Enabling following vehicles to mirror pilot vehicle speed

Junyi Ji*†¶, Alex Richardson‡¶, Yuhang Zhang†¶, Zhiyao Zhang†¶
†Civil and Environmental Engineering, ‡Computer Science, ¶Institute for Software Integrated Systems
Vanderbilt University, 1025 16th Avenue South, Nashville, TN.
*Primary contact. Email: junyi.ji@vanderbilt.edu

Abstract

In this report, a prototype for vehicle-to-vehicle kinematic communication between an ego car and pilot car is designed, using an intermediary server connected to via LTE. The ego and lead car software is a standard ROS/libpanda/can-to-ros framework [1] [2]. The test example for vehicle controller is to mirror the speed of the pilot vehicle instantaneously, mainly to test if the control vehicle can receive the signal from the pilot vehicle correctly, which the speed signal is transferred through communication and accepted by ROS network. P-control is designed to mirror the accepted speed. Closed-loop controller design are tested before running the field test. Data is collected from the field test. More closed-loop tests are simulated in Simulink. The prototype test provide insights for future tests and experiments. For the feedback and

I. Introduction

Motivated by the SAILing CAV experiment [3] and the MiddleWay experiment, we investigate building communication between vehicles within the traffic flow to enable stop-and-go waves dissipation using simple V2V communication systems. In the stop-and-go traffic, it is often observed that when the leading vehicle decelerate, the following vehicles will also soon decelerate. That is due to the propagation of congestion wave across the freeway, which is usally in the opposite direction of the traffic flow. Reported by [4], the speed of propagation ranges from -9 to -14 km/h. This indicates that stop-and-go waves could potentially be reduced by getting advanced warning of a stop-and-go wave from a pilot vehicle to the ego car via V2V communication.

Commercial automated cruise control (ACC) vehicle is usually equipped with on-board radar sensor, which make the sensing of local information possible, especially the time-space gap wrt the leading vehicle. This can allow for regulation of traffic flow. However, this doesn't easily allow for advanced warning of a stop-and-go wave. When trying to control the speed with the purpose of mitigating congestion, more non-local information from further ahead in the stop-and-go wave can be more powerful than lead vehicle information alone.

This work seeks to prototype and test the tools to enable ego car control by using a pilot vehicle as the control input. In concept, the pilot vehicle can be many cars ahead of the ego car, and send advanced downstream information and warnings about potential stop-and-go waves. This paper set up a simple scenario of the control car mirroring the speed of the pilot car just to test the communication prototype, and potential speculative investigations into stop-and-go wave detection.

The contributions of this work lie in the following:

- Design a communication framework that communicates kinematic information from the pilot car to the ego car, using the existing ROS/libpanda framework for both vehicles.
- Test the network architecture's ability to communicate and collect real-world data from both pilot car and ego car
- Based on the collected data, open and closed-loop tests are performed to verify the expected behavior of the ego car

II. Related work

Stop-and-go traffic waves are a perennial problem in traffic [4]. Stop-and-go waves emerge from the stochastic perturbations in acceleration from human drivers, which accumulate to form these waves. Furthermore, mainstream cruise controllers are not string stable - they also contribute to the emergence of these waves [5]. Recent research has demonstrated the ability of a small proportion of cars in traffic to regulate stop-and-go waves and dissipate them using strategic regulation of the ego car velocity to in essence absorb the perturbations from the other vehicles [6]. To support the deployment of these potential solutions at scale and gather the appropriate data, there is a ROS/libpanda framework that allows for vehicular telemetry and control of mainstream cars using a raspberry pi [1] [7] [2]. This has recently been deployed in real world highway environments such as [3] and the 2022 November CIRCLES experiment.

These solutions involve the incorporation of non-local information to information that is immediately accessible to the ego car's sensors. These combined allow for advanced ego car velocity management to aid in the dissipation of these stop-and-go waves, while preserving local ego car safety via safety controllers. However, the non-local information that is incorporated involves complex traffic monitoring systems such as INRIX that are high-overhead [8] and may not always be available. This leaves open the possibility of having the ego cars that are deployed coordinating amongst themselves using a low-overhead intermediary, such as a third party web server.

III. Designs and Methods

Our pilot to ego-car communication and control framework involves:

1) Reading kinematic information from the pilot car
2) Sending kinematic information from the pilot car to a third party web server
3) Marking as a database entry the VIN of the pilot car
4) The ego car polls the web server for the pilot car's current kinematic information
5) The ego car sends the pilot car's current velocity as the desired speed to a P controller
6) The output acceleration from the P controller is then mediated by a CBF, as per [3].

A. Car Prerequisite Hardware and Software

The hardware that both the pilot car and the lead car is augmented with is per the ROS/libpanda/can-to-ros framework in [2] [1] [7], and is shown in Figure 1:

1) Raspberry PI 4 with either 4 GB or 8 GB of RAM and 128 GB of storage
2) iPhone to serve as internet hotspot/LTE connection
3) MattHAT and CAN harness to connect to vehicle
4) External power supply for PI

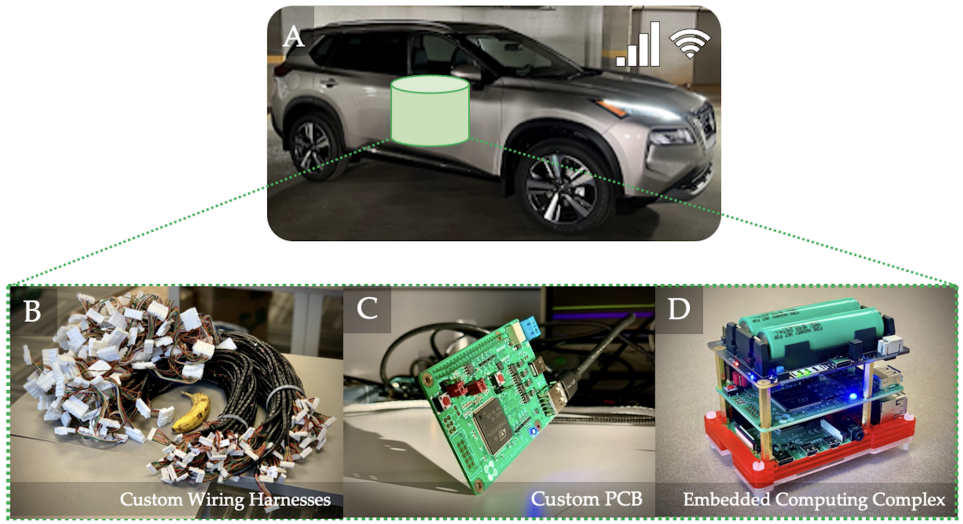


Figure 1: Overall hardware augmentation for both the pilot and the ego car, as per the ROS/libpanda/can-to-ros framework: A is the sample of the control car, B is the wiring harnesses to connect Raspberry PI 4 with ROS system, C is the customized PCB and D is the embedded computing complex to be installed on the vehicle.

B. Overall Communication and Control

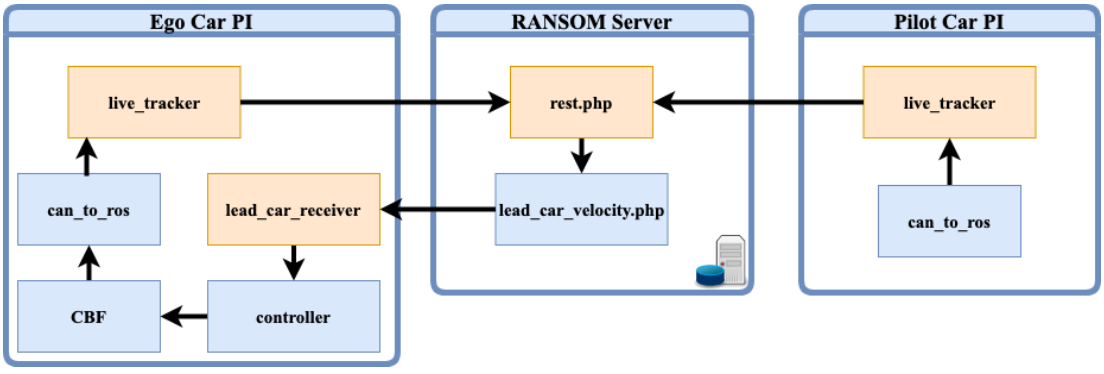


Figure 2: Communication and control structure for the project, involving the ego car, pilot car, and a third party web server, acting as an intermediary.

The overall communication and information flow is described in Figure 2. The pilot car collects kinematic information from the CAN network, and sends them to a third party PHP endpoint. The server stores this information and then passes it to the ego car, which uses it to inform itself of its target velocity.

1) Pilot Car Communication: The pilot car runs a ROS node called **live_tracker** that reads from can-to-ros and sends the following non-exhaustive attributes to **rest.php** at a rate of 1 hz:

- VIN
- Timestamp
- Velocity (kph)
- Latitude
- Longitude

2) RANSOM Server: The RANSOM server has two endpoints, **rest.php**, and **lead_car_velocity.php**. **rest.php** receives the information from the **live_tracker** of both the pilot car and the ego car and stores them in an SQL database as respective rows within a specified table. **lead_car_velocity.php** returns a JSON containing the last reported velocity of the lead car, with the VIN of the lead car specified via a server-side database configuration entry.

3) Ego Car Communication and Control: The ego car runs 4 nodes in conjunction with can-to-ros:

1) **lead_car_receiver** polls RANSOM at 1 hz and publishes the velocity of the lead car as the float ROS topic /pilot_vel.
2) **controller** is the Speed Controller itself. It produces the recommended acceleration to match the pilot velocity.
3) **CBF** is the control barrier function as per [3] and other works - it ensures that no unsafe acceleration ever occurs. **controller** feeds into this, and this node is what actually outputs the control acceleration.
4) **live_tracker** reports to RANSOM just like the pilot car does.

C. Speed Controller

In this work, the design of the controller is grounded in straightforward rule-based logic. This approach is chosen to mimic the speed of a pilot vehicle, predicated on the likelihood that the ego car will encounter similar traffic conditions. It is pertinent to note that the primary goal here is the design of an efficient vehicle-to-vehicle communication strategy. The other aspects of the framework, while necessary, are deemed secondary and are thus kept simple to support this main objective.

As detailed in Figure 3, the controller design incorporates two key inputs: the speed of the pilot vehicle and the speed of the controlled (ego) vehicle, both of which have a frequency of 20 Hz. To facilitate versatility in application, the system includes switches for both offline evaluation and online testing. Additionally, a delay module is integrated into one of the inputs to simulate communication latency between vehicles. The control acceleration is regulated by a saturation function, which ensures that the values remain within safe limits. Lastly, the resulting acceleration command is transmitted to the ROS network for execution at a frequency of 1 Hz. We set the communication delay as 3 seconds, the proportion gain as 0.8 and the saturation function as -3 for the lower limit and 1.5 for the upper limit.

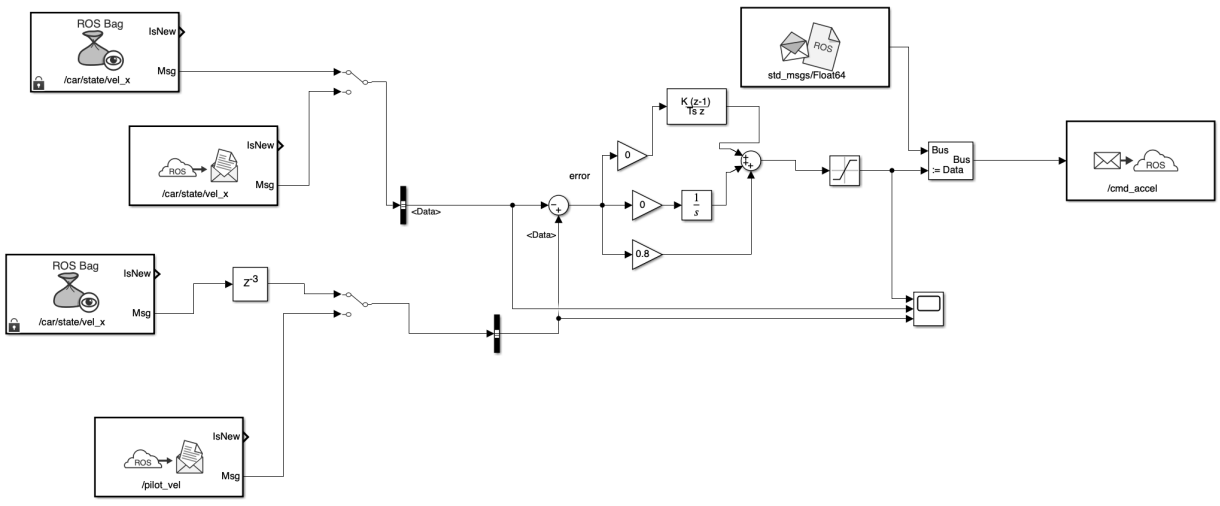


Figure 3: Simulink design diagram of the P-Controller used in this work

IV. Experiments And Results

For all of our real-world oriented experiments, we used a Honda Pilot as our pilot car, and a Toyota RAV4 as our ego car.

A. Experiment 1: Communication Tests

For the communication tests, we tested and confirmed that both the pilot car and the ego car's live_tracker(s) were contacting the server at 1 hz and storing the relevant kinematic information. During these communication tests, we were encountered bugs in can-to-ros when it attempted to rebuild its source for the Honda Pilot, which had to be addressed.

B. Experiment 2: open and closed-loop speed control tests in Docker/ROS

For these we used two bag files from the middleway test that corresponded to two Toyota SUVs following each other.

1) Open-loop Speed Control Test in Docker/ROS: In our code we provide a open_loop_simulation controller and docker container to test it in. This version of the controller outputs command accelerations and does not atempt to hallucinate velocity. Our controller behaves as expected given how we have defined its expected behavior, as seen in 4.

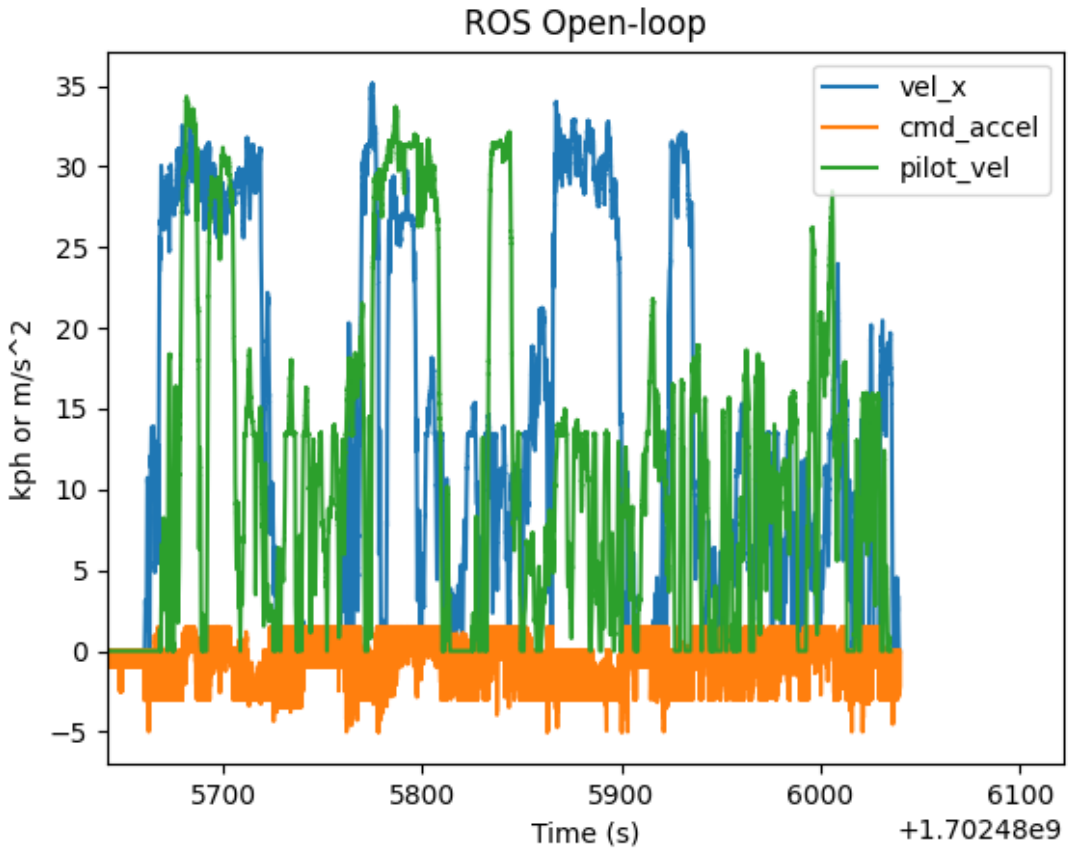


Figure 4: The results of the ROS open-loop test. As one can see, the cmd acceleration for the controller appears to correspond accordingly to ego car's velocity delta with the pilot car, which is a good sign.

2) Closed-loop Speed Control Test In Docker/ROS: This is provided as a closed_loop_simulation controller and docker container in our code. This version of the controller feeds the command accelerations into itself and hallucinates velocity. Again, the controller behaves as expected, shadowing the pilot car's velocity, as seen in Figure 5.

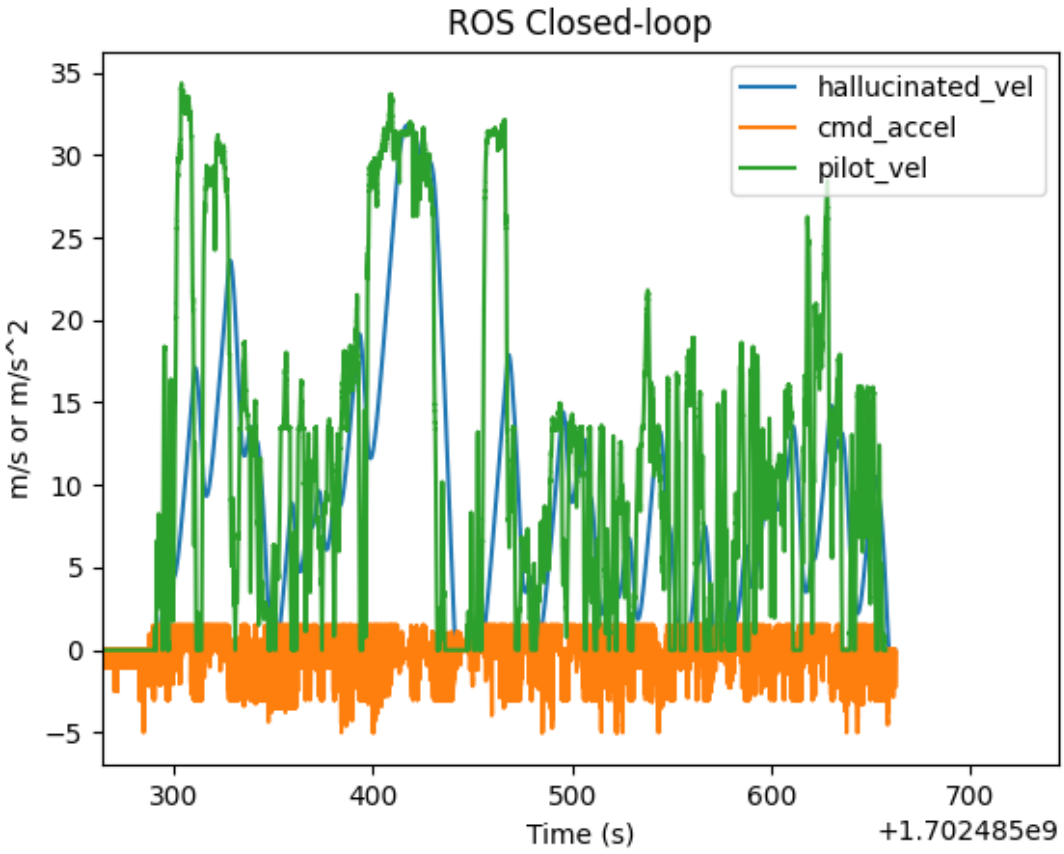


Figure 5: The results of the ROS closed-loop test. As one can see, the hallucinated velocity for the ego car based off of the simplified acceleration to velocity model provided in class closely matches the pilot velocity, indicating that the controller nominally performs what is intended.

C. Experiment 3: speed control field test

The controller was tested in the urban environment to mainly check the communication between the pilot vehicle and the control vehicle and potentially see how the pilot car's data affects the ego car's data. Unfortunately, as an oversight, the intermediary web server was misconfigured, and did not store the pilot car's VIN, but instead a bogus entry. This meant that all pilot velocity outputs from the web server were constant and meaningless, as it used a bogus row in the database. You can see the communication results in Figure 6.

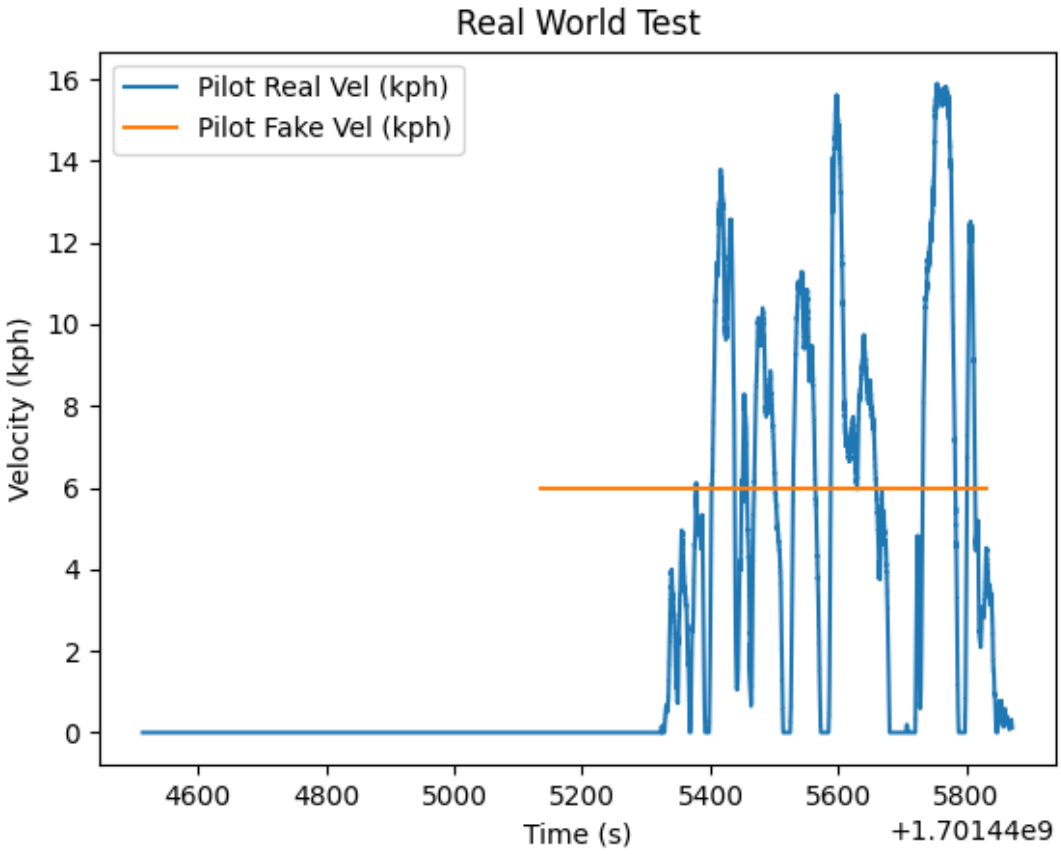


Figure 6: The results of the real-world driving test. As you can see, the real velocity of the pilot vehicle, versus what is indicated on the ego car, do not match up, with the ego car's idea of the pilot velocity remaining constant.

1) Potential Lesson Learned: When one is doing preliminary testing, if one is using a web server or other such external resource to mediate between different autonomous vehicles, it needs to be incorporated into the offline testing process. This will allow fewer mistakes to make it to real world tests, which makes the iteration process safer and quicker.

V. Is adopting this kind of controller a good idea?

In particular, for this we will need to be addressing the speed controller itself, rather than the V2V framework that we are doing a proof of concept of. For the sake of conceptual discussion, lets assume that "adopting" this means every car "mirrors" the velocity of a car n cars ahead on a single-lane, straight, non-circular roadway with roughly homogeneous spatial distribution of vehicles.

The obvious side-effect of every car adopting this policy is that changes in velocity at the head of the traffic flow will rapidly propagate through to the tail of the flow. Each car is mirroring a car downstream of it, which means that as each car mirrors the cars ahead, the cars that are mirroring **it** also begin to mirror those cars by proxy, albeit with a delay due to the P controller. Without stochastic or external perturbations, we can see that the equilibrium for the system is for all the vehicles to be at the same velocity, which would preclude any stop-and-go waves if it was achieved. Wrt our goals of dissipating and preventing stop-and-go waves, this would appear to indicate that mass adoption would serve this goal. However, this does not establish the safety of such a system in the real world, or whether the system can converge onto this equilibrium. Research in Flow/Sumo in the future could provide the appropriate indications about the stability and convergence properties of such a distributed system.

VI. Conclusion and future work

We've developed a functional V2V framework between two cars, a pilot car, and an ego car which follows it. The ego car can use its local information and the kinematic information from the pilot car in order to augment its stop-and-go wave dissipation controllers as seen fit. Furthermore, we developed a basic controller that could help ameliorate such waves by

directly mirroring the pilot vehicle's velocity. We then tested this controller and the framework, both offline and online. While the offline open and closed loop simulations showed that our controller had consistent behavior, and in concept this controller could improve stop-and-go wave dissipation, our only real-world test failed due to the web server being misconfigured. Thus, the next technical steps for the speed controller are large simulations in Flow/Sumo to study the behavior of such a controller, and more real-world tests once the properties of this controller are better understood. For this V2V framework, designing more sophisticated offline testing strategies that include testing the web server and the PIs on each car concurrently could potentially prevent such real world test failures.

ACKNOWLEDGMENT

Special thanks to Matt Nice for discussing the scope and feasibility concepts of this class project. Thanks Matt Butting for helping us tuning the can_t_ros on the Honda Pilot. Appreciation to Dan Work and Jonathan Sprinkle for their wonderful teaching and invaluable feedback throughout the whole class.

APPENDIX A
RESPONSE TO THE FEEDBACK

Feedback/questions:

1) for purposes of the class, you are (rightly) pre-coding which car is lead and which is follow. How do you think you would check this (live) if you had an on-server or on-cloud application?
   Response: In future tests, we plan to introduce a module designed to perform calculations on the server, particularly for applications on I-24. This will involve several key steps: firstly, determining whether vehicles are within the I-24 testbed region. Secondly, we will implement a layer to distinguish whether the vehicles are traveling eastbound or westbound. Lastly, we will add a layer that aligns each vehicle's location with the nearest mile marker.
2) is the mirroring done instantaneously, or in space-time with -12 mph?
   Response: For now, it is instantaneously, but for future, we hope to set up an algorithm to detail the control logic. We plan to further scope the operation domain of the controller. There are couple of interesting directions to explore. One of them is to combine the head-to-tail stability and local stability together, to see if we can use one automated vehicle to compensate for the unstable behavior from the human drivers.
3) I really like that you are simulating the comms latency with the system design in testing.
   Response: Thanks for pointing this out. We hope to test the empirical latency from the field test so that it can be the design parameters in the next iteration.

TODO:

1) for your final report, it would be great to show some candidate signals in a time-space diagram (even if drawn as a simulated signal) that shows when the values of the speed should be used, and when you are so far away that it does not matter and the follower would do something different.
   Response: More close-loop experiments are tested (see Figure 6 for details). We plan to cooperated with TransModeler/SUMO for the test and validation to gain insights.
2) suggest using PID block for future iterations, as its codegen is much more reliable what's the test status that you would use to find that this error in entering the VIN would have been caught?
   Response: Thanks for the suggestions, we make the following checklist before the next iteration:
   a) New branches updated for the Pi.
   b) Make sure the can_to_ros compiled
   c) Setup: vin to say which one is the pilot and which one is the control
   d) **Set up a check code to see if the speed is well suggested and calculated**

APPENDIX B
ADVANCED WAVE DETECTION

This section falls outside the scope of experimentation, testing, and validation. While the code will be made available, it has not yet been incorporated into the primary framework. Code and supplementary materials are available here: https://vanderbilt.box.com/s/gavdakjs5xshe3fvtt88xdpwcvfmy09w.

A. Introduction

In this section, an exploratory module for traffic wave detection is designed, under the scope that a pair of vehicles are connected where the lead vehicle is considered a probe sensor to collect and spread real-time traffic conditions as it travels, and the following vehicle adjusts its cruising speed according to the sensing information from the lead vehicle in order to mitigate the propagation of stop-and-go waves by slowing down in advance. The technical difficulty is that compared to sensing with a group of probe vehicles with sufficient penetration in the traffic that can reliably detect traffic waves, the sensing information from a single vehicle may be too noisy to be reliable. Thus, it is looked forward to an advanced wave detection module that is more robust to noises and be able to detect waves depending on multiple types of sensing information gathered. It is expected that this module outperforms naive detection strategies such as a rule-based classifier simply based on the vehicle speed. The advanced wave detection module is not implemented in the actual controller that has been tested. Thus, the following content in this section will elaborate the modular design and conceptually verify it with the test dataset.

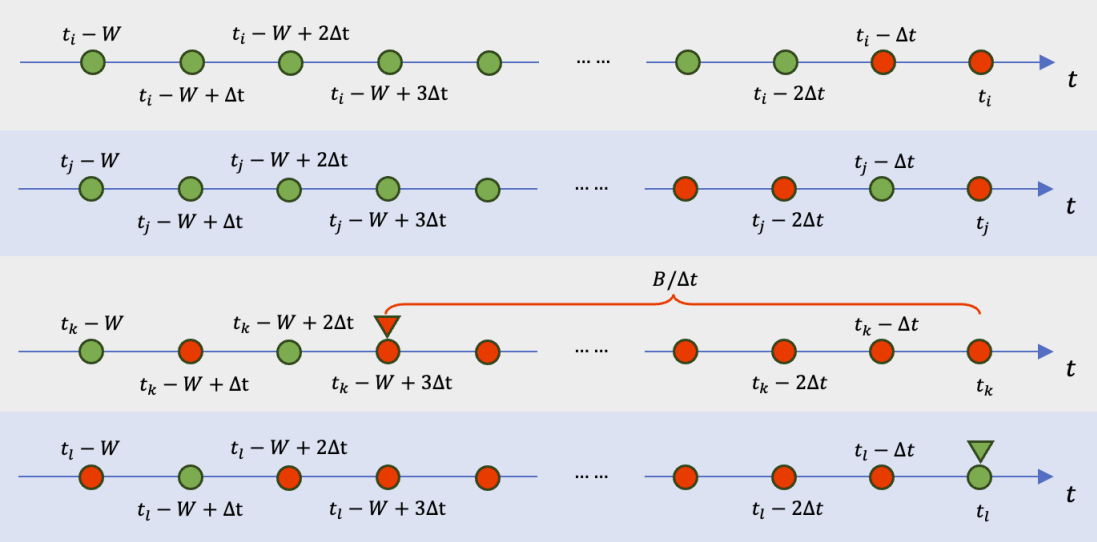


Figure 7: Example of retrospective wave detection over four timesteps. Each node on the time axis is an observation datapoint to be classified (red: congestion, green: freeflow). The symbol ∇ represents the datapoints whose time and corresponding travelled distance are reported as either entry or exit point of the traffic wave. At timestep t_i, two datapoints are labeled congestion which is shorter than the bandwidth window. At timestep t_j, the bandwidth window is reset as an exceptional datapoint at $t_j - \Delta t$ is labeled freeflow. At t_k, a sequence of consecutive datapoints in the moving bandwidth window meets the detection condition, so the first datapoint is reported as the entry point to a traffic wave. t_l is the first datapoint labeled freeflow which is reported the exit point from the wave.

B. Problem Settings

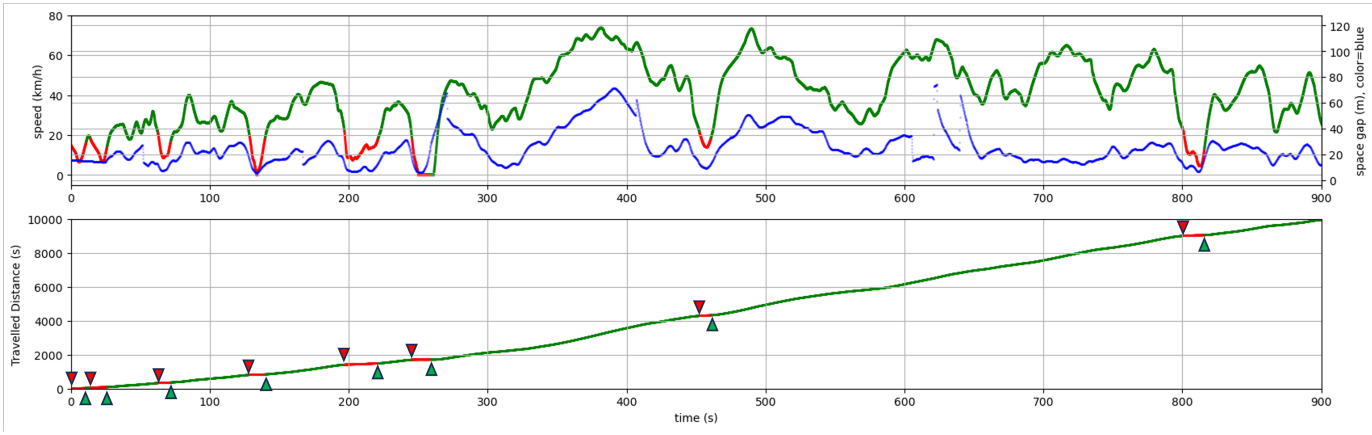
Consider the lead vehicle travelling across a discretized timespan $T_n = \{0, \Delta t, 2\Delta t, ..., t_i, ..., t_n\}$ where Δt is update time interval (typically $1/40$ second) and at each time of update t_i it reports the travelled distance $(t_i, x(t_i)))$. The detection module is designed to report the time and distance at which the probe vehicle enters and exits a traffic wave $(t_i^{in}, x_i^{in}), (t_i^{out}, x_i^{out})$. It is not possible to immediately detect the traffic wave since there are various reasons a vehicle slows down subject to future observation to distinguish whether the slow-down event is cause by propagating traffic waves. In comparison, the wave detection problem is formulated as *retrospective*, such that a time series of previous observations, $\{S(t_i - W), S(t_i - W + \Delta t), S(t_i - W + 2\Delta t), ..., S(t_i)\}$, within a pre-defined detection time window W in seconds are classified, where $S(\cdot)$ is the sensing information of an update including the speed, acceleration, and space gap of the lead vehicle. An example is given in Fig 7. Every datapoint in the time series is labeled either freeflow or congestion. The designed algorithm defines a moving bandwidth window $B < W$ also in seconds and introduce the detection condition such that the probe vehicle is considered

in the traffic wave during the period of time where all datapoints are labeled congestion. Once this condition is satisfied, the module returns the time and traveled distance that correspond to the very first datapoint in the bandwidth window, in this example, $(t_k - W + 2\Delta t, x(t_k - W + 2\Delta t))$. Once a datapoint after the bandwidth window is classified as freeflow, the module immediately report the exit time and travelled distance $(t_l, x(t_l))$ and reset the bandwidth window for the next detection. Specifically in this design, W is 30 seconds and B is 5 seconds.

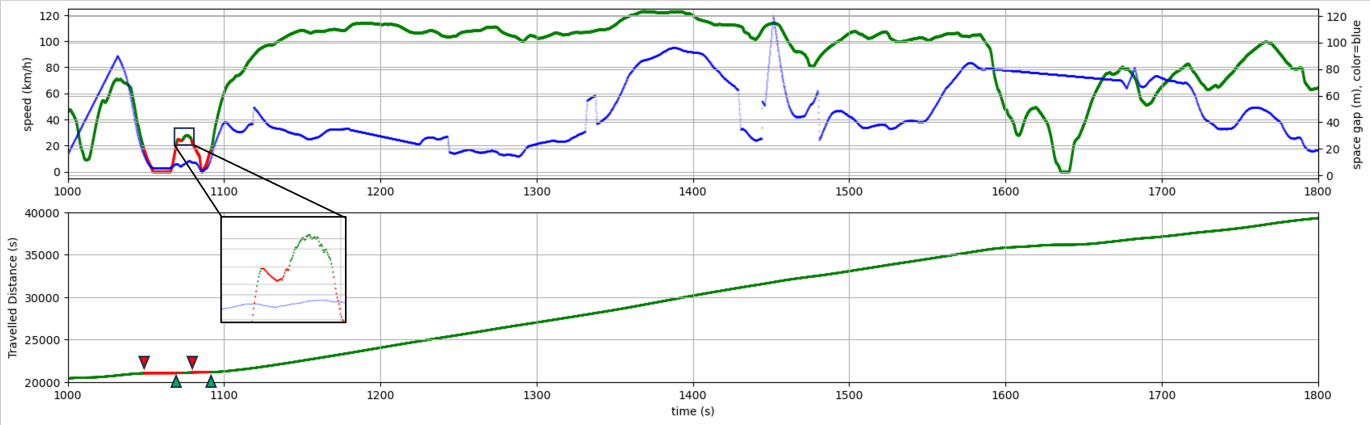
C. Data-driven Classification

Labeling each datapoint as either congestion or freeflow is a binary classification task. A Machine Learning based classifier called Support Vector Machine (SVM) is used for this task. SVM aims to find the optimal hyperplane that best separate different classes in the feature space. This hyperplane acts as a decision boundary that segregates classes of data. In our task, data points on either side of the hyperplane are attributed to congestion or freeflow. The designed module takes the advantages of SVM including (1) it efficiently calculate the classification problem even in higher dimensional input space; (2) it is capable of classifying data whose features are only nonlinearly separable (i.e., data features are nonlinearly correlated); (3) it does not require a large size of labeled data for training and is robust to overfitting.

D. Experiment Results



(a) First example with 8 traffic waves reported in a 15-minute timespan.



(b) Second example with 2 traffic waves reported in a 13.5-minute timespan. Between the two reported waves there is a unreported short sequence of congestion datapoints which is shorter than the bandwidth window of 5s. There are also two slow-down event, one from 0s and the other after 1600s with missing space gap information (interpolated in the figure). The classifier omits reporting the wave in this case since the sensing information is not reliable.

Figure 8: Testing results of the trained wave detection module. The upper plot in each subfigure shows the relationship between resulting labels and inputs (speed, acceleration, space gap). The lower ones are reported entry and exit points in time-space diagrams, respectively marked as red inverted triangles and green triangles.

The algorithm uses a human-driving dataset that has been collected on I-24 highway at the southeast side of Nashville, TN [9]. 4 among 39 trajectories in the dataset are labeled. However, since individual vehicle data does not directly reflect the traffic status, in consideration are only those datapoints *most likely* being in the traffic wave with the principle that the vehicle

meet a rapid slow-down (acceleration below -1 m/s^2), stay at low-speed (<10 km/h) for at least 5 seconds with a narrow space gap (less than 20 meters), and speed up to higher than 40 km/h afterwards. The entry and exit points are set at the speed of 20 km/h. A commonly nonlinear kernel function known as Radial Basis Function (RBF) is chosen for SVM.

Two representative classification results from the other 36 unlabeled trajectories in the dateset are shown in Fig 8. A typically expected outcome from fig 8a reports 8 traffic waves within a 15-minute period. The observations on reported datapoints appear reasonable as all of them exhibit significant stop-and-go patterns with low speed, hard deceleration, and narrow space gaps. More information can be observed from the outcome in Fig 8b with two unreported slow-down events and one ignored congestion classification. Both unreported slow-down events do not have space gap information (it is linearly interpolated in the visualization). It is desired that if the sensing information is missing or unreliable, the detection module acts conservatively and does not report. Also, there is a short period of time series (shorter than the bandwidth window B) classified as congestion ignored. It appears not to be a typical stop-and-go wave but more likely to be speed fluctuations in between two temporally adjacent waves.

E. Limitations

While this section conceptually verify the correctness of the data-driven wave detection module, it's necessary to claim this prototype has nontrivial limitations that are addressable with future developments. First, the dataset for classifier training is solely from individual human-driving vehicles which lacks the ground truth of traffic conditions. Some of labeled data might not in fact be in the stop-and-go wave, thus bringing biases in the training outcome. Besides, it's yet infeasible to test this module in the series of experiments. However, it is expected to fine-tune the classifier based on empirical test results on the highway. It is expected that, within the scope of traffic wave mitigation with a pair of connected vehicles, the proposed advanced wave detection module would be a competitive candidate to process the sensing information from the lead probe vehicle, and therefore provide reliable decision-making supports for problem.

REFERENCES

[1] M. Bunting, R. Bhadani, and J. Sprinkle, "Libpanda: A high performance library for vehicle data collection," in *Proceedings of the Workshop on Data-Driven and Intelligent Cyber-Physical Systems*, 2021, pp. 32–40.

[2] S. Elmadani, M. Nice, M. Bunting, J. Sprinkle, and R. Bhadani, "From can to ros: A monitoring and data recording bridge," in *Proceedings of the Workshop on Data-Driven and Intelligent Cyber-Physical Systems*, 2021, pp. 17–21.

[3] M. Nice, M. Bunting, G. Gunter, W. Barbour, J. Sprinkle, and D. Work, "Sailing cavs: Speed-adaptive infrastructure-linked connected and automated vehicles," *arXiv preprint arXiv:2310.06931*, 2023.

[4] M. Treiber, A. Hennecke, and D. Helbing, "Congested traffic states in empirical observations and microscopic simulations," *Physical review E*, vol. 62, no. 2, p. 1805, 2000.

[5] 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*, vol. 22, no. 11, pp. 6992–7003, 2020.

[6] R. E. Stern, S. Cui, M. L. Delle Monache, R. Bhadani, M. Bunting, M. Churchill, N. Hamilton, H. Pohlmann, F. Wu, B. Piccoli *et al.*, "Dissipation of stop-and-go waves via control of autonomous vehicles: Field experiments," *Transportation Research Part C: Emerging Technologies*, vol. 89, pp. 205–221, 2018.

[7] M. Nice, S. Elmadani, R. Bhadani, M. Bunting, J. Sprinkle, and D. Work, "Can coach: vehicular control through human cyber-physical systems," in *Proceedings of the ACM/IEEE 12th International Conference on Cyber-Physical Systems*, 2021, pp. 132–142.

[8] G. Cookson and B. Pishue, "Inrix global traffic scorecard–appendices," *INRIX research*, 2017.

[9] M. Nice, N. Lichtle, G. Gumm, M. Roman, E. Vinitsky, S. Elmadani, M. Bunting *et al.*, "The i-24 trajectory dataset," 2021.