Libpanda Apps: Managing the Deployment and Reuse of a Cyber-Physical System

MATTHEW BUNTING‡, MATTHEW W. NICE‡,
ALEX A. RICHARDSON‡,
JONATHAN SPRINKLE‡, AND DANIEL B. WORK‡

***Abstract*—A method to rapidly deploy test software in a multi-purpose Cyber-Physical System (CPS) platform is presented in the form of metadata called an app. Logistics in conducting an experiment involving a CPS can be challenging and is often minimally discussed in research results. Generally papers focus mainly on theory, experimental data, and analysis, but CPS research often requires careful validation and deployment efforts. A platform like a self-driving car is also likely to span various experiment use cases, so software must be cleanly uninstalled to avoid conflict in subsequent experiments. Managing a multipurpose yet safety-critical CPS between software changes can add a tremendous amount of setup time that only experts of the particular platform may be able to perform. The concept of experiment apps is presented to reduce experiment setup time in a CPS. The app structure is built to be domain-specific, targeting an open-source self driving vehicle ecosystem developed by Vanderbilt. The effort is done to further democratize the CPS development ecosystem by reducing the burden of software integration.**

***Index Terms*—Connected and Automated Vehicles, Package Management, Field Experiments**

I. Introduction

Conducting an experiment requires a significant amount of research and planning for careful and valid execution. In the context of a Cyber-Physical System (CPS), a stricter sense of experiment validation and verification is often required to prevent any physical damage, imposing safety concerns. The development lifecycle of a CPS may represent an augmented traditional software development lifecycle [1]. The more complicated a system becomes, more effort is required to prepare the experiment. Reducing development time in a CPS is common need [2]. Managing the deployment of a CPS has been done by designing specific software systems that aid in the adoption [3]. Frameworks have been designed to manage the deployment of an arbitrary CPS domain with wireless nodes with the intent to update device firmware [4]

The physical nature of a CPS also can impose time-critical iteration to the experimental software. This occurs when a set of experiments is planned within a specific time window due to logistical constraints. A first experiment may show immediate findings that steer the direction for software changes in subsequent experiments. This exact scenario occurred in a large scale test held in 2022 where a large scale experiment could only be held in a specific daily time window, and the number of resources could only be pooled together in a week's time [5]. This resulted in nightly assessments of experiments to discuss the experiment plan for the following morning. Software had to be developed and deployed in sub-24 hour time windows.

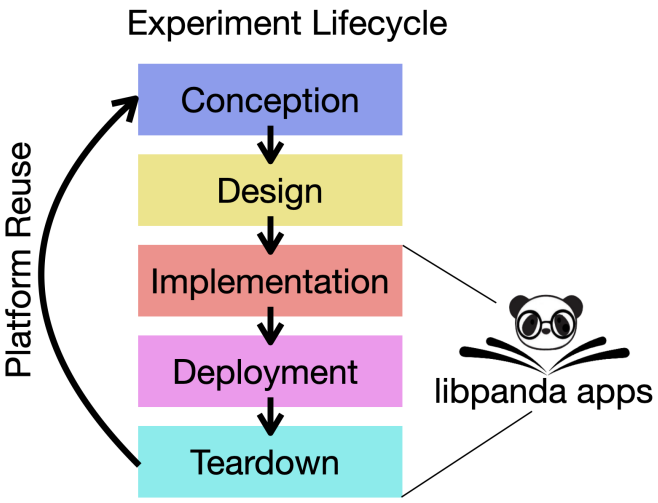

Fig. 1. A basic CPS research project development lifecycle. In the context of a libpanda-enabled vehicle, shown are where libpanda-apps can provide management benefits.

The work presented here aims to provide a framework called libpanda-apps to address the implementation, deployment, and teardown of an experiment by leveraging tools that are already familiar to CPS researchers. A CPS may be a highly valuable asset to a researcher that can be used as a platform for many types of experiments. A basic development approach of an example reusable CPS platform can be viewed in figure 1, showing the stages where the concept of libpanda-apps can provide a benefit. The presented management design solutions is target towards a specific ecosystem, Vanderbilt's Connected Autonomous Vehicle (CAV). Domain-specific development environments can greatly aid the development process of a CPS [6]. While the work here is specifically tailored for a target platform, the ideas may be generalized to other development domains that have a similar set of experimental constraints and lifecycle. The concept of libpanda-apps provides:

- Minimal effort to integrate multiple already-established

This work is supported by the National Science Foundation under awards 2111688 and 2135579, the Dwight D. Eisenhower Fellowship program under Grant No. 693JJ32345023.

‡: Vanderbilt University

979-8-3503-7594-7/24/$31.00 ©2024 IEEE
DOI 10.1109/DESTION62938.2024.00014

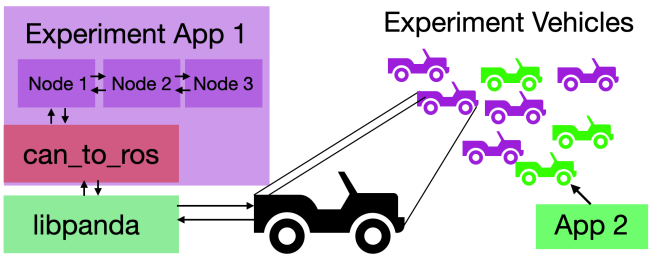

Fig. 2. The Vanderbilt self-driving vehicle development ecosystem. Libpanda acts as a low-level hardware abstraction layer. Can_to_ros abstracts raw data signals into ROS topics. Experiment software, called an App, is typically a set of connected ROS nodes. An experiment may be comprised of any number of vehicles, running multiple instances of the ecosystem with different apps.

project repositories.
- Mutual exclusion for hardware interfaces.
- Traditional software dependency installs.
- Minimal command interface to configure, install, and tear down experiments.
- Bluetooth-based interface for network sparse experiment locations.

II. Background

1) Target CPS Platform: Vanderbilt University has developed a low-cost self-driving vehicle ecosystem that democratizes the sensing and actuation in stock Toyota RAV4s. The vehicle has many use cases for experiments, typically involving traffic-influencing experiments held on open freeways. The nature of conducting traffic experiments results in the vehicle platform being located away from a laboratory setting and therefore away from easily-accessible internet. Importantly, a vehicle requires regular maintenance so experimental software must be routinely disengaged when finishing an experiment.

The vehicle ecosystem is based on interfacing with the vehicle's OEM Controller Area Network (CAN) bus and modules. Using Raspberry Pi 4 running Raspbian, the software library libpanda abstracts the hardware interfaces into CAN and GPS data [7]. Libpanda also features open-source hardware called the mattHAT that interfaces the Raspberry Pi to the CAN bus. Based on efforts to decode CAN signals [8], a package named can_to_ros [9], [10] serves as a middleware abstraction of specific CAN signals into Robotics Operating System (ROS) topics. Data can be analyzed quickly using the Python package Strym [11]. To further democratize the platform for other researchers or for naturalistic driving data purposes, a frontend named Privzone [12] provides a bluetooth-based method to configure privacy settings for data gathering.

The workflow of installing developed experiment software on Vanderbilt's self driving vehicle test ecosystem typically involves a skilled expert in the platform. The developer must be familiar with navigating the system and be able to perform the following:

- Network connect to the Raspberry Pi over SSH.
- Uninstall or disable any program that starts on boot that uses the mattHAT.
- Check all dependent development repositories for installed version, and checkout the correct version if needed.
- Clone dependent ROS repositories if missing. Make sure to follow development environment ontology.
- Transfer experiment-specific software.
- Compile all changed local code.
- Configure startup scripts to invoke experiment software on boot.

2) Operating System Package Managers: Software updates through the use of package managers is well established in various operating systems. The Advanced Package Manager (Apt) is a widely used package manager in Linux Debian-based distributions [13]. Apt's commands allow updating of multiple packages with very few commands. The required metadata for each package describes dependencies, and Apt can automatically resolve and install requirements on the dependent tree. Specific versions of software packages can also be selected for installation.

Apt and similar package managers like Python's Pip are not deemed to be well suited for managing experiment software largely due to its intention for completed, releasable code. Package managers involve a general-use structure for software that exceeds the domain of experimental software, convoluting the packaging process in domain-specific applications. Each time code changes, even in the case of a simple value change, a new package would need to be compiled. Package managers also require a particular web host of packages, meaning that a developer of experiments would need to coordinate a method to store packages on the internet. Both of these hurdles would involve experiment developers becoming experts in the area of package management and distribution, adding time to the deployment process.

3) Docker: Docker containers provide a structure of emulating a separate environment within an OS and has been a tool used in reproducible research [14]. Through minimal metadata, a Dockerfile acts as a minimal script to define dependencies and perform any setup routines. This can be leveraged for multiple experiment setups by having separate Dockerfiles for each experiment. This lends itself well to reproducibilty from the software aspect of a CPS.

Similar to package managers, Docker is intended for general use. This can lead to unconstrained implementations when targeting a specific platform. In the case of the Vanderbilt vehicle ecosystem, Docker itself is not aware of the domain constraints. Therefore a manager system would still be required to ensure proper Docker containers do not violate platform-specific constraints.

III. Design

A. Auto Updates using Git

The inception of the libpanda-app management system stems from the needs to update experiment software on a large scale of Raspberry Pis. The Raspberry Pis were installed in vehicles as a headless configuration, located near facilities with

a WiFi access point. Due to the scale, updating all Raspberry Pis manually would have involved operating the Pis with portable peripherals (monitor/keyboard) or an SSH connection after ensuring they are powered and booted. This would be a very time consuming process, potentially logistically impossible within the experiment timeframe. Code was developed and hosted under different GitHub repositories so each contributor could push updates.

To mitigate the large update effort, an automatic update routine was installed on each Raspberry Pi. Using **systemd**, the Pi's OS would invoke a network status check. If github.com was available, a script would enter all cloned repositories and pull and updates. After pulling, the required compilers would execute like ROS's **catkin_make**. Each repository's information is stored in a CSV formatted file.

Pushing code updates can raise a project management red flag regarding the ability to compile code. Also a push could be made in error that changes the intended behavior for the experiment. To circumvent these issues while still allowing developer to have unhindered use of github, the CSV file that stores repository information also stores git hashes of specific commits to be checked out. The git hashes work on any branch in a repository, preventing the need to also specify the branch. This enforces a validation step in the experiment software release process where the manifest of github hashes had to be carefully chosen and assembled.

This preliminary effort was effective for a single experiment type but was not easy to scale out to future experiments. To mitigate this while still leveraging git's familiarity and version tracking, a manager system was designed.

B. App Manager

Apps are managed by a bash script named the **libpanda-app-manager** (LPAM). The LPAM is installed as a part of the installation of libpanda. From the command line, the LPAM can install external repositories, install apps from the repositories, start and stop the app, or display status information. The LPAM provides a concise set of functions through commandline arguments:

- -d: Show details of all apps
- -g, -r: Install/Remove a github-located app repository (with optional branch/hash)
- -i -u: Install/Uninstall an app from the repository list
- -s, -k: Start/Kill the installed app
- -p: Update all repositories

The LPAM is dependent on knowing the current state of all configured repositories and the currently installed app. This information is recorded and updated by the LPAM as a manifest file. The manifest is in a YAML format, located in a libpanda configuration directory.

Adding a repository is done by specifying the github owner and repository name. The LPAM will perform a clone operation in a configuration directory. If successful, the optional branch or hash information is checked to ensure that it exists in the repository. If not, then the manifest is not updated, the repository is deleted, and an error is reported. If everything checks out, then the manifest updates the manifest with repository information along with the app metadata described in section III-C. Figure 3 shows an example if installing a repository successfully, printing the configured apps.

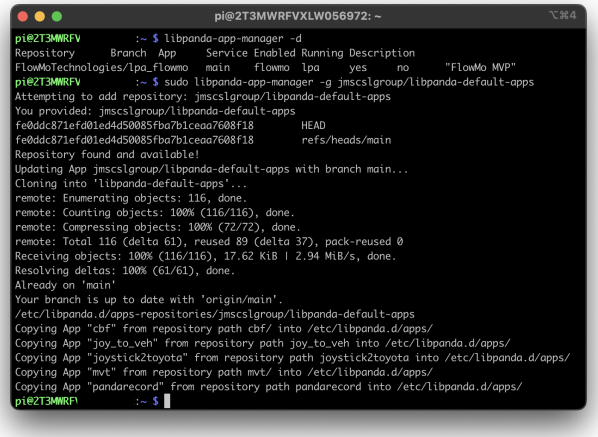

Fig. 3. The command line interface invoking the LPAM. This example shows how an app repository was added and the recognized apps that were within the added repository.

The install command will either provide a menu of available apps to be installed based on a name defined the repository metadata, or install based on a provided name. An installed app is expected to start on boot, therefore systemd is leveraged. The installation will check to see if the app is either a bare executable or ROS launch file. In the case of a ROS-based app, the robot_upstart ROS package is invoked with the app's provided launchfile to generate the systemd configuration. Similarly, the appropriate uninstall process is chosen based on the app definition.

Due to the target platform where only a single program can access the mattHAT at a time, mutual exclusion is enforced. This means that only one app can be installed at a time. This is easily handled by the LPAM when installing a new app. If another app already exists, then the LPAM will check the manifest and uninstall the old app before installing the new app. By keeping track of the state of installed apps, the user does not need to know how to remove the old software since this step will be automatically handled, greatly reducing system debugging and reconfiguration time.

C. App Repository Structure

A github repository must have a YAML configuration file with app metadata, along with directories of scripts corresponding to each app. An example structure may look like the following:

```
https://github.com/<group>/<repository>
|-- libpanda-apps.yaml
|-- app_name_1
|    |-- start.sh
|    |-- stop.sh
```

```
|    |-- description.txt
|-- app_name_2
     |-- description.txt
```

This structure imposes a minimal set of additional artifacts to be added to a repository. The YAML file describes a set of dependencies of each app along with some configuration parameters. The directory of scripts provides a way to run commands during installation or starting/stopping an app.

The following libpanda-apps.yaml example has two apps, pandarecord and joy2veh. The pandarecord app is a basic utility provided by libpanda and needs minimal configuration. The joy2veh is an example ROS node that is dependent on an Apt package, a Pip Python library, and ROS nodes in three different repositories. The LPAM will invoke an Apt and Pip install of the dependencies if the app is to be installed.

```
apps:
  - name: joy2veh
    properties:
    dependencies:
      - apt: libdiagnostic-updater-dev
      - pip3: numpy
    ros_repositories:
        - owner: jmscslgroup
          repo: can_to_ros
          branch_or_hash: 17dd12e9
        - owner: MatthewNice
          repo: joy2veh
          branch_or_hash: indigo-devel
        - owner: MatthewNice
          repo: joystick_drivers
  - name: pandarecord
    properties:
```

The ROS repositories are cloned into a catkin workspace that features other parts of the Vanderbilt ecosystem, including can_to_ros. Each specified ROS repository has an optional **branch_or_hash**. If a hash is provided (can_to_ros in the example), the code will remain the same on each LPAM update. This also lends will to experiment reproducibility. If a branch is provided (joy2veh), then the cloned repository will always pull updates from the provided branch. If no branch or hash is provided (joystick_drivers), then the repository will always track the default main branch.

The app metadata can be added to any github repository. However, note that a repository does not need to include and source code for the app. An example of this is the set of default apps shown as an example located at https://github.com/jmscslgroup/libpanda-default-apps. This repository only contains app metadata which references other repositories with functional source code.

D. Auto Updates

libpanda's auto-update is performed both on boot and on shutdown. This is done in both stage due to the environment of running inside a vehicle, where WiFi access is likely to exist at either the end or start of trip. The update procedure has been modified to include updates of apps. The update procure liaisons app updates through the LPAM with scripted LPAM calls. This ensures that the manifest maintains integrity.

The LPAM manifest is versioned in order to handle added features that may impact the ontology of its contents. The LPAM is built with a version migration framework to aid in the event that an old system is no longer valid with an updated LPAM, ensuring the system can be backward compatible.

Automatic updates can be seen as a three step process in the context of the LPAM. First the libpanda update procedure updates itself, including any revisions made to the LPAM. Regardless if libpanda or the LPAM had any changes, the LPAM is then also invoked to pull any updates from any of the configured app repositories. This pulls any updates made to the app configuration YAML in the repository. Lastly, the current installed app undergoes an uninstall and reinstall, resulting in pulling any updates from the ROS repository dependencies or installing additional Apt or Pip dependencies.

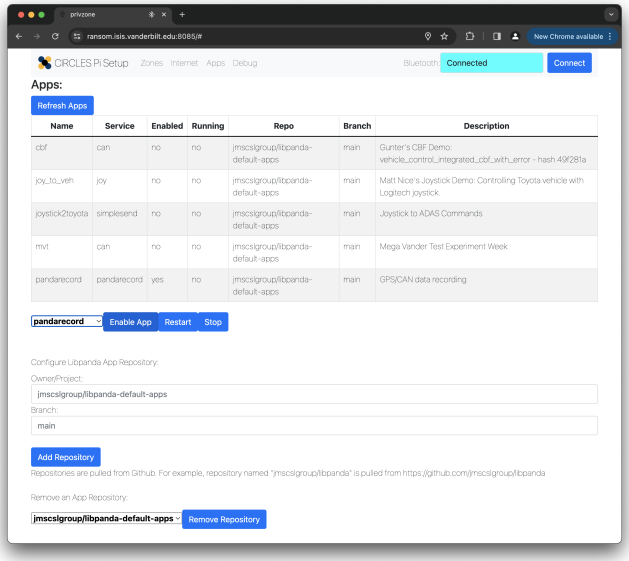

Fig. 4. An augmented Privzone showing the state of the installed apps. The table quickly tells the user that the installed(Enabled) app is pandarecord, and is not currently running. The dropdown menu is available to select various apps to be installed, and can be restarted or stopped with a button click. App repositories can also be added and removed from the interface.

E. Web-based Bluetooth Frontend

The LPAM reduces the complexity of installing apps, but it does not circumvent the issue of managing a CPS in a network sparse field. By leveraging the existing frontend Privzone [15], configuration views have been augmented to include an abstraction the LPAM. Privzone is able to connect to a Raspberry Pi that has libpanda installed through a over Bluetooth Low Energy (LE)-based backend service named Bluezone. By adding more communication protocols, states of the LPAM are made more readable on Privzone's frontend. Figure 4 shows the app interface.

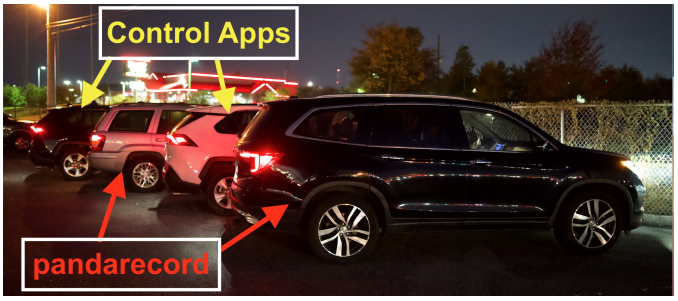

Fig. 5. Four vehicles equipped with libpanda-apps. Two vehicles were installed with a default pandarecord app for pure data collection and the other two had installed an experimental control app. The vehicles were staged offsite away from WiFi. The experiment setup was verified using Privzone, shown in figure 6

IV. Case Studies

A. App as Development Tool

An experiment idea was proposed to have a single vehicle get infrastructure-provided information about variable speed limit information [16]. This project had a short deadline of two weeks, and had a few components that needed to be developed. Because libpanda-apps were a part of the ecosystem, more time was able to be spent on developing, and leading to rapid iteration.

One of the tasks needed for the system was velocity controller. Due to the complicated system dynamics, it was deemed best to tune the controller using the empirically based Ziegler-Nichols method. The controller was designed using MATLAB's Simulink, generating a standalone ROS node. During tuning, the control designer with code generation tools located offsite due to scheduling issues. However, personnel running the vehicle could advise new tuning parameters remotely. The control designer was able to push changes to git, and the vehicle operators could invoke LPAM updates to pull and install the changes. This structure greatly increased productivity and kept focus on the development task of controller tuning rather that software configuration.

B. Multiple Vehicles, Multiple Apps

A new experiment following the efforts in section IV-A was to minimally modify the function of the app, but scale the experiment to include a total of four vehicles. In this experiment, two vehicles would be controlled while two other vehicles would be equipped as data-logging vehicles. The vehicle can be seen in the experiment staging area away from network access in figure 5.

Reusing software greatly reduces development time in an iterative change like those required, however changes in software can make reproducing old results challenging. The app infrastructure lends itself well to this situation since the configuration of software packages and corresponding versions are captures in the repository YAML. Creating a new app for this experiment meant to duplicate the app metadata in the YAML with a new name and configure the software packages

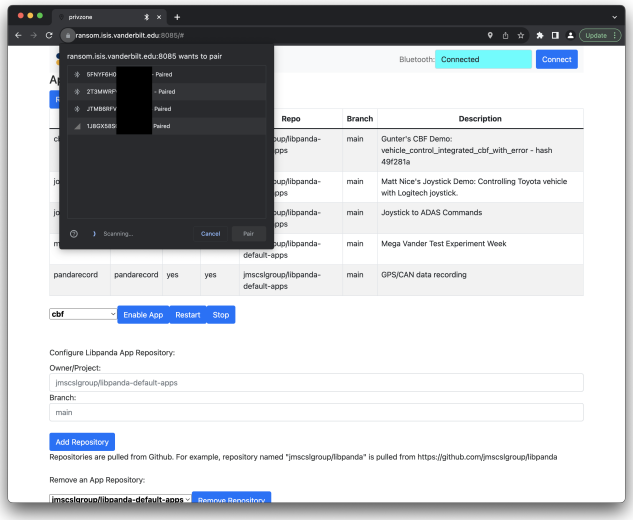

Fig. 6. Experiment vehicles being checked using bluezone-connected libpanda-apps, showing the broadcasted VIN in the bluetooth device search. All cars were quickly verified to be running with correct software.

to invoke the new code, preventing any functional changes to the prior experiment's codebase.

Development of the experimental software was only performed on one vehicle. Auto updates removed the need to constantly maintain the second control vehicle. Once the experimental software was deemed valid, the second vehicle would automatically pull app updates.

The experiment was intended to take place during particular traffic conditions that can drastically change in a short timespan, causing scheduling challenges. Verification in the field was performed to ensure all proper apps versions were installed on associated vehicles. Bluezone was effective in this regard by not needing a WiFi network nor needing any hardware connections with peripherals like a monitor and keyboard. Figure 6 shows a screenshot of Privzone during software verification with the four vehicles appearing as bluetooth devices. Each vehicle was checked for the current version and if it was currently running in a short time, preventing any hurdles that might have compromised experiment timing.

Beyond this experiment, numerous other experiment apps were installed, exhibiting the easy app transition process. After a couple months, a demonstration was planned to show recent research products, and this particular experiment was deemed to be the best showcase. Since the repository was already configured, Privzone easily reinstalled the experiment, saving personnel setup time. This is also serves to demonstrate the new reproducible aspect of the ecosystem.

V. Conclusion

Libpanda-apps have shown to be an invaluable tool that opens up the possibilities of research within tight logistical deadlines by greatly reducing the setup time cost. Better development times of new apps can be achieved by leverage the update features with multiple code repositories. The auto

updates can also lend itself to software to many platforms. YAML configurations can be reused in new app creation with a copy then modify workflow. The app structure also greatly helps in reproducing past experiments by reducing the burden to relearn old setup techniques.

A. Future Work

Further efforts will be integrated over time to increase the efficacy of the system. One improvement is to reduce the amount of configuration files required to define a project, including the things like the start and stop scripts. A more configurable ontology could also be implemented so that the root directory is not the required location for the app directory.

While mutual exclusion is required in terms of hardware control, the specific ROS node that controls the vehicle could be its own separate launch file. Doing so would eliminate the need for mutual exclusion, and simultaneous apps could be a possibility.

App arguments could be defined so that Privzone could send dynamic parameters much like that in a command line. This could eliminate the need to push code updates when only changes by value are needed between experiments.

References

[1] T. Bures, D. Weyns, B. Schmer, E. Tovar, E. Boden, T. Gabor, I. Gerostathopoulos, P. Gupta, E. Kang, A. Knauss *et al.*, "Software engineering for smart cyber-physical systems: Challenges and promising solutions," *ACM SIGSOFT Software Engineering Notes*, vol. 42, no. 2, pp. 19–24, 2017.

[2] M. Obstbaum, U. Wurstbauer, C. König, T. Wagner, C. Kübler, and V. Fäßler, *From a Graph to a Development Cycle: MBSE as an Approach to reduce Development Efforts*. Deutsche Gesellschaft für Luft-und Raumfahrt-Lilienthal-Oberth eV, 2017.

[3] H. Yu, H. Qi, and K. Li, "A powerful software-defined cyber-physical system to expand cps adoption," *Software: Practice and Experience*, vol. 52, no. 4, pp. 904–916, 2022.

[4] M. Szczodrak, Y. Yang, D. Cavalcanti, and L. P. Carloni, "An open framework to deploy heterogeneous wireless testbeds for cyber-physical systems," in *2013 8th IEEE International Symposium on Industrial Embedded Systems (SIES)*. IEEE, 2013, pp. 215–224.

[5] M. Nice, M. Bunting, A. Richardon, G. Zachar, J. W. Lee, A. Bayen, M. L. Delle Monache, B. Seibold, B. Piccoli, J. Sprinkle, and D. Work, "Enabling mixed autonomy traffic control," *arXiv preprint arXiv:2310.18776*, 2023.

[6] J. El-Khoury, F. Asplund, M. Biehl, F. Loiret, and M. Törngren, "A roadmap towards integrated cps development environments," in *1st open EIT ICT labs workshop on cyber-physical systems engineering*, 2013.

[7] 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.

[8] M. W. Nice, M. Bunting, G. Zachar, R. Bhadani, P. Ngo, J. Lee, A. Bayen, D. Work, and J. Sprinkle, "Parameter estimation for decoding sensor signals," in *2023 ACM/IEEE 14th International Conference on Cyber-Physical Systems (ICCPS)*, 2023.

[9] 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.

[10] M. W. Nice, M. Bunting, J. Sprinkle, and D. Work, "Middleware for a heterogeneous cav fleet," in *2023 5th Workshop on Design Automation for CPS and IoT (DESTION)*, 2023.

[11] 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.

[12] M. Bunting, M. W. Nice, D. Work, J. Sprinkle, and R. Golata, "Edge-based privacy of naturalistic driving data collection," in *2023 ACM/IEEE 14th International Conference on Cyber-Physical Systems (ICCPS)*, 2023.

[13] J. G. MacKinnon, "The linux operating system: Debian gvu/linux," 1999.

[14] C. Boettiger, "An introduction to docker for reproducible research," *ACM SIGOPS Operating Systems Review*, vol. 49, no. 1, pp. 71–79, 2015.

[15] M. Bunting, M. Nice, D. Work, J. Sprinkle, and R. Golota, "Wip abstract: Edge-based privacy of naturalistic driving data collection," in *Proceedings of the ACM/IEEE 14th International Conference on Cyber-Physical Systems (with CPS-IoT Week 2023)*, 2023, pp. 256–257.

[16] 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.