Approaches for Synthesis and Deployment of Controller Models on Automated Vehicles for Car-following in Mixed Autonomy

Rahul Bhadani
rahul.bhadani@vanderbilt.edu
Vanderbilt University
Nashville, TN, USA

Matthew Bunting
matthew.r.bunting@vanderbilt.edu
Vanderbilt University
Nashville, TN, USA

Matthew Nice
matthew.nice@vanderbilt.edu
Vanderbilt University
Nashville, TN, USA

Fangyu Wu
University of California, Berkeley
Berkeley, CA, USA
fangyuwu@berkeley.edu

Amaury Hayat
Ecole des Ponts Paristech
Blaise Pascal 6 et, Champs-sur-Marne
France
amaury.hayat@enpc.fr

Jonathan W. Lee
University of California, Berkeley
Berkeley, CA, USA
jonny5@berkeley.edu

Alexandre Bayen
University of California, Berkeley
Berkeley, CA, USA
bayen@berkeley.edu

Benedetto Piccoli
Rutgers University–Camden
Camden, NJ, USA
piccoli@camden.rutgers.edu

Benjamin Seibold
Temple University
Philadelphia, PA, USA
seibold@temple.edu

Dan Work
dan.work@vanderbilt.edu
Vanderbilt University
Nashville, TN, USA

Jonathan Sprinkle
jonathan.sprinkle@vanderbilt.edu
Vanderbilt University
Nashville, TN, USA

ABSTRACT

This paper describes the software design patterns and vehicle interfaces that were employed to transition vehicle controllers from simulation environments to open-road field experiments. The approach relies on a life cycle that utilizes model-based design and code generation, along with agile software development, and both software- and hardware-in-the-loop testing, with additional safety margins. Autonomous designs should consider the dynamics of mixed autonomy in traffic to safely operate among humans. The software that provides a vehicle's behavior intelligence is often developed through simulation, which may have a mismatch between dynamics, or as a result of a reinforcement learning workflow, which may be a black box with challenges to analyze. In each of these cases, it is important to have research interfaces that provide strongly typed data streams accessible to researchers who are not software experts while continuing to satisfy safety and liveness constraints. This paper describes how we design the hardware platform interfaces and software design process for a mixed autonomy traffic experiment with a leader-follower scenario. Controller synthesis for these vehicles requires clearly articulated vehicle interfaces and software design patterns for successful onboard deployment. Testing strategies for such controllers are also described before algorithms are transitioned to full-scale field experiments with safety operators for the vehicles. Testing strategies include software-in-the-loop simulation testing, hardware-in-the-loop simulation, ghost-car testing, and read-only testing in live traffic. With our approach, we were not only able to validate our controller synthesized in scripts and simulation, but also able to scale deployment to multiple vehicles.

CCS CONCEPTS

• **Computer systems organization** → **Robotic control**; **Robotic autonomy**.

ACM Reference Format:
Rahul Bhadani, Matthew Bunting, Matthew Nice, Fangyu Wu, Amaury Hayat, Jonathan W. Lee, Alexandre Bayen, Benedetto Piccoli, Benjamin Seibold, Dan Work, and Jonathan Sprinkle. 2023. Approaches for Synthesis and Deployment of Controller Models on Automated Vehicles for Car-following in Mixed Autonomy . In *DI-CPS '23: The 3rd Workshop on Data-Driven and Intelligent Cyber-Physical Systems for Smart Cities, May 9, 2023, San Antonio, TX, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3576914.3587711

DI-CPS '23, May 9, 2023, San Antonio, TX, USA

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/3576914.3587711

1 INTRODUCTION

Recent experiments on both closed-road tracks and open-roads with other drivers have enabled mixed autonomy research to grow in complexity while concurrently building confidence in design processes and risk mitigation strategies that permit agile design workflows to thrive.

The seminal mixed-autonomy Arizona Ring Road experiment, conducted in 2016 [1], utilized a closed-road test track. To mitigate

the risk of rear-end collision, the automated vehicle used the Followerstopper, a classical controller that used a piecewise function to modulate its speed depending on its relative distance and relative velocity with respect to its human leader [2, 3].

The deployment of the Followerstopper safety controller used a model-based design approach [4, 5], transforming the same software that was used for data analysis (offline) to be deployed as an executable in ROS through code generation. That same model-based design workflow allowed researchers to conduct adaptive cruise control on an open road in 2018 [6], concluding that existing adaptive cruise control systems on many commercially-available vehicles do not exhibit string stability criteria when operated. For the purposes of this paper, we point out that this 2018 experiment was made possible due to our ability to gather the same data, reformulate and generate the code, and re-run the experiment with different parameters while staying confident in the safety of our trained vehicle operators when operating near the boundary of string instability.

More recently, in 2021, we conducted a mixed autonomy experiment using 5-10 automated vehicles on an open road near Nashville on I-24 [7]. The goal of that experiment was to enable our team to plan to develop a larger experiment at a 10x scale or more. Developing and deploying controllers at such a scale requires rethinking on how to design software and deploy it with efficiency.

In this work, we describe the life cycle of a mathematical controller from its development through theory to its deployment on the physical vehicle for field operational tests. The life cycle consists of model development and refinement, training in the case of the black-box model, testing in simulation, devising a strategy for testing with the physical vehicle, and deployment on open roads.

The deployment process, in this case, significantly departs from traditional software applications as vehicles are safety-critical, and there is a potential for damage and loss of life. Vehicle application engineers and controller designers can use the design pattern discussed in this paper to design and deploy controller models methodically. The paper presents the use case of field operational tests conducted in Nashville in 2021 with a small set of vehicles with a goal to scale to more than fifty vehicles. Two kinds of controllers were deployed through this framework: (i) rule-based or classical controllers, and (ii) black-box, or deep-learning-based controllers.

The rest of the paper is as follows: we first discuss issues around controller development and deployment in Section 2. In Section 3, we discuss some software design patterns for controller refinement, simulation, testing, and deployment. In Section 4, we describe a case study of developing a controller for traffic smoothing and energy reduction. We conclude the paper with some thoughts and takeaway messages in Section 5. Note that this paper mentions a number of tools developed over the years and readers are encouraged to reference various papers such as [8–12].

2 COMMON ISSUES AROUND THE CONTROLLER DEVELOPMENT AND DEPLOYMENT

A common task surrounding car-following controllers is the verification of mathematical models to achieve their objective. However, there are a few issues that are often overlooked:

- **Vehicle dynamics:** The algorithm development of a car-following controller often simplifies the dynamics of the vehicle, considering it as a mass or with simple Eulerian kinematics or a bicycle model. While using a point mass is reasonable for studying the dynamics of overall traffic from a broader perspective, controllers designed with point mass dynamics may exhibit instabilities (or other mismatches) when deployed on a physical system.
- **System delay:** Car-following controllers often behave differently with finite system delay and delay due to communication overhead within multiple computing modules or electronic control units on board the vehicle. Moreover, converting a continuous-time controller to a digital controller during implementation requires selecting a sampling time, controller gain, and other parameters.
- **Vehicle interfaces:** Disciplines interested in car-following control range from computer science to applied mathematics and physics. Many collaborators may be unfamiliar with vehicle interfaces, and the availability of desired system states such as speed, relative speed, speed of the leader car, etc., until near deployment or while in testing. This incentivizes design processes that can enable modifications to the controller after implementation for deployment.

In the above scenario, traditional CPS design practices may lead to brittleness because the classic V pipeline involves designing, validating, verifying, implementing, and testing, which results in the need to redesign and carry out other steps when tests fail. The goal of this paper is to describe how our test interfaces mitigate this risk by requiring model-based approaches that can be re-validated through regression and simulation analysis to confirm that undesired behaviors have been eliminated before redeployment.

2.1 An example of the disconnect between theory and the practice

Early into the mixed autonomy research, Cui et al. [13] developed a second-order car-following model with feedback for stabilizing traffic flow in a ring road of the form in Equation (1).

$$\ddot{x} = f(\dot{x}_j, \dot{x}_{j+1}, x_j, x_{j+1}) \tag{1}$$

In Equation (1), x_j, x_{j+1} are positions of the follower and the leader vehicles in the traffic stream, and $\dot{x}_j$, and $\dot{x}_{j+1}$ are their speed. The above equation provided theoretical benefits, but our testbed, both in simulation and on the physical vehicle [4], had limitations. The testbed lacked the ability to instantaneously command acceleration and did not have velocity as an available input without filtering relative distance, as done in [14]. The situation led to a redesign of the car-following controller in conjunction with testing on the testbed in an iterative and agile manner.

3 DESIGN PATTERN FOR CONTROLLER SYNTHESIS AND DEPLOYMENT

Software design pattern specifies a life-cycle of an engineering solution from math and paper to a digital computer for production that can be repeated over and over again [15, 16]. Software architects specify design practices that identify rules and emphasize abstractions, reusability, and collaborative approach for multi-disciplinary

teams. Next, we describe the development life cycle we took for developing novel car-following controllers.

3.1 Model development

The controller development life-cycle starts with mathematical equations in the form of differential equations, piecewise functions, or state-space models. At this point, mathematicians and algorithm designers are involved in choosing the form of a controller and verifying its properties through scripting in either Python or MATLAB. However, at this stage, consideration for vehicle dynamics is often absent, or the dynamics are merely considered as point-mass.

3.2 Model-based Design

Once an initial draft of the car-following controller is ready, vehicle interfaces are required to be agreed upon that can be used for the abstraction of the vehicle plant as well as the controller model. This requires clearly articulating what inputs are expected to and from the plant as well as the controller and their data types. Then, equations are coded along with abstracted interfaces using pre-defined libraries such as one present in Simulink or created using domain-specific modeling languages such as WebGME [17, 18]. Stand-alone coding artifacts such as a ROS (Robot Operating System) package are generated that can be executed in the simulation and/or with physical hardware. ROS also helped fill the gap between sim and real by introducing real-life system delay, non-homogenous data types, and non-uniform sample frequency among various signals representing states of the vehicle and control commands.

3.3 Software-in-the-loop Simulation (SWIL)

Once the implementation is ready for execution, it is tested in simulation. At this stage, simulation involves modeling a scenario, the dynamics of vehicles, and any communication paradigms to replicate real-life situations such as fixed sample time with slight variations, finite delay, and varied sample time among different input-output signals. The choice of sample time plays an important role as converting a continuous-time domain controller to a digital domain without retuning controller parameters may make the plant output unstable. This situation can be easily replicated using a classic example of an inverted pendulum on a cart. This step, along with all the previous ones, is an iterative process where data or logs from the simulation are recorded and analyzed to meet expectations and ensure the safety properties of the controller.

3.4 Hardware-in-the-loop Simulation (HWIL)

After validation with SWIL simulation, the HWIL approach is taken where a physical platform such as an actual vehicle is used in conjugation with some components of the simulation. In this scenario, the actual car may be made to follow a virtual vehicle, sensor data may come from the simulation, etc. Such a strategy is an elegant solution for safety reasons as even if there is a collision or unsafe behavior, it doesn't happen with the real vehicle.

3.5 Additional Discussion

There are other considerations in the controller design pattern, such as compatibility of the controller model with both simulation and physical hardware, to avoid rewriting any portion of the code. Building such compatibility requires a sufficient level of abstraction supported by various parameters.

In addition to customized software practices, we also adopted standard practices of continuous development and continuous integration and managed our issues through a GitHub Board for issue tracking among a large team of researchers. Each controller development stage was versioned and marked with commit hashes for automatic pulling and deployment on physical vehicles to avoid overwriting approved changes.

4 CASE STUDY OF ENERGY-REDUCING & TRAFFIC-SMOOTHING CONTROLLER

The software design pattern and life cycle for a classical controller deployed as a part of a larger CIRCLES experiment are discussed in this section. The CIRCLES project utilizes ROS for abstraction, vehicle interface, and implementation, while Gazebo is used for simulating 3D rigid-body vehicle dynamics. A high-level schematic of the main objective of the project is illustrated in Figure 1. Referring to Figure 1, an energy-reducing and traffic-smoothing controller is defined as

$$u(t) = f(v, v_{\text{lead}}, x_{\text{rel}}, v_{\text{rel}}, a; \Theta) \tag{2}$$

where v is the driving speed of the vehicle to be controlled, called ego vehicle, v_{lead} is the measured speed of the leader vehicle, x_{rel} is the headway distance between ego vehicle and its leader, v_{rel} is the relative speed of the leader with respect to the ego vehicle, and a is the acceleration of the ego vehicle. $u(t)$ is the reference command which may be an acceleration command or velocity command. Energy-reducing controller is parameterized by Θ. Further, the safety controller is defined as

$$w(t) = f(v, v_{\text{lead}}, x_{\text{rel}}, v_{\text{rel}}, a, u; \Phi) \tag{3}$$

where w is the control command. The safety controller is parameterized by Φ. Parameters Θ and Φ are decided through a series of tests in SWIL simulation, HWIL simulation, and testing controller in live traffic with read-only mode. Subsequent paragraphs describe these strategies.

For classical controllers, it is possible to abstract the energy-reducing controller and safety controller into one block. However, if the initial prototype of the targeted energy-reducing controller is unsafe, then overall operation can be made safe using a supervisory controller such as Followerstopper [2, 19]. For compatibility and data exchange between various components developed by independent subteams, we specified vehicle interfaces with respective ROS topic names, their data types, and relevant signals, as shown in Table 1.

4.1 Classical Controller

For classical controllers, we implemented a modified version of Micromodel controller [20] in Simulink that, along with the ROS toolbox allows code generation into a standalone ROS node for time-triggered execution[1]. We use ROS for execution through a `roslaunch` file for SWIL simulation that executes the leader-follower scenario for collision avoidance and interface checking. The `roslaunch`

[1]We do not use the real-time operating system, but rather depend on ROS and its time-triggered behavior through Linux to emulate this behavior.

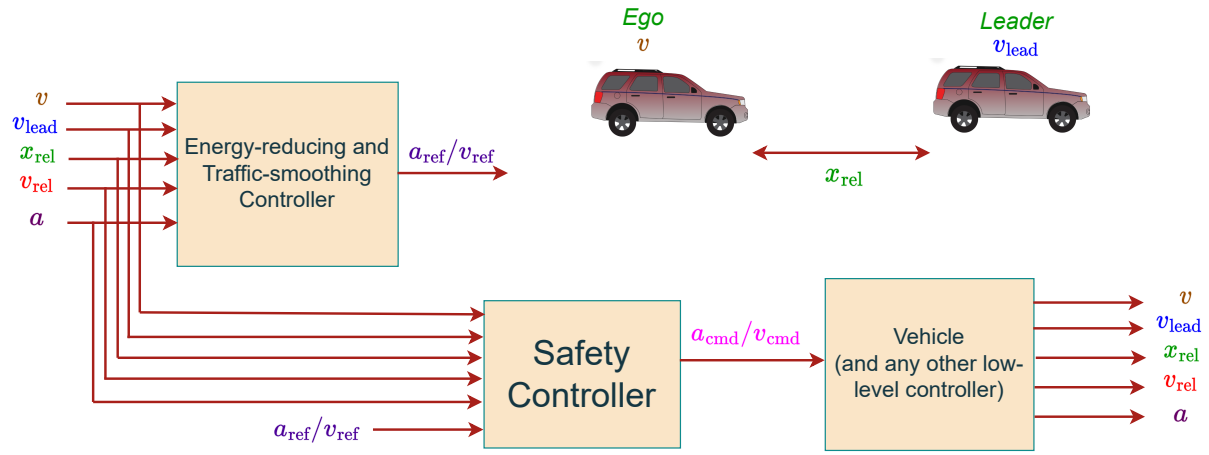

Figure 1: A high-level schematic of the main objective of an energy-reducing and traffic-smoothing controller for car-following strategy in a mixed-autonomy traffic experiment. The controller is designed for the longitudinal movement of the ego vehicle to follow its leader which may be a human-driven vehicle.

Data Type	Data	Topic Name	Data Type	Signal
Input	Ego's driving velocity (m/s)	/vel	std_msgs/Float64	data
Input	Space gap between ego and leader (m) (Toyota-only)	/lead_dist	std_msgs/Float64	data
Input	Relative speed of leader (m/s) (Toyota-only)	/rel_vel	std_msgs/Float64	data
Input	ACC Set Point (mph)	/acc/set_speed	std_msgs/Int16	data
Input	ACC State (encoded integers) off/disabled/enabled/faulted	/acc/cruise_state_int /acc/cruise_state	std_msgs/Int16 std_msgs/String	data data
Input	ACC Vehicle Ahead (0/1)	/acc/mini_car	std_msgs/Int16	data
Input	ACC distance setting (1/2/3)	/acc/set_distance	std_msgs/Int16	data
Input	ACC button press states (Nissan-only) none/resume/set/distance/cancel/system_toggle	/acc/acc_btns /acc/acc_btns_int	std_msgs/String std_msgs/Int16	data
Estimated	Estimated Velocity of the Leader (performed in onnx2ros package)	/v_ref	geometry_msgs/Twist	linear.y
Output	Predicted Acceleration	/v_ref	geometry_msgs/Twist	linear.z
Output	Predicted Speed from RL Acting as v-des or reference speed for Followerstopper	/v_ref	geometry_msgs/Twist	linear.x
Output	Commanded Velocity from Followerstopper to the ego car	/cmd_vel	geometry_msgs/Twist	linear.x
Output	Regions of Followerstopper	/region	std_msgs/UInt8	data

Table 1: Vehicle interfaces with various ROS topic types, their data types, and signal components required for data exchange between various components.

file allows for the parametrization of the controller at runtime and avoids copy/paste or cloning errors during the controller design or code regeneration.

A UML class diagram showing the relationships between various entities abstracted as classes for SWIL simulation is shown in Figure 2. In a leader-follower scenario, SWIL simulation has one leader vehicle controlled in an open-loop manner using real-world trajectory data. The subsequent vehicles are controlled using energy-reducing controllers such as MicromodelController. A SWIL leader-follower scenario is shown in Figure 3.

After approving the SWIL simulation test, we move to the HWIL simulation test. In the HWIL simulation test, a real vehicle is made

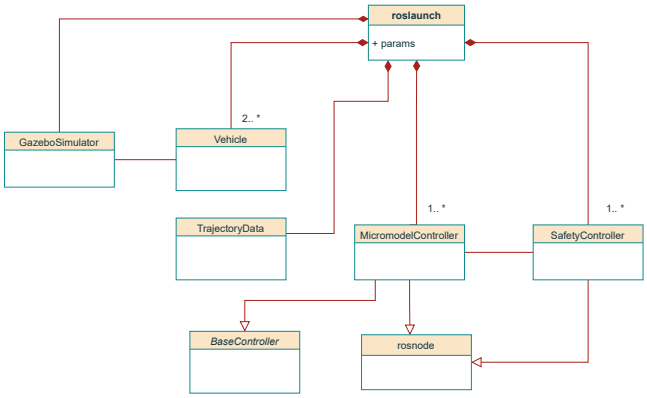

Figure 2: A UML class diagram showing the relationship between various entities for the purpose of SWIL simulation for testing a classical controller.

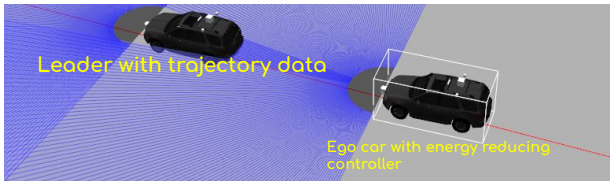

Figure 3: A leader-follower scenario in SWIL scenario with a leader being controlled using real-world trajectory data while ego vehicle follows the leader, controlled by a car-following controller.

to follow a ghost car. To imitate the ghost car, we simulate the relative distance and velocity of the ghost leader by playing back real-world driving data. Such a strategy is safer because it allows us to verify potential collisions without actually damaging the real vehicle. Once the ghost car scenario passes, we test the controller by following a real vehicle. However, there are differences between the SWIL test and the HWIL test. The HWIL test involves interacting with a low-level controller that requires commands in the form of CAN bus messages. To facilitate sending commands to the car in CAN bus message form, we use the CAN-to-ROS package [10] which has the ability to convert CAN bus messages to ROS messages or vice versa. CAN bus message commands are then relayed through the Libpanda library to control the vehicle [8] using comma.ai Panda devices.

Another novel strategy we use for testing our controller in the car is read-only testing, where the controller is deployed and run on the vehicle, but the command doesn't eventually operate the vehicle. Instead, it is intercepted before it reaches Libpanda for control, and we only observe the output. In this case, the vehicle command comes from the vehicle's stock ACC. This strategy is shown in Figure 4.

4.2 Deep-Learning Trained Controller

In contrast to classical controllers, deep-learning-based controllers are data-driven; they require a huge amount of data sets to train

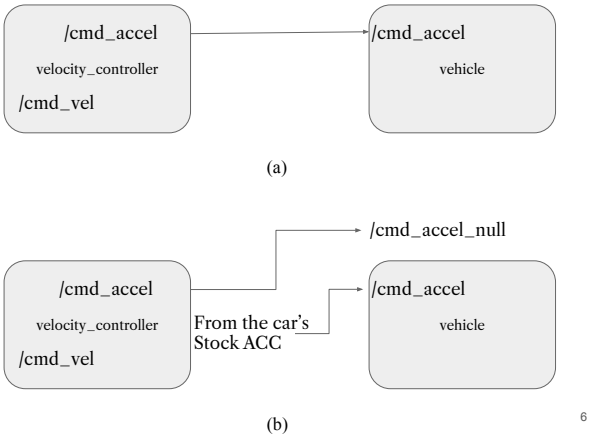

Figure 4: Read-only testing strategy in live traffic. The strategy ensures that we can observe the data for validity and any unsafe behavior. (a) Shows a scenario where directly testing the controller in live traffic may be dangerous if parameters are not tuned correctly. (b) Shows a scenario of read-only testing where the controller command is intercepted and replaced by the command from the vehicle's stock ACC.

the algorithm. In our case, we employed deep reinforcement learning (deep RL) to create a car-following strategy with the objective of reducing energy consumption while achieving uniform traffic flow. We used a modified version of the deep-RL controller mentioned in [11] for the mixed-autonomy traffic experiment. In this case, the objective of the deep-RL controller is to reduce fuel conseumption over a given time horizon. The training resulted in a black-box model that can be used for real-time prediction. To ensure interoperability with ROS, we transformed the model trained using PyTorch into ONNX[2]. Finally, we created the ONNX2ROS package[3], which is a ROS package that enables real-time prediction of control commands based on input states using a black-box trained deep-RL model. Figure 5 shows a UML class diagram that displays the various entities involved in the SWIL simulation of a deep-RL-based controller.

Results from regression testing under a wider variety of traffic conditions, both in SWIL simulation, HWIL simulation, and live traffic allowed us to tune parameters for both classical and deep-RL-based controllers. After successfully passing the regression testing, the experiment was conducted at a scale.

5 CONCLUSION & DISCUSSIONS

In this paper, we describe the software design pattern and lifecycle for taking a controller equation from math to hardware deployment for field testing on a physical vehicle. We employed a model-based design for faster prototyping, SWIL simulation, HWIL simulation, and various other strategies for controller tuning and safety checks before conducting the experiment at scale. Our approaches allowed

[2]https://github.com/onnx/onnx
[3]https://github.com/CIRCLES-consortium/algos-stack/tree/master/onnx2ros

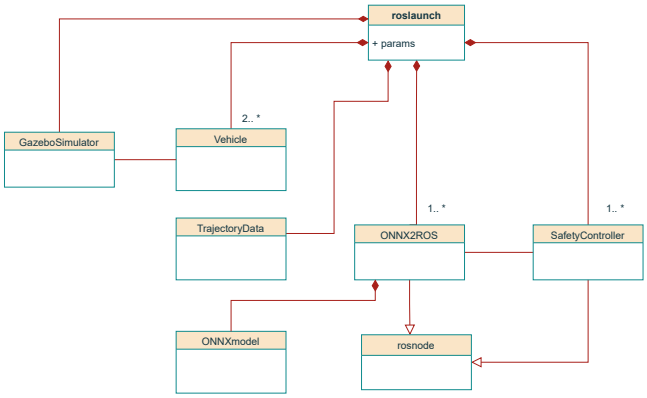

Figure 5: A UML class diagram showing the relationship between various entities for the purpose of SWIL simulation for testing a black-box deep-RL controller.

several dozen iterations per week for refining a controller's parameter selections and tuning, which with traditional software engineering practices, may take months.

6 ACKNOWLEDGMENTS

This material is based upon work supported by NSF (CNS-2135579), the Dwight D. Eisenhower Fellowship program under Grant No. 693JJ32345023 (Nice), and by the U.S. Department of Energy's Office of Energy Efficiency and Renewable Energy (EERE) under the Vehicle Technologies Office award number CID DE-EE0008872. The views expressed herein do not necessarily represent the views of the U.S. DOE or the U.S. Government.

REFERENCES

[1] 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.

[2] F.-C. Chou, M. Gibson, R. Bhadani, A. M. Bayen, and J. Sprinkle, "Reachability analysis for followerstopper: Safety analysis and experimental results," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 8607–8613.

[3] 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*, ser. ICCPS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 280–290. [Online]. Available: https://doi.org/10.1145/3302509.3314026

[4] R. Bhadani, J. Sprinkle, and M. Bunting, "The CAT Vehicle Testbed: A Simulator with Hardware in the Loop for Autonomous Vehicle Applications," in *Proceedings 2nd International Workshop on Safe Control of Autonomous Vehicles (SCAV 2018), Porto, Portugal, 10th April 2018, Electronic Proceedings in Theoretical Computer Science 269*, vol. 269, 2018, pp. 32–47.

[5] R. Bhadani and J. Sprinkle, "Prototyping vehicle control applications using the cat vehicle simulator," *arXiv preprint arXiv:2301.04574*, 2023.

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

[7] S. T. McQuade, C. Denaro, M. Mahmood, J. W. Lee, G. Gumm, J. M. Sprinkle, D. B. Work, B. Piccoli, B. Seibold, and A. M. Bayen, "Medium-scale to large-scale implementation of cyber-physical human experiments in live traffic," *IFAC-PapersOnLine*, vol. 55, no. 41, pp. 83–88, 2022, 4th IFAC Workshop on Cyber-Physical and Human Systems CPHS 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896323001143

[8] 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.

[9] R. Bhadani, M. Bunting, M. Nice, N. M. Tran, S. Elmadani, D. Work, and J. Sprinkle, "Strym: A python package for real-time can data logging, analysis and visualization to work with usb-can interface," in *2022 2nd Workshop on Data-Driven and Intelligent Cyber-Physical Systems for Smart Cities Workshop (DI-CPS)*. IEEE, 2022, pp. 14–23.

[10] 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.

[11] N. Lichtlé, E. Vinitsky, M. Nice, B. Seibold, D. Work, and A. M. Bayen, "Deploying traffic smoothing cruise controllers learned from trajectory data," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 2884–2890.

[12] J. W. Lee, G. Gunter, R. Ramadan, S. Almatrudi, P. Arnold, J. Aquino, W. Barbour, R. Bhadani, J. Carpio, F.-C. Chou *et al.*, "Integrated framework of vehicle dynamics, instabilities, energy models, and sparse flow smoothing controllers," *arXiv preprint arXiv:2104.11267*, 2021.

[13] S. Cui, B. Seibold, R. Stern, and D. B. Work, "Stabilizing traffic flow via a single autonomous vehicle: Possibilities and limitations," in *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2017, pp. 1336–1341.

[14] M. Segata, R. L. Cigno, R. K. Bhadani, M. Bunting, and J. Sprinkle, "A lidar error model for cooperative driving simulations," in *2018 IEEE Vehicular Networking Conference (VNC)*, 2018, pp. 1–8.

[15] R. Xu, "A design pattern for deploying machine learning models to production," *California State University, San Marcos*, 2020.

[16] J. O. Coplien, "Software design patterns: common questions and answers," *The patterns handbook: Techniques, strategies, and applications*, vol. 13, p. 311, 1998.

[17] R. Bhadani, M. Bunting, and J. Sprinkle, "Model-based engineering with application to autonomy," *Complexity Challenges in Cyber Physical Systems: Using Modeling and Simulation (M&S) to Support Intelligence, Adaptation and Autonomy*, p. 10255, 2019.

[18] G. Karsai, "From modeling to model-based programming," in *Leveraging Applications of Formal Methods, Verification and Validation. Modeling: 8th International Symposium, ISoLA 2018, Limassol, Cyprus, November 5-9, 2018, Proceedings, Part I*. Springer, 2018, pp. 295–308.

[19] R. K. Bhadani, B. Piccoli, B. Seibold, J. Sprinkle, and D. Work, "Dissipation of emergent traffic waves in stop-and-go traffic using a supervisory controller," in *2018 IEEE Conference on Decision and Control (CDC)*. IEEE, 2018, pp. 3628–3633.

[20] A. Hayat, X. Gong, J. Lee, S. Truong, S. McQuade, N. Kardous, A. Keimer, Y. You, S. Albeaik, E. Vinistky *et al.*, "A holistic approach to the energy-efficient smoothing of traffic via autonomous vehicles," *Intelligent Control and Smart Energy Management: Renewable Resources and Transportation*, vol. 181, p. 285, 2022.