Platooning as a Service of Autonomous Vehicles

Duo Lu
Arizona State University
Tempe, Arizona
duolu@asu.edu

Zhichao Li
University of California, San Diego
San Diego, California
zhl355@eng.ucsd.edu

Dijiang Huang
Arizona State University
Tempe, Arizona
dijiang.huang@asu.edu

***Abstract*—Smart vehicles equipped with computers and wireless communication devices are emerging on the road. These vehicles can drive themselves, communicate to other vehicles, connect to the Internet, and provide value-added services to the drivers and passengers. With the advent of such technology, it is possible to form a "platoon" of autonomous vehicles on the road, where they follow a common leader vehicle in the same lane on the highway and maintain close proximity to save fuel, improve road capacity and passenger comfort. However, realizing such vision faces difficulties on both software architecture and vehicle control. In this paper, we propose a service-oriented perspective for the software modules on the autonomous vehicles, where platooning is designed as an independent service interacting with other components of the vehicle. We also built a prototype system with low cost vehicle-like mobile robots and ran experiments to demonstrate the effectiveness of our service framework and our platooning control algorithm. Our hope is that the platooning as a service approach can help in the construction of more efficient, interoperable, and secure autonomous vehicles in the future.**

I. Introduction

With the penetration of electronics and software in the automobile industry, vehicles are equipped with onboard sensing, computing, and communication devices to achieve a certain level of autonomy. Such technologies enable these smart vehicles to drive in a platooning pattern on the highway, where a tightly spaced string of vehicles follow the same leader, maintain close proximity, communicate with each other, and move under fully automated longitudinal and lateral control [1]. Platooning brings many benefits. For example, reduced air drag for the vehicles inside a platoon provides better fuel efficiency, especially for heavy-duty trucks [2]. Also, reduced inter-vehicle distance and speed disturbance of the traffic help to achieve both higher road capacity and more comfortable travel experience (*e.g.,* less travel time, less fatigue from driving, higher safety, etc.). Moreover, emerging autonomous vehicles have the capability of sensing the surrounding environment and drive themselves. As a result, platooning software can be installed on such vehicles as a value-added service.

However, a platoon of vehicles is a complex distributed cyber-physical system [3], which raises a few challenges due to its unique characteristics. First, platooning based Cooperative Adaptive Cruise Control (CACC) has its inherent complexity due to the tight integration of sensing, communication, and control, especially when both longitudinal and lateral control are considered together [4]. With the need of handling heterogeneous dynamics, changing road and weather conditions, such a platooning controller is difficult to design [5]. Second, current vehicles from different vendors lack interoperability in network protocol and software. Third, the platooning software faces issues in security and trust, especially when the platoon software needs to steer the vehicle. Fourth, the effectiveness of platooning relies heavily on the performance and reliability of the sensors and vehicle-to-vehicle (V2V) communication. Thus the platooning software must be able to tolerate sensor noise, anomalies, communication delay and message loss.

978-1-5386-2723-5/17/$31.00 ©2017 IEEE

In the past, vehicle platooning has been studied extensively from both control perspective [6], [7] and network perspective [8], [9]. These works build the foundation of the construction of platoon-based system. One critical requirement of designing such a system is to maintain platoon string stability [10] through control algorithm [11], [12] and spacing policy [13], [14]. Other requirements include efficient platoon management [15], [16] and real-time information dissemination. Nevertheless, it is rarely studied from a software architecture's point of view. For the value-added service such as platooning, an architecture transition from current vehicle to future smart vehicle is required — a transition from specially developed embedded system with fixed function to a unified software framework capable of integrating various services, just like the transition from featured phone to smartphone.

In this paper, we propose a service-oriented perspective for the software modules on the autonomous vehicles to partially address the challenges. Our major contribution is the concept and practice of designing the platooning software as an independently deployable microservice [17], [18] encapsulated inside a container (*e.g.,* application or OS level container, partitioned system, or virtual machine), which interacts with other vehicle software components such as vehicle cruise controller through a secure message based service framework. This framework reduces the complexity and improves the security as well as the interoperability of the vehicle software. Our platooning service adds a state estimation module decoupling the network part and the controller part, so as to minimize the influence of small sporadic failure of V2V communication and sensor anomalies. In addition, we implemented the whole system with custom-made low cost mobile robot vehicles to demonstrate the effectiveness of our platooning-as-a-service framework.

The rest of the paper is organized as follows. We propose our framework in section 2. Then we demonstrate how to engineer such a platooning service with the framework by

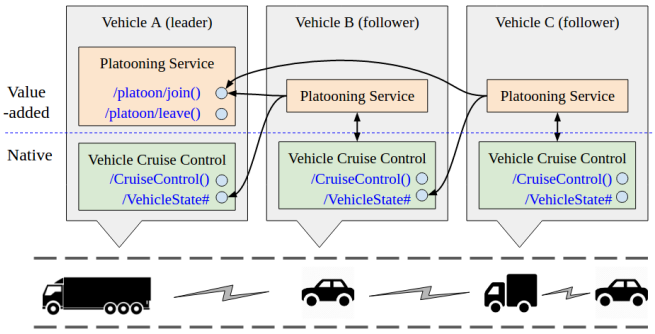

Fig. 1. Platooning system software architecture

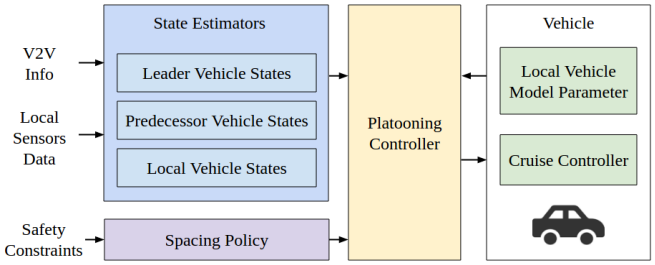

Fig. 2. Platooning service structure

an example implementation in section 3. At last, we draw conclusion and discuss future works in section 4.

II. System Model

A. Software Architecture

In our system, each vehicle can be regarded as a mobile computer hosting many software modules for various functions. Each software module is an independent containerized microservice, providing well-named service API through a message based protocol. Such service API can be a method for remote-procedure-call (RPC) from other modules, or an event channel for publishing events to other subscribed modules. For example, a smart vehicle may have two microservices: a value-added service for platooning, and a native service for vehicle cruise control, shown in Fig. 1. The cruise control microservice provides a `CruiseControl()` method accepting control commands including desired driving speed (controlling how fast the vehicle drives) and steering angle (controlling how hard the vehicle turns), which resembles the function of acceleration pad and the steering wheel. Other software components, like the platooning microservice, can take over the control of the vehicle by calling this method. Besides, the cruise control microservice also provides an event channel for publishing vehicle states regularly, so that other software components can obtain important vehicle information such as the current driving speed.

All these interfaces have well-defined message formats, semantics, and service level specification, detailing their behavior, and the messages are routed through an API gateway. The API gateway enforces authentication, access control, and auditing. For example, only trusted components, such as the platooning microservice, can take over the control of the vehicle, while an infotainment service may not have such permission. In addition, the API gateway may expose certain service APIs to the external requesters through the similar message interface over wireless communication, *e.g.,* in a platoon, a follower vehicle may subscribe the leader and its immediate predecessor's `VehicleState#` event channel to obtain their speed and acceleration information.

Decoupling the vehicle software into multiple microservices and putting each of them into a container with only message interface reduces system complexity and improves security. In this way, each software module can be individually isolated to guarantee its required computing resources, and hence, its performance as well as real-time characteristics. Moreover, it is easier to individually develop, verify, test, and certify each individual microservice for the software provider, and it is more flexible for the user to purchase and install value-added services. For example, the user can install a third-party developed "speeding warning" service, which obtains the current driving speed through the `VehicleState#` event channel and provides warning according to the stored information of speed limit with the map.

Multiple vehicles are connected by a vehicular ad hoc network (VANET) on the road, which enable their V2V communication. Each vehicle is an independent site, like website, hosting its services, where other vehicles can setup direct connection with it (one hop), and make use of the services. For example, such a connection may be based on DSRC/WAVE, and the messages may be delivered through WAVE Short Message Protocol (WSMP) or IPv6 over DSRC/WAVE. The events for publishing vehicle states (*i.e.,* the beacon messages) can be broadcast utilizing the inherent characteristics of wireless communication, while the receiving vehicle delivers such message only to the subscribed software components.

A platoon of vehicles is a self-managed system where the goals are (1) maintaining the same speed as the leader, (2) keeping a certain separation distance according to safety policies, and (3) driving within the lane. In our model, any vehicle equipped with platooning service can be a leader. If no one follows it, it is a trivial platoon of only the leader vehicle. Any other vehicle can join a platoon at the end of the string, and any vehicle in the platoon can leave, by coordinating with the leader vehicle. The leader vehicle can be independently driven (manually or automatically), and other vehicles in the platoon are driven by the platooning controller. For example, assume a platoon of vehicles driving in the HOV lane on a highway, an adjacent vehicle can catch up the vehicle cluster and join them by entering the HOV lane, following the last vehicle in the platoon, calling the leader's `/platoon/join()` method, waiting for confirmation, and switching from free-driving mode to platooning mode. If a vehicle in the platoon needs to leave, it calls the leader's `/platoon/leave()` method, waits until the leader finishes coordination with all other vehicles in the platoon, then it switches to free-driving mode and leaves the HOV lane.

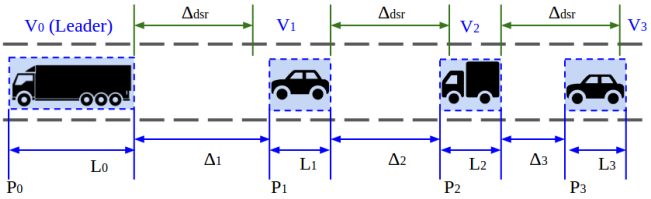

Fig. 3. Platoon model

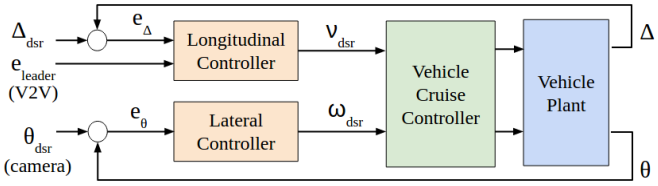

Fig. 4. Platooning controller structure

The platooning service contains three major components shown in Fig. 2.

First, a component estimates the motion states of the leader, the immediate predecessor, and itself by fusing the data from both local sensors and V2V communication. This module decouples controller from the network and improves controller robustness against sensor anomaly and communication loss. For example, a typical platooning follower receives the position, speed, and acceleration of the leader and its predecessor from V2V communication, and it measures the predecessor's speed and the separation distance to it. In this way, it can run a Kalman filter to fuse the received data and the measured data to derive more accurate estimation of their states. Even if a few beacon messages are lost, the leader's states can be inferred by assuming the acceleration would not rapidly change.

Second, the platooning microservice uses a spacing policy to determine the optimal separation distance to the predecessor vehicle, according to the vehicle dynamics, the map, weather, road condition, and communication quality. For example, a heavy-duty truck would need more separation distance to a sedan than another heavy-duty truck because the larger difference of dynamic model (and hence, difference in distance to brake). Similarly larger separation distance may be required in mountainous areas or in the rain due to poor wireless communication. Changes of desired separation distance are made slowly to avoid influence on the string stability.

Third, a platooning controller is utilized to drive the vehicle autonomously in a platoon. In our system, the longitudinal control algorithm is a two-path PID controller with predecessor-leader strategy, which takes input of the estimated states and expected separation space, and output the desired driving speed to the cruise controller. The lateral control algorithm is a simple PID controller which uses cameras and other sensors to track the lane markers and output expected steering angle to keep the vehicle in the lane. The controller parameters are determined at installation based on the local vehicle dynamic model.

B. Platoon Model and Control Algorithm

In this paper we mainly focus on longitudinal control on the highway, so we ignore the case of junctions and treat the road as a straight line for simplicity. The platoon model is shown in Fig. 3, where each vehicle V is a rigid body with a bounding box, with a few physical states, including its length L, position (x, y), offset on a road segment P, orientation θ, longitudinal speed v, yaw speed ω (*i.e.,* the steering angle), and acceleration a. Collectively, these physical states are defined as s and broadcast in the beacon messages. Each vehicle is also able to measure the separation distance to its immediate predecessor using its own onboard sensors, denoted as Δ, *i.e.,* Δ_i is the distance between V_{i-1} and V_i, where the leader is V_0. As mentioned above, the spacing policy will generate a desired separation distance Δ_{dsr}, which should be maintained by the follower.

Our platooning controller decouples longitudinal control and lateral control for simplicity (shown in Fig. 4). The longitudinal controller of the ith vehicle takes leader's states s_0, immediate predecessor's states s_{i-1}, and the sensor measurement Δ_i as input. It contains two PID paths, one uses speed difference from the leader (the feed-forward-path) and the other uses distance difference from the desired value (the Δ-path), as shown in Algorithm 1, then it outputs the desired driving speed (v_{dsr}). During the calculation of error in the feed-forward path, it also fuses the differential term of speed difference from the leader with the acceleration difference of the leader. Similar strategy is also applied to the differential term of separation distance in the Δ path. For lateral control, cameras or other sensors are used to track the lane markers and derive the desired vehicle orientation θ_{dsr}, then the error is calculated together with current vehicle orientation θ, and a simple PID controller is used to obtain the desired turning speed ω_{dsr} from the error. In real world scenario, it is possible that the lane makers would be blocked by predecessor vehicles if the separation distance is small, so the leader might need to broadcast the lane curvature for a certain distance in the beacon messages to other vehicles in the platoon. After that, the desired driving speed and steering angle are sent to the vehicle cruise controller to drive the vehicle.

The longitudinal controller should also satisfy the following constraints [19] to maintain string stability, *i.e.,* to prevent disturbance propagation in the platoon.

1) $\|H(s)\|_\infty \leq 1$
2) $h(t) > 0$

where H(s) is the steady state error transfer function, *i.e.,* $e_i = \Delta_i - \Delta_{dsr}$, $h(t) = e_i(t)/e_{i-1}(t)$, $H(s) = \mathcal{L}(h(t))$. This can be achieved in our system by adjusting the platooning controller parameters according to the dynamic model of predecessor when joining the platoon. In our implementation, our robot vehicle together with cruise controller can be approximately modeled as a second order linear time-invariant (LTI) system (detailed in the next section). With careful choice of the PID parameters of the platooning controller, we can guarantee

Algorithm 1: platooning_longitudinal_loop()

input : Δ_{dsr}
output: v_{dsr}
$\Delta \leftarrow$ get_distance_to_predecessor().
$v \leftarrow$ get_my_speed().
$a \leftarrow$ get_my_acceleration().
$v_{leader} \leftarrow$ get_leader_speed().
$a_{leader} \leftarrow$ get_leader_acceleration().
$v_{pre} \leftarrow$ get_predecessor_speed().
$e_{ff} \leftarrow v_{leader} - v$.
$e'_{ff} \leftarrow e'_{ff} + e_{ff}$.
$e''_{ff} \leftarrow$fuse(diff(e_{ff}), $a_{leader} - a$).
$v_{ff} \leftarrow$PID($e_{ff}, e'_{ff}, e''_{ff}$).
$e_{\Delta} \leftarrow \Delta_{dsr} - \Delta$.
$e'_{\Delta} \leftarrow e'_{\Delta} + e_{\Delta}$.
$e''_{\Delta} \leftarrow$fuse(diff(e_{Δ}), $v_{pre} - v$).
$v_{\Delta} \leftarrow$PID($e_{\Delta}, e'_{\Delta}, e''_{\Delta}$).
$v_{dsr} \leftarrow v_{ff} + v_{\Delta}$.
return v_{dsr}.

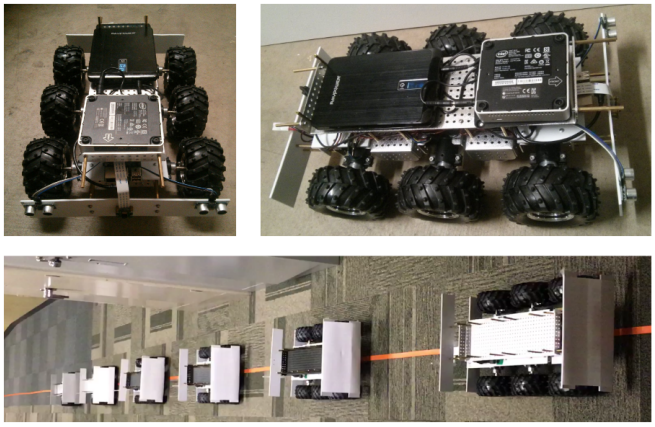

Fig. 5. VC-truck and experimental platooning system.

string stability with a certain range of desired separation distance and leader behavior.

III. Evaluation

A. Testbed

We implemented our platooning system on VC-bots testbed [20] with a secure microservice framework [21]. VC-bots is a vehicular cloud computing [22] testbed constructed with several types of mobile robot vehicles for testing and evaluating vehicular network protocols and vehicular cloud applications. In our implementation, we use six VC-truck robot vehicles, shown in Fig 5. VC-truck has an onboard computer as its local cloud resource, which provides virtual machines as a secure on-demand execution environment for the platooning service. Each VC-truck has multiple WiFi adapters running at different channels, and one adapter is configured in access point mode to allow connections from other robot vehicles, while other adapters are configured in end point mode to initiate connection to other robot vehicles. Above WiFi, we implemented a custom UDP based message protocol to provide services over V2V communication.

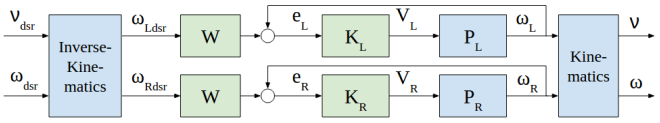

Fig. 6. Cruise controller structure.

The VC-truck is a six wheel differential drive robot vehicle with geared DC motors. Details of the modeling and controller design can be found in [23]. Although the robot we use is different from that in [23], the model is similar with minor refinement. In this paper, we modeled its dynamics as a two-input-two-output LTI system, and approximated using two decoupled single-input-single-output first order LTI systems at low frequency, shown in Fig. 6. The two inputs are motor voltages (V_L, V_R) on the left and right sides (which is equivalent to the duty cycles of motor PWM signals at fixed voltage), and the two outputs are wheel angular velocities (ω_R, ω_L). The transfer function of decoupled and approximated system is shown as following:

$$P_L = P_R = \frac{\omega_L}{V_L} = \frac{\omega_R}{V_R} = \alpha \left(\frac{\beta}{s+\beta} \right)$$

where α and β are model parameters. For VC-truck, $\alpha = 1.129$, $\beta = 5.932$.

Based on this model, we designed a PI controller with roll-off and pre-filter (called the inner loop controller) to drive the wheels, as follows,

$$K_L = K_R = \frac{g(s+z)}{s} \left(\frac{N_1}{s+N_1} \right)$$

where g and z are chosen carefully to achieve a settling time around 0.5 second and nearly no overshoot. For VC-truck, $g = 1.1$, $z = 7$, and the roll-off coefficient $N_1 = 100$, which gives the following controller parameters: $K_p = 1.1$, $K_i = 7.7$. The following pre-filter will ensure the response to a step reference command approximates the result of the original model.

$$W = \frac{z}{s+z}$$

The cruise controller translates longitudinal speed v and yaw speed ω to wheel angular velocities (ω_R, ω_L) through kinematics, and utilize the inner loop controller to drive the vehicle. It runs at 20 Hz.

VC-truck is equipped with various sensors which are able to obtain all the physical states used by the controller. For example, beacon based indoor positioning system as well as onboard LIDAR sensor are used to track its location, rotary encoders are used to measure the wheel speed, and inertial measurement unit is used to obtain the acceleration, yaw speed, and orientation. Besides, two ultrasonic rangefinders

are mounted in the front of the robot vehicle to measure the separation distance to the previous robot vehicle. Moreover, we place tapes on the ground as lane markers and use a front camera on the robot vehicle to detect them for lateral control.

B. Platooning System Setup

In our experimental platooning system, we use six VC-truck to form a platoon, driving on a nearly straight line, with constant spacing policy (0.7m separation distance). Each vehicle has direct WiFi connection with the leader and its immediate predecessor. The leader starts from stationary state and drives at a constant speed (0.2 m/s). All followers have already joined the leader's platoon before the starts and they drive automatically by the platooning controller. The platooning controller we implemented is a PI controller for Δ-path and a PI controller for feedforward-path, shown as follows,

$$K = K_\Delta + K_{ff} = \frac{g_\Delta (s+z_\Delta)^2 \times N_2^2}{s(s+N_2)^2} + \frac{g_{ff}(s+z_{ff}) \times N_3}{s(s+N_3)}$$

For our platoon system of VC-trucks, $g_\Delta = 1$, $z_\Delta = 0.5$, $g_{ff} = 0.4$, $z_{ff} = 1.5$, $N_2 = N_3 = 100$. This gives us the following controller parameters: $Kp_\Delta = 1$, $Ki_\Delta = 0.5$, $Kp_{ff} = 0.4$, $Ki_{ff} = 0.6$. The platooning controller runs at 10 Hz, independently from the cruise controller, and the platoon beacon messages are sent at 20 Hz, synchronized with the cruise controller. Given the vehicle model, cruise controller and platooning controller of VC-truck, careful selection of these PID parameters can satisfy string stable condition.

During the implementation, we found that our low cost sensors including the ultrasonic range finder and the inertial sensor are sensitive to noise, which makes the controller very difficult to tweak when the differential term is applied, so we just drop that term and use only PI controller instead. Also due to the sensor capability, separation distance measurement is very unreliable when the lane is curved, which limits our experiment on a straight lane.

C. Empirical Results

The empirical results of vehicle speed and separation distance is shown in Fig. 7, which is in accordance with simulation results shown in Fig. 8. At the beginning, all robot vehicles are placed with the desired separation distance (0.7m). During the experiment, all followers can catch up with the leader's sudden start and settle at stable states within 20 seconds. At stable states, there will be small speed and separation distance disturbance, which will not be amplified through the vehicle string. We also demonstrate the situation when the leader applies a hard brake towards the end of the experiment (not simulated), as a case of the reaction under emergency condition. In this situation, the speed of the leader suddenly decreases to zero, and all followers are able to brake accordingly to avoid collision. It should be noted that in our experiment, the leader does not send out an explicit hard brake message but only its states, and the followers use the same platooning controller to stop the vehicles. Thus, each follower

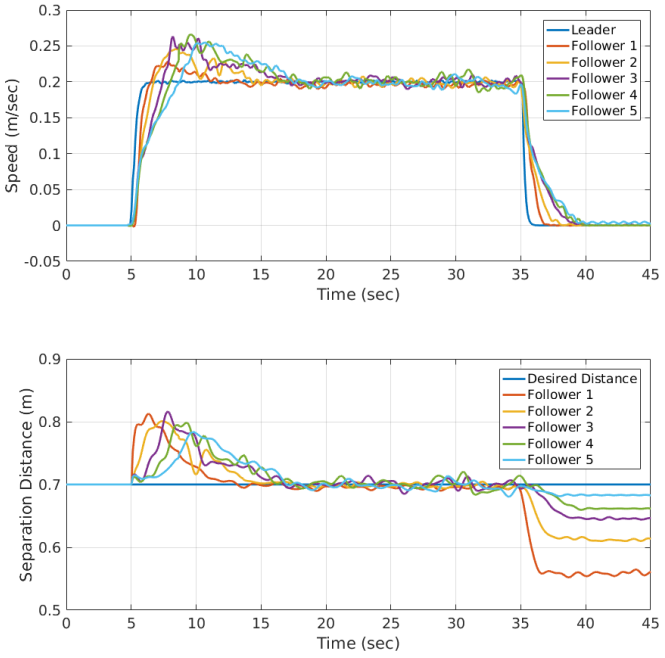

Fig. 7. Empirical results of speed (upper) and separation distance (lower)

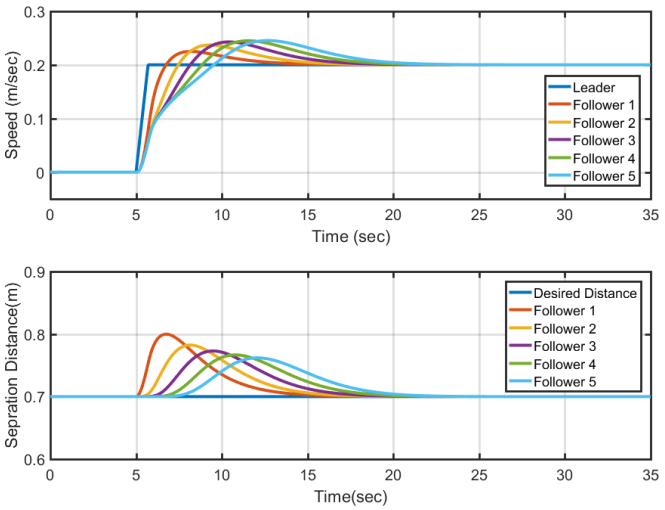

Fig. 8. Simulation results of speed (upper) and separation distance (lower)

does not brake hard enough than a synchronized hard brake of the whole platoon. As a result, the separation distance at stop will be shorter than desire. This also provides us information on the choice of separation distance to avoid collision.

In Fig. 9. we show the decomposition of platooning controller output of the two paths for the first follower vehicle. We can trade off the impact of these two paths with different set of controller parameters, depending on whether more focus is spent on dynamic performance (*i.e.,* quicker speed response but larger separation distance oscillation) or stability (*i.e.,* slower speed response but smaller separation distance oscillation). In general, a small g_Δ results in smaller separation distance oscillation as well as longer time to converge, but Δ-

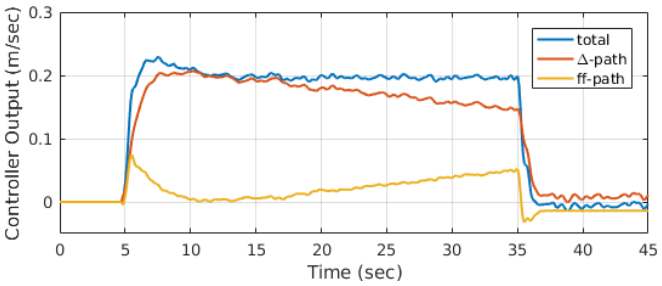

Fig. 9. Decomposition of platooning controller output for the two paths

path itself can not guarantee stability through the whole platoon. With the feed-forward-path, both dynamic performance and stability are improved significantly, but careful setting of z_{ff} is necessary when the platoon is long.

During the experiment, we did not observe significant influence from V2V communication quality. The main reason is the low bandwidth consumption of the communication (beacon message only costs 0.8 KB/s per vehicle when sending at 20 Hz). As a result, we can keep the round trip time of a V2V communication message within 3 milliseconds, which satisfies most cooperative control applications whose controller runs at less than 50 Hz. These requirements can be easily satisfied with real world DSRC/WAVE based vehicular network. Also our state estimation module can tolerate most irregular message loss as well as sensor anomalies, which further looses the requirements. It should be noted that our purpose of this implementation is demonstration the effectiveness of our framework, rather than seeking optimal controller algorithms. Our simplified vehicle dynamic model and WiFi based vehicular network would definitely have limitation when scaled to real world vehicles. Similarly the performance our PID based platooning controller is also limited compared to more advanced controller.

IV. Conclusion and Future Work

In this paper, we propose a service-oriented perspective for the software architecture on autonomous vehicles, where platooning is a value-added service independently deployed in its container and interacting with other components of the vehicle as well as other vehicles through message based service interface. To demonstrate our idea, we designed and implemented a platoon of low cost robot vehicles with a secure microservice framework. Our experiments show the effectiveness of our model and algorithm to autonomously drive a platoon of vehicles, while keeping small separation distance and tolerating speed disturbance.

Due to the hardware and space constraints, we can only test the simplified case using identical vehicles driving on a straight line. Although DSRC/WAVE has similar theoretical performance as WiFi, in real world application, communication quality would be largely influenced by the local terrain and platoon length. Thus, in the future, we will upgrade our robot testbed and conduct experiments with more complicated conditions to thoroughly evaluate our system.

References

[1] L. Li and F.-Y. Wang, *Advanced motion control and sensing for intelligent vehicles*. Springer Science & Business Media, 2007.

[2] A. Alam, B. Besselink, V. Turri, J. Martensson, and K. H. Johansson, "Heavy-duty vehicle platooning for sustainable freight transportation: A cooperative method to enhance safety and efficiency," *IEEE Control Systems*, vol. 35, no. 6, pp. 34–56, 2015.

[3] D. Jia, K. Lu, J. Wang, X. Zhang, and X. Shen, "A survey on platoon-based vehicular cyber-physical systems," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 263–284, 2016.

[4] R. Kianfar, M. Ali, P. Falcone, and J. Fredriksson, "Combined longitudinal and lateral control design for string stable vehicle platooning within a designated lane," in *Intelligent Transportation Systems (ITSC), 2014 IEEE 17th International Conference on*. IEEE, 2014, pp. 1003–1008.

[5] E. Shaw and J. K. Hedrick, "String stability analysis for heterogeneous vehicle strings," in *American Control Conference, 2007. ACC'07*. IEEE, 2007, pp. 3118–3125.

[6] 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.

[7] P. Fernandes and U. Nunes, "Platooning with IVC-enabled autonomous vehicles: Strategies to mitigate communication delays, improve safety and traffic flow," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 1, pp. 91–106, 2012.

[8] D. Jia, K. Lu, and J. Wang, "On the network connectivity of platoon-based vehicular cyber-physical systems," *Transportation Research Part C: Emerging Technologies*, vol. 40, pp. 215–230, 2014.

[9] A. Vinel, L. Lan, and N. Lyamin, "Vehicle-to-vehicle communication in C-ACC/platooning scenarios," *IEEE Communications Magazine*, vol. 53, no. 8, pp. 192–197, 2015.

[10] P. Seiler, A. Pant, and K. Hedrick, "Disturbance propagation in vehicle strings," *IEEE Transactions on automatic control*, vol. 49, no. 10, pp. 1835–1842, 2004.

[11] L. Xiao and F. Gao, "Practical string stability of platoon of adaptive cruise control vehicles," *IEEE Transactions on intelligent transportation systems*, vol. 12, no. 4, pp. 1184–1194, 2011.

[12] J. Ploeg, N. Van De Wouw, and H. Nijmeijer, "Lp string stability of cascaded systems: Application to vehicle platooning," *IEEE Transactions on Control Systems Technology*, vol. 22, no. 2, pp. 786–793, 2014.

[13] 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.

[14] J. Zhao, M. Oya, and A. El Kamel, "A safety spacing policy and its impact on highway traffic flow," in *Intelligent Vehicles Symposium, 2009 IEEE*. IEEE, 2009, pp. 960–965.

[15] A. Uchikawa, R. Hatori, T. Kuroki, and H. Shigeno, "Filter multicast: a dynamic platooning management method," in *Consumer Communications and Networking Conference (CCNC), 2010 7th IEEE*. IEEE, 2010, pp. 1–5.

[16] M. Segata, B. Bloessl, S. Joerer, F. Dressler, and R. L. Cigno, "Supporting platooning maneuvers through IVC: An initial protocol analysis for the join maneuver," in *Wireless On-demand Network Systems and Services (WONS), 2014 11th Annual Conference on*. IEEE, 2014, pp. 130–137.

[17] J. Lewis and M. Fowler, "Microservices," 2014.

[18] S. Newman, *Building Microservices*. " O'Reilly Media, Inc.", 2015.

[19] S. Sheikholeslam and C. A. Desoer, "Longitudinal control of a platoon of vehicles," in *American Control Conference, 1990*. IEEE, 1990, pp. 291–296.

[20] D. Lu, Z. Li, D. Huang, X. Lu, Y. Deng, A. Chowdhary, and B. Li, "VC-bots: a vehicular cloud computing testbed with mobile robots," in *Proceedings of the First International Workshop on Internet of Vehicles and Vehicles of Internet*. ACM, 2016, pp. 31–36.

[21] D. Lu, D. Huang, A. Walenstein, and D. Medhi, "A secure microservice framework for IoT," in *Service Oriented Software Engineering (SOSE), 2017*. IEEE, 2017.

[22] M. Gerla, "Vehicular cloud computing," in *Ad Hoc Networking Workshop (Med-Hoc-Net), 2012 The 11th Annual Mediterranean*. IEEE, 2012, pp. 152–155.

[23] Z. Li, "Modeling and control of a longitudinal platoon of ground robotic vehicles," Master's thesis, Arizona State University, 2016.