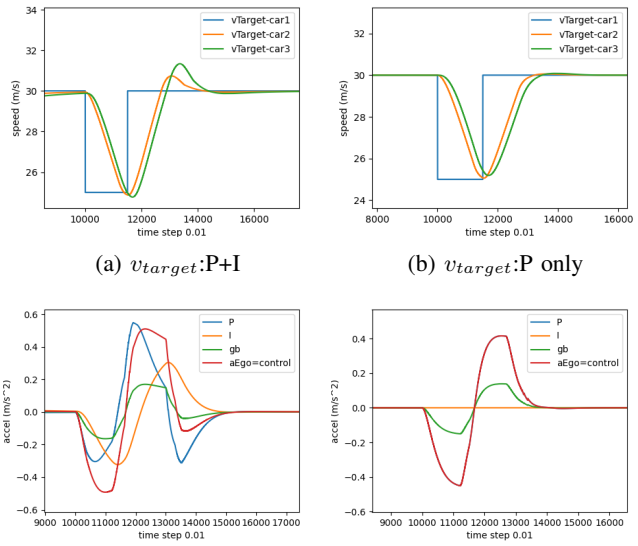
1) Integral gain: While integral (I) gain is designed to reduce steady-state error and is almost indispensable in real-world implementation, the integral accumulation also brings negative impact on the SS. For longitudinal control of vehicles, we notice I gain can: i) sometimes have opposite



(a) spacing (b) v_{target}

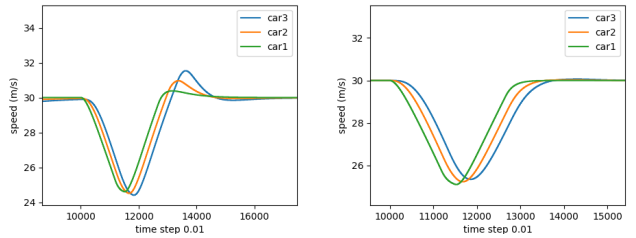
(c) v_{pid} (d) speed

Fig. 3: The impact of an overshooting actuator on SS



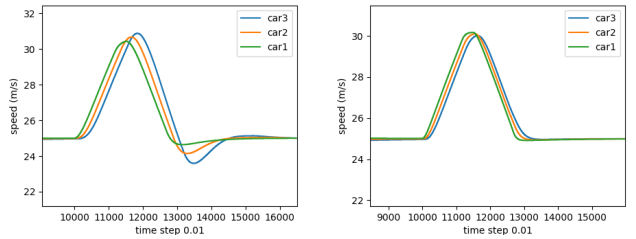
(a) v_{target}:P+I (b) v_{target}:P only

(c) slow control:P+I (d) fast control:P only



(e) overshoot:P+I (f) accurate shoot:P only

Fig. 4: The impact of integral term on SS



(a) small (1*P) gain (b) large (2*P) gain

Fig. 5: Impact of P gain on SS

sign against the proportional term (see Fig.4(c)), which cancels out the proportional control for reducing the error; ii) accumulate quickly with a rapid speed changes in target speed, which can undermine SS; and iii) take long time to settle down. Accordingly, inappropriate I gain can make the controller slower and cause frequent overshoot from the setpoint, as shown in Fig.4(b). Combining the two effects, we see a string unstable platoon in Fig.4(a).

2) Proportional gain: For a controller with only proportional (P) gain, the impact of P gain is similar to the actuator's overshooting or undershooting effects, given the perfect $compute\_gb$ function and the $gb2accel$ introduced in section II. From Fig.4(d), we can see a scaled P gain changes the $control$ in the same way that a scaled $gb2accel$ function does. Thus, it is safe to say increasing P gain could help SS as an overshooting actuator does. Due to space constraints, we briefly show the differences of SS from low-level controllers with small P gain and with large P gain.

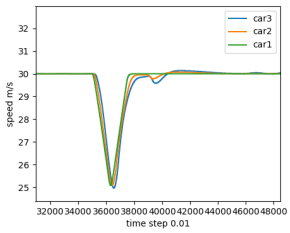
3) Feedforward gain: In a PIF controller, the feedforward term is $K_f \cdot a_{pid}$. Overall P works to reduce speed error and feedforward (F) gain directly provides an acceleration command. Thus, F gain is usually set as 1 and dominates the control input among the three gains in the PIF controller. In the simulation, we executed the MPC+PIF loop and compared the default and doubled F gain, as shown in Fig.6.

The comparison results suggest that tuning up the F gain can also make the low-level controller faster, see Fig.6(b) and (d), which improves the SS; see Fig.6 (a) and (c). We also observed that P and I terms can sometimes work against the F term because a_{pid} and $e = v_{pid} - v_{ego}$ can have different signs. Notice that although $2K_f$ brings better SS, the controller is too sensitive towards errors and creates fluctuations in steady state. Hence, a proper F gain for preventing over-sensitive to small errors is also necessary.

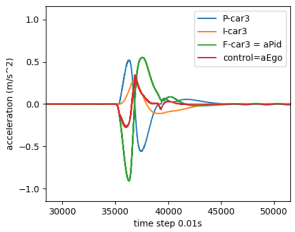
To summarize, in terms of the impact of control gains on the SS, we achieve the similar finding: increasing P and F gain will help SS if they are not too large to cause oscillations. In addition, integral accumulation can lead to overshooting that damages SS.

V. Validation on Real Cars

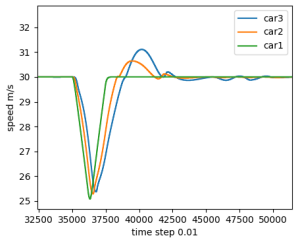
In this section we validate the above findings using data from commercial vehicles running the ACC algorithms. Specifically, we use Comma Two, an after-market device from Comma.ai that can overwrite the stock ACC commands.



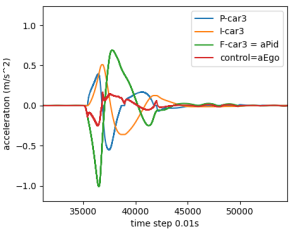
(a) speed with small F



(b) control with small F



(c) speed with large F



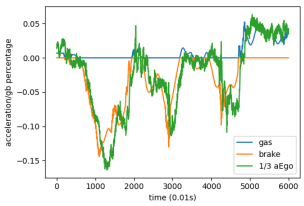
(d) control with large F

Fig. 6: The impact of F gain on SS

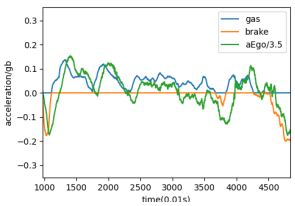
The authors have a custom fork[1] of the Openpilot and have published some branches with the different upper-level planners and low-level controllers mentioned earlier.

A. Relation between gas/brake to acceleration

First we show the data collected from a 2020 Toyota Corolla and a 2019 Honda Civic, as validation for the assumption of a "perfect" *compute_gb* function we used in numerical simulations.



(a) Toyota



(b) Honda

Fig. 7: Relation between gas/brake and acceleration on real cars

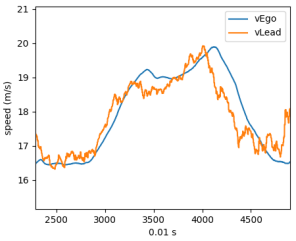
Fig.7 proves that a simple scale function is a good fit to the relationship between gas/brake and true acceleration. Note that we are using market vehicles without any modifications on the hardware system. This means the relationship can be easily transferred. [2]

[1]https://github.com/HaoZhouGT/openpilot: Includes implementation of ACC+PI, MPC+PIF, and IDM+PIF for self-driving longitudinal control

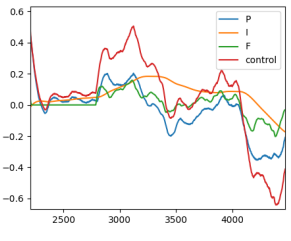
[2]This finding is a bit surprising but may be because modern ACC modules can accept direct acceleration commands as inputs and execute the acceleration in a desired manner. However, we are not sure the *gb* is gas/brake percentage, it might be a scaled desired acceleration for ACC modules to execute. According to Openpilot community [22], at least the new ACC modules from General Motor and Toyota after 2020 are potentially applying such a logic.

B. Validation of the integral impact

Integral term can induce overshooting from a low-level controller to the overshooting of lead vehicle speed and thus undermines the SS. The Fig.8(a) showcases a string unstable example where the follower overshoots the lead vehicle around step 3500, meanwhile the Fig.8(b) clearly shows that integral accumulation (orange curve) is occurring and dominates the control.



(a) integral accumulation (speed)



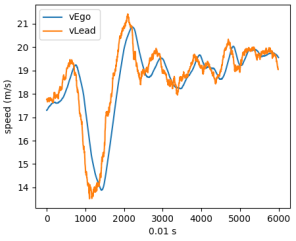
(b) Integral accumulation (control)

Fig. 8: Impact of integral on SS in a real drive

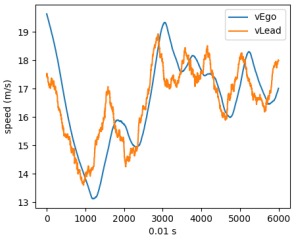
C. Validation of the impact of a fast/slow low-level controller

As indicated earlier, the P or F gains and the gas/brake system share a similar mechanism to affect the SS via the low-level controller. The major finding is that a fast low-level controller improves SS and a slower controller undermines it.

To verify this, we compare two low-level controllers sharing the same planner (MPC) but with different low-level parameters. Specifically, the fast low-level controller uses $1.0K_p$, $0.33K_i$, $1.2K_f$ and an overshooting *compute_gb* defined as $gb = 1/3 \cdot control$. By contrast, the slow low-level controller adopts $0.5K_p$, $0.33K_i$, $1.0K_f$ and an undershooting *compute_gb* defined as $gb = 1/5 \cdot control$. The numbers before control gains are scales of the default values.



(a) fast low-level controller



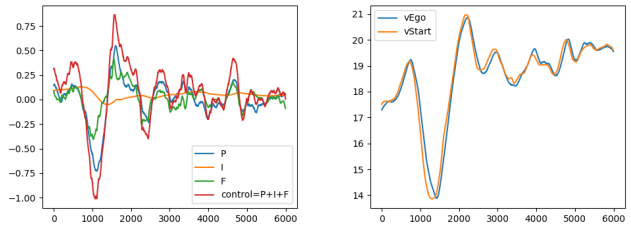
(b) slow low-level controller

Fig. 9: Impact of a fast/slow low-level controller on SS

Fig.9 displays the two real-world drives using a fast/slow low-level controller. The ego vehicle follows a human-driven lead vehicle in natural driving where the lead vehicle changes its speed occasionally on a curvy road. It is apparent that the fast low-level controller is able to dampen the lead speed

changes while the slow low-level controller amplifies them. The results suggests better SS can be obtained from a fast low-level controller.

To show how a fast low-level controller can achieve SS, Fig.10(a) displays the detailed P, I, F terms in the *control* input. As stated before, a more dominanting F term can help SS, and P gain is expected to be consistent with F term, not cancel out its effect. Also we would like to limit the impact of integral overshoot. Additionally, Fig.10(b) shows that v_{start} is close to v_{ego}, which provides support for our method to use v_{start} to investigate the impact on SS in section III.



(a) P, I, F in *control* (b) v_{start} is close to v_{ego}

Fig. 10: A real SS ACC with a fast low-level controller

VI. Concluding Comments and Outlook

The paper introduces, for the first time, open-source commercial ACC algorithms to analyze the impact of low-level controller on SS. It finds a fast-response low-level controller can improve SS while a slow-response controller can undermine it despite having a string stable planner. Accordingly, we infer that it would be more beneficial to focus efforts on improving the low-level controller rather than designing more string stable ACC/CACC models which have become available in some recent cars.

We suggest focusing future research efforts on: i) novel design to prevent the integral overshoot in low-level control; ii) better understanding the MPC controller design and leveraging traditional CF models as alternatives and iii) fine-tuning of a fast low-level controller on real cars that balances the SS, safety and driving comfort.

Acknowledgment

The authors would like to acknowledge the warm help from Joe Pancotti for his commit in modifying the Honda Bosch interface [23]. We also appreciate the helpful discussions with Shane Smiskol and csouers at Openpilot's community.

References

[1] E. Shaw and J. K. Hedrick, "String stability analysis for heterogeneous vehicle strings," in *2007 American Control Conference*, 2007, pp. 3118–3125.

[2] S. Feng, Y. Zhang, S. Li, Z. Cao, H. Liu, and L. Li, "String stability for vehicular platoon control: Definitions and analysis methods," *Annu. Rev. Control.*, vol. 47, pp. 81–97, 2019.

[3] G. J. L. Naus, R. P. A. Vugts, J. Ploeg, M. J. G. van de Molengraft, and M. Steinbuch, "String-stable cacc design and experimental validation: A frequency-domain approach," *IEEE Transactions on Vehicular Technology*, vol. 59, no. 9, pp. 4268–4279, 2010.

[4] J. Zhou and H. Peng, "Range policy of adaptive cruise control vehicles for improved flow stability and string stability," *IEEE Transactions on intelligent transportation systems*, vol. 6, no. 2, pp. 229–237, 2005.

[5] G. Gunter, C. Janssen, W. Barbour, R. E. Stern, and D. B. Work, "Model-based string stability of adaptive cruise control systems using field data," *IEEE Transactions on Intelligent Vehicles*, vol. 5, no. 1, pp. 90–99, 2019.

[6] M. Makridis, K. Mattas, B. Ciuffo, F. Re, A. Kriston, F. Minarini, and G. Rognelund, "Empirical study on the properties of adaptive cruise control systems and their impact on traffic flow and string stability," *Transportation research record*, vol. 2674, no. 4, pp. 471–484, 2020.

[7] D. Yanakiev and I. Kanellakopoulos, "Variable time headway for string stability of automated heavy-duty vehicles," in *Proceedings of 1995 34th IEEE Conference on Decision and Control*, vol. 4. IEEE, 1995, pp. 4077–4081.

[8] C.-Y. Liang and H. Peng, "Optimal adaptive cruise control with guaranteed string stability," *Vehicle system dynamics*, vol. 32, no. 4-5, pp. 313–330, 1999.

[9] C. Liang and H. Peng, "String stability analysis of adaptive cruise controlled vehicles," *Jsme International Journal Series C-mechanical Systems Machine Elements and Manufacturing*, vol. 43, pp. 671–677, 2000.

[10] G. J. Naus, R. P. Vugts, J. Ploeg, M. J. van De Molengraft, and M. Steinbuch, "String-stable cacc design and experimental validation: A frequency-domain approach," *IEEE Transactions on vehicular technology*, vol. 59, no. 9, pp. 4268–4279, 2010.

[11] V. Milanés, S. E. Shladover, J. Spring, C. Nowakowski, H. Kawazoe, and M. Nakamura, "Cooperative adaptive cruise control in real traffic situations," *IEEE Transactions on intelligent transportation systems*, vol. 15, no. 1, pp. 296–305, 2013.

[12] G. Gunter, D. Gloudemans, R. E. Stern, S. McQuade, R. Bhadani, M. Bunting, M. L. Delle Monache, R. Lysecky, B. Seibold, J. Sprinkle *et al.*, "Are commercially implemented adaptive cruise control systems string stable?" *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[13] V. Milanés and S. E. Shladover, "Modeling cooperative and autonomous adaptive cruise control dynamic responses using experimental data," *Transportation Research Part C: Emerging Technologies*, vol. 48, pp. 285–300, 2014.

[14] Comma.ai. Comma.ai – introducing openpilot. [Online]. Available: https://comma.ai/

[15] Y. Zhou and S. Ahn, "Robust local and string stability for a decentralized car following control strategy for connected automated vehicles," *Transportation Research Part B: Methodological*, vol. 125, pp. 175–196, 2019.

[16] M. Diehl, "Toolkit for automatic control and dynamic optimization." [Online]. Available: https://acado.github.io/

[17] S. Gong, A. Zhou, and S. Peeta, "Cooperative adaptive cruise control for a platoon of connected and autonomous vehicles considering dynamic information flow topology," *Transportation Research Record*, vol. 2673, no. 10, pp. 185–198, 2019.

[18] H. Zhou, "Details of the mpc for longitudinal control without a neural network," https://howardchow92.medium.com/, 2020.

[19] ——, "The mpc designed for end-to-end longitudinal self-driving at openpilot, comma.ai." [Online]. Available: https://howardchow92.medium.com/

[20] T. Li, D. Chen, H. Zhou, J. Laval, and Y. Xie, "Car-following behavior characteristics of adaptive cruise control vehicles based on empirical experiments," *Transportation Research Part B: Methodological*, vol. 147, pp. 67–91, 2021.

[21] S. Klinge and R. H. Middleton, "Time headway requirements for string stability of homogeneous linear unidirectionally connected systems," in *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, 2009, pp. 1992–1997.

[22] S. Smiskol, "Discussion on the "compute_gb" function in openpilot community," 2021.

[23] J. Pancotti, "The git commit that makes honda bosch vehicle to have a vision-based openpilot control." [Online]. Available: https://github.com/jpancotti/openpilot/commit/468b89a