Data from the Development Evolution of a Vehicle for Custom Control

Matt Bunting
Institute of Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
matthew.r.bunting@vanderbilt.edu

Rahul Bhadani
Electrical & Computer Engineering
The University of Arizona
Tucson, Arizona, USA
rahulbhadani@email.arizona.edu

Matt Nice
Civil & Environmental Engineering
Vanderbilt University
Nashville, Tennessee, USA
matthew.nice@vanderbilt.edu

Safwan Elmadani
Electrical & Computer Engineering
The University of Arizona
Tucson, Arizona, USA
safwanelmadani@email.arizona.edu

Jonathan Sprinkle
Computer Science
Vanderbilt University
Nashville, Tennessee, USA
jonathan.sprinkle@vanderbilt.edu

***Abstract*—In order to develop custom controllers intended to operate vehicles on a live highway, a series of data collection-focused tests were performed at increasing stages of complexity. Modern vehicles with features like Adaptive Cruise Control (ACC) feature a rich set of sensors and drive-by-wire mechanisms. The presented stages of data collection begins with the analysis of raw data provided by various vehicles, and eventually results in spoofing Controller Area Network (CAN) protocols for sending control commands to operate a vehicle. This paper covers the data and technical efforts needed at various stages. The raw data and tools to plot the data are also publicly available.**

***Index Terms*—datasets, autonomous vehicles, traffic, control systems**

I. Introduction

The data described in this document covers the development evolution of an experimental vehicle platform for control experiments. Many modern vehicles are equipped with sensors that can provide rich data (such as radar data) when tapping into the vehicle's infrastructure [1]. Such vehicles use these sensors for sophisticated behaviors like Adaptive Cruise Control (ACC), and further tapping into these vehicle's network of sensors and actuators can enable custom vehicle control [2]. The data here presents the journey of from safely gathering data, to decoding data, to characterizing the system's response, to creating custom cruise controllers using a Toyota Rav4 and Raspberry Pi.

Modern vehicles mainly use a Controller Area Network (CAN) bus to let electronic sub modules communicate with one another [3] [4]. There has been a tendency for manufactures to have all electronic modules in a car communicate over this bus, including engine diagnostics, cruise control units, and headlamps [5]. Because the CAN protocol is an open standard, off-the shelf devices are able to read and send information over the CAN bus [6].

Though the CAN protocol is an open standard, the data shared between modules is typically proprietary, due to reasons such as trade secrets or concern for safety. Manufacturers will typically not disclose information on the design of their controllers, sensor interpretations, nor the handshaking needed to operate submodule states. Reverse engineering efforts are then needed to build custom modules that can interact with various Original Equipment Manufactured (OEM) modules. Some companies such as Comma.ai have entered this space by selling hardware that can connect to a vehicle's infrastructure and inject their own control commands.

The data gathered here presents the journey from decoding data to building custom cruise controllers. The vehicle of choice is a Toyota Rav4 due to the existing Comma.ai hardware. The solutions from comma.ai however do not fully meet the needs of control and sensing for this project. Though comma.ai is capable of getting data and controlling vehicles, the open-source software does not provide the performance of data collection nor interfaces for advanced control that could track velocity trajectories for applications such as traffic wave dampening [7], [8]. Therefore building on comma.ai existing hardware was needed to ensure data integrity so that data could correctly drive development of custom control.

The data presented here is a curated set from six experiments that were instrumental in the development of a controllable vehicle. There were other experiments performed however the results are similar from those presented. Each test represents achieving an important milestone in development cycle.

II. Hardware Setup

Retrofitting a vehicle for data collection and control is possible through the use of some off-the-shelf hardware. One necessity that manufacturers have integrated into vehicles since 1996 is the Onboard Diagnostics (OBD) port [9]. The OBD and newer OBD-II port have been needed by vehicle emission regulation standards. Typically, the OBD port is intended for a limited set of readable information through the Parameter ID (OBD-PID) protocol [6]. Separate CAN busses will exist in a

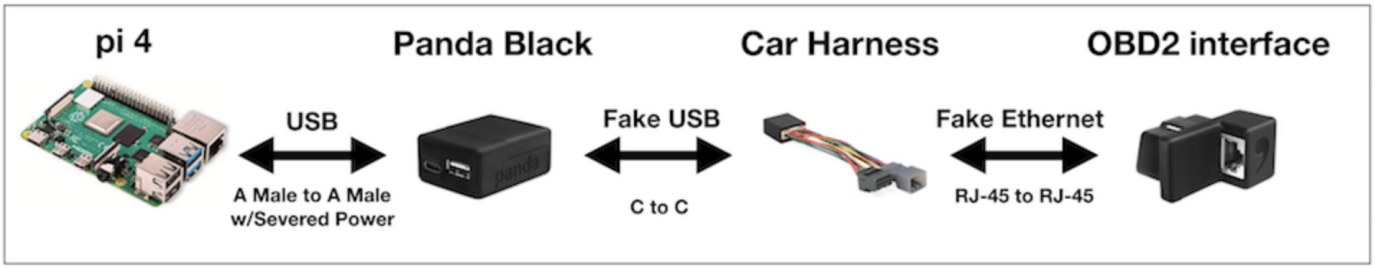

Fig. 1. Connections of the Comma.ai Panda.

vehicle to handle subsets of data communication. The OBD-II has a specialized CAN bus that does not share low-level messages from all devices. Most off-the-shelf CAN devices for vehicles are in the form of OBD-II readers. The OBD-II does not meet our use case of the need to read live information of specific sensors, in a timely manner, while also allowing custom control.

The limitations of pure OBD-II readers have opened a space for some companies to build devices that can interact with the low-level CAN busses. Comma.ai provides such hardware and has provided software and firmware as open-source. One module provided by Comma.ai, the Panda, is able to tap into the CAN bus at two points to gather information and inject CAN messages. The Panda can then translate messages to a computer through USB.

Each Panda has three CAN peripherals for bus connections. One of these is connected to the OBD-II port to gather high-level diagnostic information such as the Vehicle Identification Number (VIN). The other two CAN peripherals are attached between an ACC specific module in the vehicle and rest of the CAN network. This connection is what allows the Panda to block messages form being sent in order to send custom messages. Similarly, the ACC module needs information about the vehicle's following distance, so this point of connection provides important low-level information for the design of a custom ACC.

Scalability is an important factor, driven by research goals to have coordinated driving of perhaps hundreds of cars to carry out energy-savings control control. Even though the panda is an off-the-shelf unit that meets our needs, it still requires a computer to be connected with sufficient storage space for collection and computational power to run custom controllers. Fortunately a Raspberry Pi 4 is an inexpensive and ample solution for our needs. Figure 1 shows how all the pieces fit within a vehicle.

Though Comma.ai provides software that can run on a Raspberry Pi, we quickly discovered that there were missed packets of data from the CAN bus. The order of missed packets was around 50%. The provided code was written in Python which has a large overhead. This resulted in the development of a high-speed C++ library coined Libpanda, with a targeted focus to run on a Raspberry Pi [10]. In doing so we were able to capture all CAN data while reducing the CPU usage from 100% to 30%. Libpanda also reads the GPS module built into the Panda and logs all GPS data. Other features were also added to libpanda, such as the ability to synchronize the Pi's system clock with GPS time, since the Raspberry Pi does not have a battery for time keeping and will typically be away from a network to sync time over a Network Time Protocol (NTP) server.

To gather data in a streamlined manner, a set of scripts have been included along with the libpanda package to automatically upload data gathered from experiments. Figure 2 showcases the lifecycle of a data collection experiment, from the Raspberry Pi boot from boot to the car shutting down.

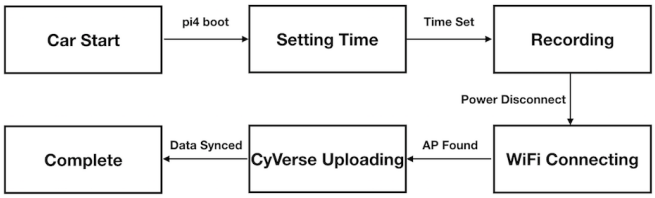

Fig. 2. The lifecycle of a data collection experiment

This data is uploaded to CyVerse, a hub catered towards data science for data storage and data processing [11]. CyVerse can host data and tools and allow collaborators to engage with the data. Similarly, data and tools can also be published so that anyone can use the data in their research. Tools also aid in reproducing results from gathered data. All data presented here is publicly available on CyVerse.

The Python-based package Strym has also been developed in conjunction with Libpanda for the analysis of raw CAN and GPS data [12]. The plots shown in this paper use Strym. Plots are constructed directly from raw CAN data using a CAN Database (DBC) file, which has a set of descriptors for decoding data. Strym can run in Jupyter notebook, and CyVerse has the ability to host a notebook instance so data may be analyzed directly from the servers without the need to download data files.

III. Test 1: Radar Data Validation

The first set of tests we conducted was in Nashville. We treated the Panda in a read-only state, so that the vehicle could operate normally. A Panda was still used in this test as opposed to a normal OBD-II reader since the main focus was to gather data pertaining to the front radar sensor. Though the Panda

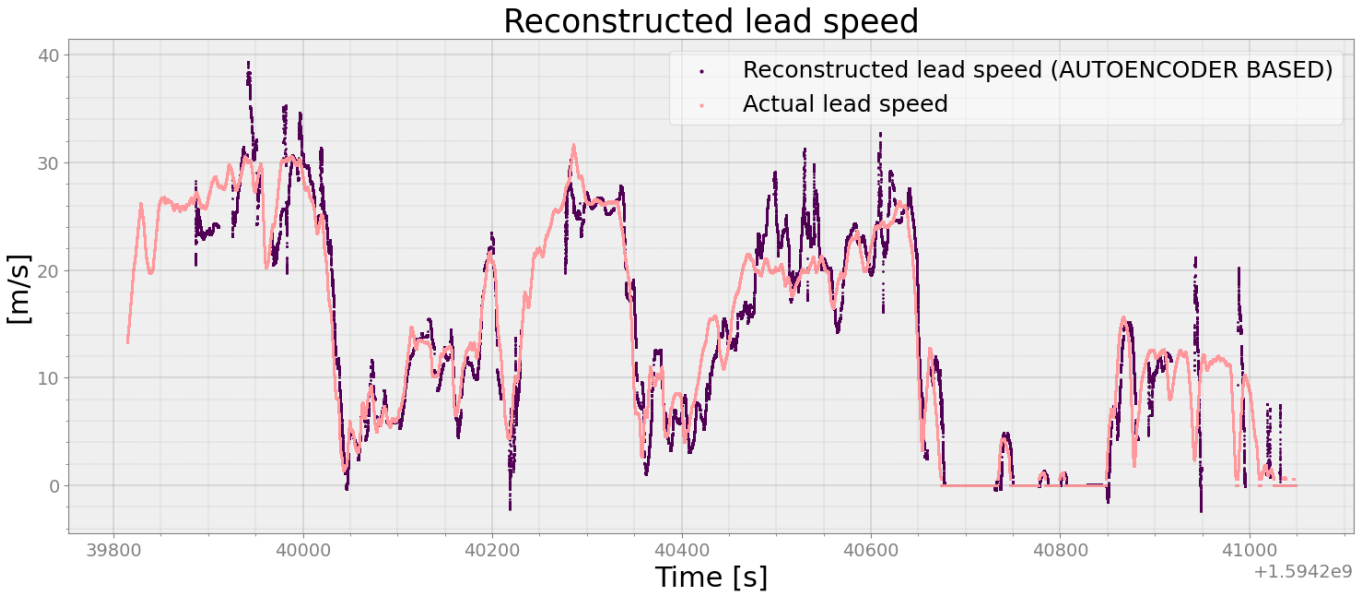

Fig. 3. Reconstructed lead speed based on CAN bus messages of follower vehicle from Test 1

can intercept signals form the CAN bus, an internal safety configuration of the Panda only allows signals to be sniffed. This test was performed on live roads around Nashville, but to ensure that data could come from a leader vehicle under partially controlled conditions, a known driver drove a Honda Pilot. This would reduce the number of unexpected events in the test such as cut-ins or exceeding the speed limit. similarly, the Honda was also equipped with data-collection hardware to check the process of aligning data. A dashcam was also equipped to provide visual data validation in the instance of unexpected events causing potentially anomalous looking data.

- To determine if the follower car (Rav4) can measure the leader car (Honda Pilot)'s velocity reliably.
- To develop a dataset for later work on decoding radar messages from leader car (Honda Pilot).
- To create alignment algorithm for aligning CAN bus timeseries data obtained from multiple sources – namely the leader and the follower car.
- Types of Vehicles: Honda Pilot (leader) and Toyota Rav4 (follower)
- Dashcam footage used to verify vehicle data
- Location: Nashville, TN, clockwise around Downtown
- Date: July 8, 2020

A. Results

This data served to decode radar messages from the raw CAN data for what the car reports as a "leader distance" and "radar tracks". Figure 4 Shows dashcam footage overlaid with post-processed radar data, validating successful decoding of radar data. This demonstrated the point cloud data provided by the raw radar sensor lines up with the leader vehicle and other object on the road, including vehicles in other lanes and passing structures like barriers. .

Fig. 4. Dashcam footage with post-processed radar decoding

IV. Test 2: Radar Data with Overhead Cameras

To provide further validation of radar data and time synchronization, while also validating architectures of another parallel project, a similar experiment was repeated under a particular section of I-24 in Nashville called the Testbed. The Testbed encompasses multiple overhead cameras that capture the highway in 4K resolution, useful for measuring all vehicles on the road, not just vehicles equipped with Panda hardware and ACC. To expand upon the dataset, an additional vehicle was included in the front of the platoon.

In addition to measuring leader distance, an important state that can be used in control laws and safety mechanism is knowing the leader's relative velocity. This test also served to measure the leader relative velocity from direct CAN data, along with inferring relative velocity of non-instrumented vehicles on the Testbed.

- Create a 3-vehicle platoon

- Drive under the I-24 Instrumented Testbed (18 cameras from 3 poles currently operational at that time)
- Types of Vehicles: Honda Pilot (Follower) and Toyota Rav4 Hybrid (Leader) and Honda Fit (unrecorded Super Leader)
- Location: I-24, South-southwest of Nashville
- Date: October 13, 2020

A. Results

Fig. 5. Overhead footage with dashcam

Similar to the first test of gathering and aligning data, figure 6 shows the similar result of aligning speed data in this experiment. This data not only aids in the development of the testbed, but further validates our methods of aligning data from multiple vehicles.

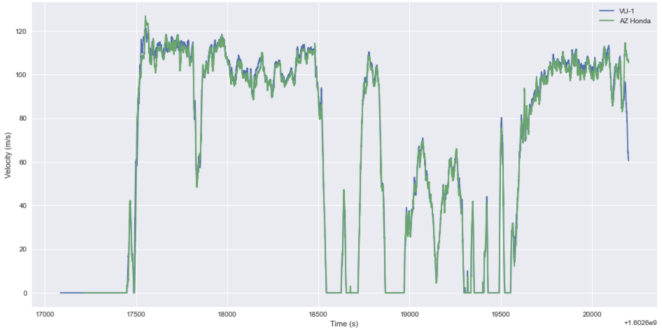

Fig. 6. Speed data aligned from the Toyota and Honda.

V. Test 3: Spoofing Cruise Control Commands

Gearing up for sending control commands requires understanding the necessary protocols of various submodules. These submodules often have built-in runtime verification, so that errors detected either by value or in the handshaking process will cause an error that resumes all vehicle control to the driver, along with an error message. In the case of the Toyota Rav4, causing such an error will result in an audible chime along with a "Cruise Control Malfunction" message. When this error state of the vehicle's OEM modules are reached, all further commands to re-engage control of the vehicle are ignored until the vehicle is restarted. In order to ensure proper state handling, this test was performed to validate the operation of submodules. This protocol has no official documentation, so ensuring proper submodule handling is the result of reverse-engineering the runtime verification built into the Toyota Rav4.

- Type of Vehicle: Toyota Rav4
- Location: Tucson, AZ
- Date: February 03, 2021

Test 3 aims to perform a set of actions that should safely disengage the cruise controller without causing an ACC error. This happens during normal vehicle operation regularly, for example when the brake is pressed by the driver when the cruise controller is active and engaged. In the OEM ACC unit, the pedals are read through CAN data. If a pedal is pressed, the acceleration request values need to be set to 0. If the values are non-zero, then a cruise control fault occurs. This test explored a set of sequences that were known to potentially cause an ACC fault if these conditions were not met by a spoofed ACC module running as C++ code on the Raspberry Pi.

- Control the car using the ROS-based commands from the Raspberry Pi.
- Check ACC interface state machine for safe disengagement

VI. Test 4: Measuring Energy Usage

Part of the overarching goals to create a custom ACC for vehicles is to reduce the energy consumption with the controllers. Finding the live fuel consumption rate is therefore the focus of this test. In tandem, a specialty tool for Toyota vehicles called Techstream Connects to the OBD-II port, and can interact with many of the vehicle's electrical components. This can be used for mechanics to operate various actuators like the windows or doorlocks for vehicle service and repair. Similarly, Techstream can report the fuel rate, much like other off-the-shelf OBD-II scanning tools. Using an OBD-II splitter, data can be recorded by Techstream concurrently with libpanda. Due to technical limitations of the Panda device, the OBD-II port cannot read fuel rate simultaneously while having controls enabled, therefore fuel consumption needs ot be inferred by other CAN messages.

To measure the effects of energy based on driving inputs, a long, relatively flat section of road was found near Marana, Arizona. Test 4 performed various drives over this road while recording data using the splitter.

- Gather concurrent information from both libpanda and techstream, on a fairly straight road with almost no grade. Various driving maneuvers to be performed.
- If successful, we would be able to compare data from Toyota Techstream to data from libpanda under the same scenario, leading to validation of fuel models, and gradient information.
- .
- RahulValidating fuel usage prediction model. We developed a mathematical model for predicting fuel usage based on speed and acceleration. As this model is an

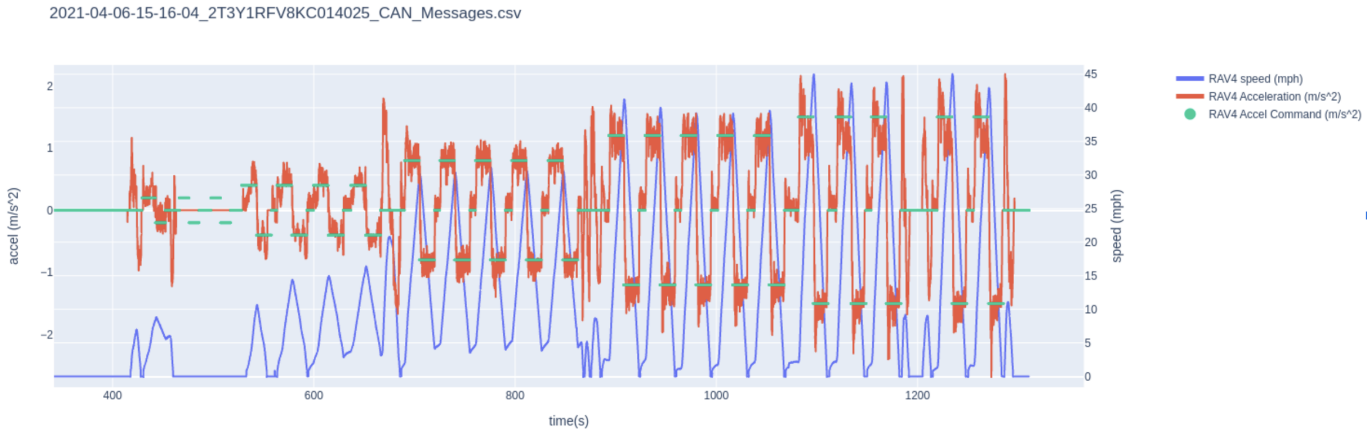

Fig. 7. Command response due to various step inputs of acceleration and deceleration. The commanded acceleration is shown with the vehicle's reported acceleration form the CAN bus, along with the vehicle's speed.

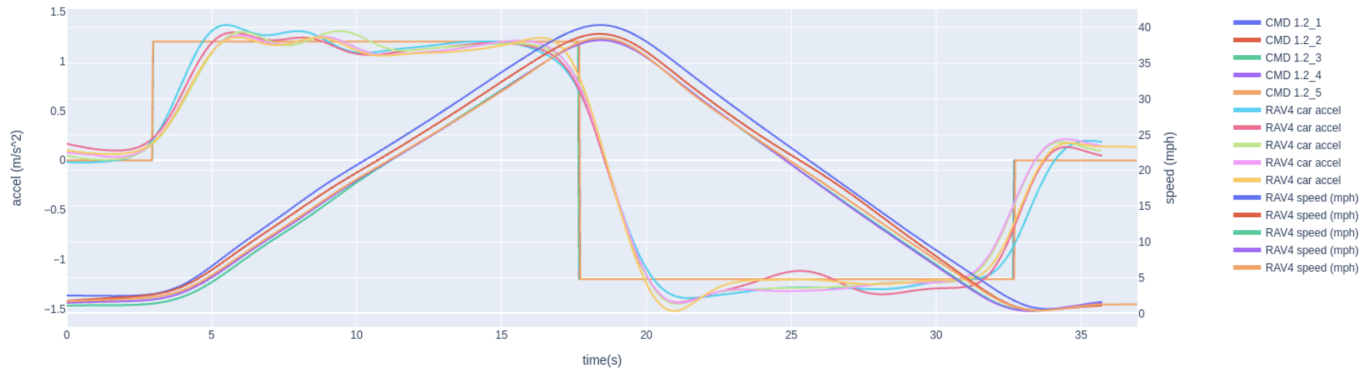

Fig. 8. From the same data as figure 7, however with the same $1.2\frac{m}{s^2}$ magnitude step input response overlaid on each other to check for system consistency.

approximation and doesn't take into account road grade, the assessment was necessary for model's validity.

- Type of Vehicle: Toyota Rav4
- Location: Marana, AZ
- Date: 11 February 2021

A. Results

To further validate data, the vehicle speed reported by Techstream is also recorded to be compared against the speed CAN messages recorded by libpanda. The combination of a ground-truth set of fuel consumption data can be used to compare the proxy data of other known messages provided by the CAN and GPS data.

VII. Test 5: System Characterization Through Step Input Responses

In order to build a controller for a system, a model of the system must be known. A common practice in controller design for a blackbox scenario is to apply a known input and observe the system's response. In the case of the Toyota Rav4, the stock ACC unit sends a requested acceleration. This is one message that the Panda can actively block and reproduce. Important to note is that this is a request, the true acceleration that the vehicle will perform is based on many factors such as engine torque at different RPM, current speed, transmission state, payload, road incline, and many other hidden states of the vehicle. Typical simple approximation is to treat the system as a 2nd-order system, which can be easily characterized and validated from the response of a step input.

- Type of Vehicle: Toyota Rav4
- Location: Marana, AZ
- Date: April 06, 2021

Test 5 applies a series of step inputs at varying initial conditions of speed, along the same stretch of fairly flat road in Marana, Arizona. Step inputs were also applied at different magnitudes and at different initial conditions. According to documentation of the Toyata Rav4 from Comma.ai, the maximum supported acceleration request was $1.5\frac{m}{s^2}$ and deceleration was $-3.0\frac{m}{s^2}$. We decided to test the maximum allowable acceleration, but the reaching the maximum deceleration raised potential safety concerns in final controller implementations so characterizing the system under extreme deceleration was not considered. The set of step responses applied were as follows:

- Starts from 0-2 mph at 0.2, 0.4, ..., 1.2, 1.4, 1.5 m/s^2 for 15 seconds

Fig. 9. Checking the vehicle's dynamics with a custom PID speed controller

- Starts from 0 mph, then sending 5 steps in a row at 0.2, 0.4, 0.6, 0.8, 1.2, 1.5 m/s^2
- Starts from 0 mph then sending at 0.2, 0.4, 0.6, 1.2, 1.5 for 15 seconds then the same magnitude but negative for 15 seconds.
- Starts from 10 or 15 mph then sending negative accelerations at -1.5, -1.7

A. Results

This test featured a set of multiple step response experiments resulting in a plethora of data, so only one full experiment will be shown in this paper. Figure 7 shows a successful full experiment of running the third set of step responses, starting at or near a speed of 0, then sending an acceleration request step response of various magnitudes followed by an immediate equal negative magnitude. As the figure shows, there were some issues with commanding the car from a full stop with small acceleration requests. This is likely due to not overcoming the static friction of the braking system. This was very noticeable in the field, so the experiment was adjusted to bring the car to near 0 speed instead of being fully stopped. Similarly, large acceleration requests were unable to be closely met likely due to the mechanical limitations of the vehicle.

Figure 8 looks at the set of five acceleration/deceleration requests of magnitude $1.2\frac{m}{s^2}$. The initial speed is was a near 0 in each case with a slight variance. This showcases the consistency of the vehicle's response and demonstrates characteristics that are observable in second order systems, like overshoot, rise time, and steady-state error.

VIII. Test 6: PID Controller Validation

The data gathered from the previous test for system identification was used to approximate a model of the vehicle based on acceleration command inputs. With this model in place, a speed controller was designed, translating a speed set point to acceleration commands. The format of controller used was a traditional PID. The benefit of collecting prior system identification information was the possibility to simulate controller designs against the model to check for system dynamics. However, this model is merely an approximation and the only way to observe its true performance is by integrating hardware-in-the-loop testing.

- Type of Vehicle: Toyota Rav4
- Location: Marana, AZ
- Date: April 16, 2021

A. Results

Figure 9 shows the result of applying a PID controller to the Toyota Rav4. The vehicle is only under control through the custom PID controller when the **CRUISE_ACTIVE** signal is non-zero. The disengagement occurred based on the driver re-assuming control of the vehicle through pedal presses, validated from Test 3 in this paper. Shown are the commanded and measured acceleration, along with the vehicle's speed. At around the time of 300 seconds there is a response due to a step input of speed into the PID controller set at 15mph and the initial speed of the car at 0mph. It is also apparent that the acceleration of the vehicle closely tracks the requested acceleration. Similar to the system characterization experiment, the true acceleration does exhibit some overshoot and delay relative to the requested signal which is to be expected. The speed in this case also overshoots, reaching 17.5mph from a 15mph step input. This is likely either due to integrator windup or errors or incorrect assumptions in the vehicle's approximate second-order model.

IX. Conclusion

Upon publication, the processed and aligned data sets, as well as the software notebooks that produced the included plots and perform the presented analysis, will be made available through public license and citeable DOIs for reproduction by the research community.

The data from these experiments demonstrates the development evolution of a self-driving vehicle through means of off-the-shelf hardware and reverse engineering efforts. Even though each data set has a targeted set of behaviors that were desired to be observed, all raw CAN data is available for post-processing further reverse engineering efforts – even outside of the intended use case for a future specific large-scale experiment. By providing access to raw vehicle data and tools for the analysis and plotting of data, other researchers may benefit outside of the original intent.

ACKNOWLEDGMENT

This material is based upon work supported by the U.S. Department of Energy's Office of Energy Efficiency and Renewable Energy (EERE) award number CID DE-EE0008872. The views expressed herein do not necessarily represent the views of the U.S. Department of Energy or the United States Government. Additional support is provided by NSF awards 2135579 and 2151500.

REFERENCES

[1] J. Wenger, "Automotive radar-status and perspectives," in *IEEE Compound Semiconductor Integrated Circuit Symposium, 2005. CSIC'05.* IEEE, 2005, pp. 4–pp.

[2] 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.

[3] W. Lawrenz, *CAN system engineering*. Springer, 1997, vol. 121.

[4] M. Farsi, K. Ratcliff, and M. Barbosa, "An overview of controller area network," *Computing & Control Engineering Journal*, vol. 10, no. 3, pp. 113–120, 1999.

[5] M. Bertoluzzo, P. Bolognesi, O. Bruno, G. Buja, A. Landi, and A. Zuccollo, "Drive-by-wire systems for ground vehicles," in *2004 IEEE International Symposium on Industrial Electronics*, vol. 1. IEEE, 2004, pp. 711–716.

[6] Z. Q. Wei, X. W. Wang, D. N. Jia, and H. Liu, "An information collection and transmission strategy of vehicle state-aware system based on obd technology and android mobile terminals," in *Applied Mechanics and Materials*, vol. 719. Trans Tech Publ, 2015, pp. 573–579.

[7] 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.

[8] R. Bhadani, M. Bunting, B. Seibold, R. Stern, S. Cui, J. Sprinkle, B. Piccoli, and D. B. Work, "Real-time distance estimation and filtering of vehicle headways for smoothing of traffic waves," in *Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems*, 2019, pp. 280–290.

[9] K. McCord, *Automotive Diagnostic Systems: Understanding OBD I and OBD II*. CarTech Inc, 2011.

[10] M. Bunting, R. Bhadani, and J. Sprinkle, "Libpanda - a high performance library for vehicle datacollection," in *The Workshop on Data-Driven and Intelligent Cyber-Physical Systems (DI-CPS), CPS-Week, 2021*, 2021.

[11] T. L. Swetnam, R. Walls, U. K. Devisetty, and N. Merchant, "Cyverse: a ten-year perspective on cyberinfrastructure development, collaboration, and community building," in *AGU Fall Meeting Abstracts*, 2018.

[12] R. Bhadani, J. Sprinkle, G. Lee, M. Nice, G. Gunter, and S. Elmadani, "Strym: A python package for real-time can data logging, analysis and visualization to work with usb-can interface," 2020. [Online]. Available: https://github.com/jmscslgroup/strym