ACTA IMEKO
ISSN: 2221-870X
December 2024, Volume 13, Number 4, 1 - 6

Modified vehicle architecture for rapid prototyping

Jan Sobotka[1], Jiří Novák[1], Jiří Pinkava[1]

[1] *Czech Technical University in Prague, Prague, Czech Republic*

ABSTRACT
A modern car is made up of a considerable amount of software running on many Electronic Control Units interconnected by communication infrastructure. Together, it creates a rigid system that is not easy to modify to run modified or custom code to implement new vehicle functions or collect scientific data. The paper presents a modified (electric) vehicle architecture that allows rapid prototyping of new software-oriented features. The architecture enables the control of a specific vehicle, the acquisition of arbitrary vehicle data, and offers an interface for communication with the driver. The solution is based on the hardware modification of the production car. The vehicle is equipped with an Advantech industrial computer, which creates an environment to run the prototyped software. All platform services are available through the REST API.

Section: RESEARCH PAPER

Keywords: vehicle; rapid; prototyping; software; function; testing; e/e architecture; gateway; CAN; message; manipulation; software-defined

Citation: J. Sobotka, J. Novák, J. Pinkava, Modified vehicle architecture for rapid prototyping, Acta IMEKO, vol. 13 (2024) no. 4, pp. 1-6. DOI: 10.21014/actaimeko.v13i4.1774

Section Editor: Platon Sovilj, University of Novi Sad, Serbia

Received February 17, 2024; **In final form** May 13, 2024; **Published** December 2024

Funding: [This research has been realized using the support of Technology Agency of the Czech Republic, programme National Competence Centres, project #TN01000026 Josef Bozek National Center of Competence for Surface Transport Vehicles. This support is gratefully acknowledged.

Corresponding author: Jan Sobotka, e-mail: jan.sobotka@fel.cvut.cz

1. INTRODUCTION

Passenger cars are an object of research activities in many research areas. These activities include, for example, vehicle stability control, passenger comfort, simple and advanced driver assistance systems [1], or various optimizations of energy consumption. Implementing a new algorithm or feature into a standard production car to validate it is often as challenging for an independent researcher as the research itself. The main reason is that cars are made up of proprietary software. Today, passenger cars mostly use a domain-oriented architecture of in-vehicle electronic systems, which relies on a large number of electronic control units (ECUs) interconnected by different networking technologies [2]. Details about particular ECU functionality [3], as well as data communication [4] over vehicle networks, are subject to public access restrictions. In addition, each car manufacturer uses its own architecture of internal electronic systems and in-vehicle networking. Understanding internal vehicle communication requires access to the corresponding data matrices. Complexity and lack of information for implementation in a real car often lead to validation of research results on vehicle or subsystem models [5]. These models, either intentionally or by mistake, often miss important aspects of reality, and the validity of the research validation is then limited. In some cases, the research results should be validated by the driver and passengers (user experience); here, the actual implementation of the research result is essential. To provide a solution, the HW/SW framework was designed and developed, which hides the details of vehicle-dependent implementation. It provides a high-level API allowing information acquisition from the vehicle (e.g., vehicle speed, internal and external temperature, seat occupation, air condition settings) and simultaneous control of selected vehicle functionalities (e.g., air condition settings, adaptive cruise control settings, infotainment settings, drive mode settings) in a vehicle (and possibly the vehicle manufacturer) independent way. Thus, a new prototype functionality can reside at the top of the framework, utilize the data available from vehicle information systems, and control vehicle settings according to the researcher's goals. The proposed framework is based on communication modification that allows control of the desired functionalities. This article is an extended version of the conference proceedings [6].

2. E/E ARCHITECTURE MODIFICATION ANALYSIS

Electronic functionalities are implemented in various ways. It depends on the E/E (Electrical/Electronic) architecture used [7]. The typical vehicle architecture evolves from several isolated systems to communicating specialized ECUs to general ECUs over time. The authors of [8] studied the possibilities of car modification for autonomous driving prototyping. According to the distribution of functions, the system can be divided into zones. Communication between zones is protected by single or multiple domain controllers. The actual trend is limiting the number of ECUs towards a centralized architecture [9]. This section analyses the possibility of modification for rapid prototyping of new software-defined functionalities. From a prototyping perspective, the situation is supposed when an ECU is connected to a vehicle ecosystem at least by a single communication link.

2.1. ECU Software Modification

A natural way to quickly prototyping a new feature is to modify the ECU software. The situation is complicated by two facts. First, running your own code on an ECU without manufacturer support is a challenging task [10], as often even the car manufacturer does not have the original source codes available. Second, I/Os are distributed over multiple ECUs; thus, code modification in a single ECU might not be sufficient. For rapid prototyping, it is necessary to have the ability to extend existing code for a new feature. Running its own software on ECU hardware thus leads to the implementation of complete ECU software, including communication with the rest of the car. This is nearly impossible for an independent researcher.

2.2. Open Source ECU

The prototyping platform in the form of an open ECU is another direction. The suitability depends on the application domain. Many of these projects are intended to replace the original electronic fuel injection (EFI). The typical application area is budget motorsport projects. For example, Speeduino project, based on Arduino Mega 2560 project [11]. For general-purpose ECUs, there are several OS projects. Let us mention EB Corbos Linux [12] and Automotive Grade Linux [13]. It can create a basis for the implementation of an own ECU for prototyped functionality. The problem is similar to that discussed in the previous section. In addition to the prototyped function, the open ECU has to be integrated into the existing car ecosystem.

2.3. Physical Signals Modification

Every vehicle electronics system uses some sensors and actuators. Modification of input signals, as well as output signals, is the first possibility for alternating the system behaviour. This approach is widespread in Hardware-in-the-loop (HIL) testing [14]. The disadvantage of this technique is its limited generality. Different types of sensors and actuators require specific solutions. Also, the concurrent usage of I/O for an existing and prototyped function could be problematic.

2.4. Communication Modification

Modern passenger cars are based on internal communication. Various communication standards are used. The most commons are CAN/CAN FD, LIN, Automotive Ethernet, and FlexRay [15]. In the rest of the paper, the CAN/CAN FD technology is assumed. A widespread scenario is communication among the ECUs based on the exchange of entities called signals. An example of a signal is the outside temperature measured by an ECU and transmitted to other network nodes by CAN bus [16]. Switch status (pressed/released) checked and transmitted in respective CAN message by one ECU toward another node equipped by an actuator can be another example. The message is received by other ECU and used to control the output as required. To change the switch state, the dedicated communication device that filters and/or modifies the original message and sends a modified one toward the ECU of interest should be used.

2.5. Service-Oriented Architectures

Not only signal communication is used in modern vehicles. Service-oriented protocols are becoming an integral part of modern cars. These protocols can either be proprietary or open. An example of an open protocol is Scalable service Oriented MiddlewarE over IP (SOME/IP) [17] as a part of AUTOSAR. A representative of a proprietary protocol can be BAP (Bedien- und Anzeigeprotokoll), used in Volkswagen Group vehicles. The advantage of services-oriented communication is that requests for service provided by ECU can come from any other ECU. And thus, the service requests can be generated using a suitable communication interface connected to a bus.

3. CAR ARCHITECTURE FOR RAPID PROTOTYPING

The new platform proposed for rapid prototyping of new vehicle functions uses the principles described above in paragraphs 2.4 and 2.5. The central element is the Advantech ARK-3520P industrial computer. The interconnection between devices is depicted in Figure 1. The technologies used are Linux based. In the case presented, Ubuntu 22.04 is used as an operating system. PC is extended by Kvaser Hybrid Pro 2xCAN/LIN interfaces. The software architecture is shown in Figure 2. The structure is divided into car-dependent and car-independent layers.

All platform services are available on the Car REST API [18], which is independent of the vehicle (and the vehicle manufacturer). The car-dependent layer consists of three modules, providing for vehicle data acquisition, required vehicle functionality control, and access to vehicle infotainment screen (user-defined screens, driver touch responses). Each of these software modules is supported by the hardware modules that physically interfere with the car's functions (see Figure 1).

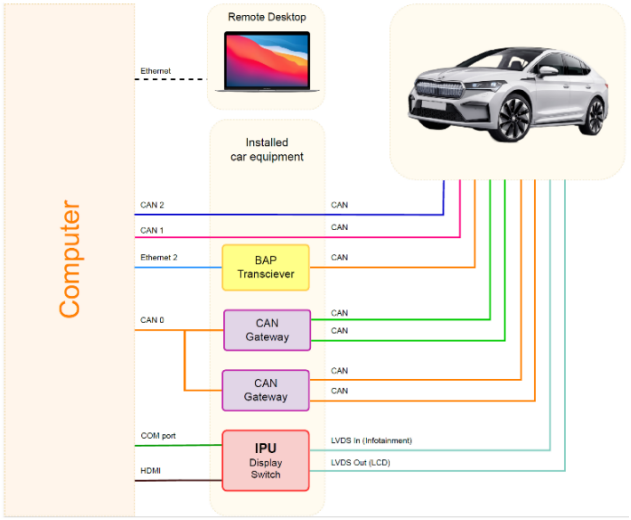

Figure 1. Car Hardware Setup.

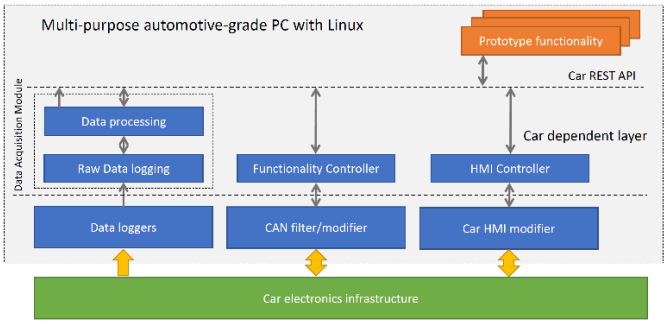

Figure 2. Car Architecture for Rapid Prototyping.

3.1. Data Acquisition Module

The Data Acquisition [19] module is implemented using the Data loggers HW layer, consisting of external CAN FD/USB interfaces. They are supported by the native Linux CAN interface SocketCAN. DAQ module receives CAN messages from an in-vehicle network and interprets them using an industry standard DBC definition file in physical quantities. At the same time, the BAP protocol messages (which are transmitted in specific CAN messages) are parsed, providing complete information about the state and activities of the vehicle. The stream of acquired vehicle signals (in JSON format) is pushed to the configured TCP endpoint (using the Car REST API) from the prototyped application. Each signal value is accompanied by its timestamp. Acquisition from multiple channels is supported.

3.2. Functionality Controller Module

The preferred way to control the functionality of the car is to manipulate vehicle communication networks in real time. The structure of the data manipulation system is shown in Figure 3. To independently control selected vehicle functions, the modules called CAN Gateway (which provide real-time manipulation of a CAN message) are inserted into the internal vehicle communication paths. Their activities (which are independent of specific vehicle functionality that should be initiated) are controlled by a Functionality Controller module. The CAN Gateway allows one to block / pass on selected messages, modify their content (with respect to some application layer constraints), and transmit / monitor chosen messages. The microcontroller used (STM32G4 line) is powerful in its ability to handle the application CRC code if it is used. The CAN Gateway

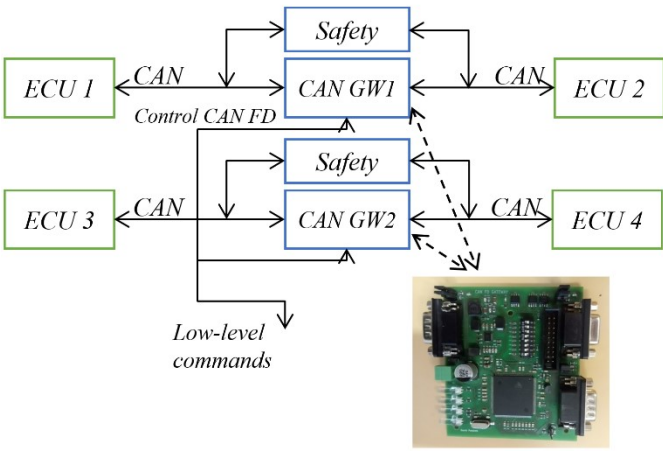

Figure 3. CAN Gateway Application Principle.

Table 1. CAN Gateway parameters

MCU	STM32G4x4
CAN interfaces	2 + 1 (control)
Communication standards	2.0 B / CAN FD
Communication speed	up to 8 Mbit/s
Control interface	CAN FD
Connection	3 × D-SUB 9

functionality is vehicle and function independent. Thus, the list of supported functionalities can be extended without firmware modification (only the Functionality Controller module will require a software update in such a case). Service-oriented communication used to control specific car functionality is also implemented within this SW module. Flexible-Data CAN variant is also supported. Each CAN gateway is equipped with an emergency circuit; In case of any problem, the CAN gateways are bridged, and the original CAN interconnection is restored to keep the driver and passengers safe. The parameters of the module are summarized in Table 1.

3.3. HMI Controller

Human-Machine Interface (HMI) Controller module provides for communication with driver/passengers. It relies on the video inserter and the CAN Gateway hardware support. The general structure of the HMI subsystem is shown in Figure 4. The video inserter is placed between the vehicle Infotainment ECU and its display. It allows switching between the standard video source (infotainment ECU) and the user prototype generated video (HDMI output of the Linux computer). If the SW prototype needs to use the car display, the display is switched to the prototyped screen and the user response is awaited (see Figure 5). The display is connected through the FDP-Link interface. The display unit supports touch-screen functionality - user actions are reported from the display to the infotainment ECU by CAN messages. The CAN Gateway module is used here again, allowing redirection of touch screen message of interest, when the response on prototype generated screen is expected. Simultaneously, it blocks the transport of touchscreen messages from the display to the Infotainment ECU when the screen generated by the prototype software is active (to avoid false car Infotainment ECU reactions).

4. PLATFORM USAGE POSSIBILITIES

The rapid prototyping platform enables users to create and validate applications that have the potential to transform how drivers interact with their vehicles. These applications can leverage data from vehicle networks through the Data

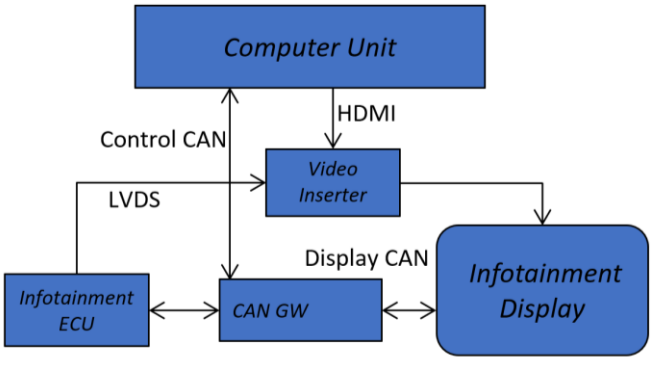

Figure 4. HMI Control Application Principle.

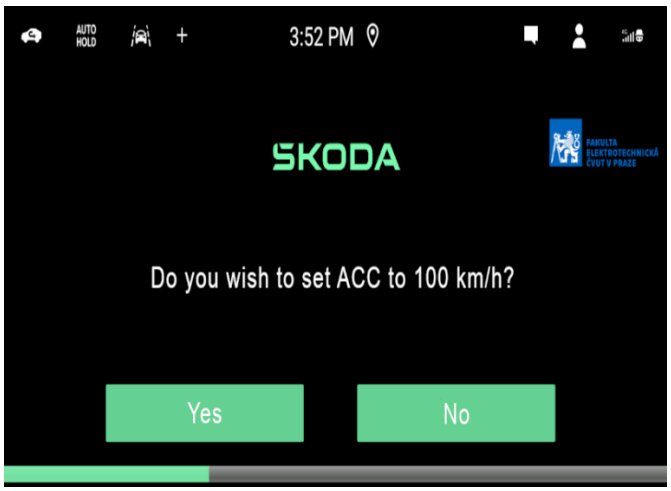

Figure 5. Prediction Based ACC Activation Offer.

Acquisition subsystem, manage specific car functions via the Functionality Controller subsystem, and utilize the internal user interface through the HMI Controller subsystem, all accessible through the vehicle-agnostic car REST API. In order to assess the platform's capabilities and demonstrate its user-friendliness, we have developed and executed two sets of examples. The first set includes traditional (rule-based) applications, while the second set comprises machine learning (ML) based applications. All the applications detailed below can be customized using the car's HMI.

4.1. Classical Application Use Cases

The first example application uses information provided by cars about the actual location and external map sources to get information about the tunnels. Implements automated activation of internal air circulation when the car approaches the tunnel and switches it back when the tunnel is left. This functionality can be configured as fully autonomous or user-confirmed.

The second example application uses information provided by the car about the internal temperature and the occupation of the seats. If the temperature is below the limit, the heated seat is activated. Temperature limits and heating level can be configured individually for particular seats. This application can be easily extended to support automated seat ventilation control if supported by car hardware.

4.2. Machine Learning based Use Cases

The next application again uses the information provided by the car about the actual location. Simultaneously, it acquires information about the ACC (Adaptive Cruise Control) activity and required car speed (in case the ACC is active). During the training phase, the ML model learns how the driver uses ACC. When trained, its predictions are used to (de-)activate the ACC and set the required speed. Another example is focused on car infotainment. On weekday mornings, the driver takes the children regularly to school. They always ask him/her to activate the function of playing songs from his mobile phone via infotainment. An ML-based application receives information on date/time, seat occupation, and infotainment setting. It learns the usage pattern and autonomously activates the audio source for the children's phone when they get on and back to the driver's favorite radio station when they get off. The ML was implemented using the TensorFlow [20] framework and the own learning algorithm.

4.3. DAQ Usage Use Case

In addition to prototyping of new vehicle features, the presented platform can serve as a powerful data acquisition toolbox. Provides an arbitrary number of CAN/CAN FD channels. Each channel is connected as a network interface using SocketCAN. Optimized DBC parsing written in C++ companied by streaming of acquired data to specified TCP endpoint allows data recording with original data rate. Optionally, the recorded signals can be stored in the SQL database for further processing.

5. PLATFORM USAGE RESULTS

The proposed solution was evaluated by implementing the following functionalities. Two representatives use ML, and one is based on a deterministic algorithm. The described use cases were incorporated into Škoda Enyaq Coupé RS iV.

5.1. Learning based ACC Activation assistant

This feature works in such a way that the system learns places where the driver usually activates the ACC. At these locations, the activation of ACC is offered by the HMI display, as shown in Figure 5. Driver behaviour is observed by several signals provided by the DAQ module. There is the state of ACC (active, disabled), the desired distance from the car ahead, driver ID, and the coordinates of the Global Navigation Satellite System (GNSS). From the coordinates, the direction of the vehicle is extracted. This information is used to handle the situation in which a driver wants to activate the ACC in only one direction. In general, the situation is assumed to be different depending on the direction of the road.

The learning is based on the own learning algorithm from the previous ACC activation database. The key point is the recognition of the same place where the ACC was activated. The first condition is the distance between two places less than 50 m, and the variation of the vector of the driving direction less than 30 degrees. In case of no match, the location address is searched by reverse geocoding. It is carried out by the Nominatim engine [21]. This address is searched from the database of previous activations. The match condition is the distance between two places less than 150 m, and the variation of the vector of the driving direction less than 30 degrees. In the case of match, the counter of previous actions is incremented. ACC activation is offered after three previous activations. The information extracted by the nominated engine is checked and extended by the speed limit using Overpass [22]. The ACC automatic activation use case was evaluated in the Prague 6 district. The map with the learned places is shown in Figure 6.

5.2. Seat heating assistant

This use case is inspired by the situation where many passengers after getting into the frozen car turn the seat heating to the maximal level. With interior warming, they start to decrease the heating level until they completely turn it off. The behaviour mentioned can be automated as shown in Figure 7. This specific system has three levels and is normally controlled by the infotainment display. The assistant is enabled by turning the seat heating on. The heating level is decreased by a timeout or by user input. The timeout for particular levels was defined as a linear function descending with interior temperature. The system can adapt to the situation where the user manually decreases seat heating. The seat heating assistant was tested on a car with four individually controlled heated seats.

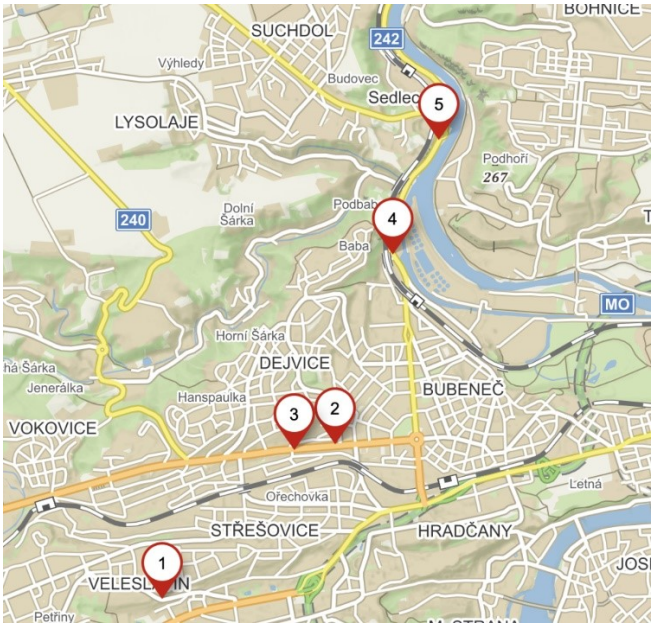

Figure 6. Map with learned places nearby CTU Campus.

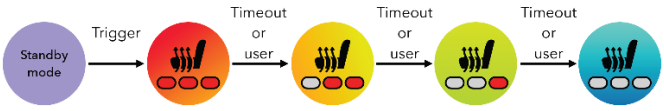

Figure 7. Decrease heating level after timeout.

5.3. Automatic Air Recirculation

Vehicles equipped with air conditioning usually have the function of air recirculation. The purpose is to prevent the penetration of odors or increase the efficiency of cooling. This function is usually controlled by a button. We implemented automation of this feature. The idea is to prevent smell in the interior by closing the air flap before the car enters the polluted area. The flap is opened after the car leaves the defined area.

Automatic air recirculation was prototyped using two strategies. The first is based on points of interest extracted from a map (wastewater treatment plants, industry objects, etc.). POI uses a two-point definition (see Figure 8). One point with an arbitrary radius can be used as an alternative definition. The second strategy is based on learning places where the driver usually activates this function, similar to the ACC use case. TensorFlow framework based on decision trees is used for this task. Figure 9 shows the probability of a closed air flap in a testing area. Red is probability 100 – 71 %, orange 70 – 51 %, yellow 50 – 36 %, and green is below 35 %. The experimental threshold is 60% for closing the flap and 35% for opening it.

6. CONCLUSION

The platform for rapid prototyping of new vehicle features was presented. Addresses the problem of rapid prototyping or evaluation of new software-defined functions, algorithms, and data acquisition on production vehicles. The main capabilities are data acquisition of the information available in vehicle networks, independent control of selected vehicle functionalities, and HMI access. The approach used can be adapted to vehicles of different manufacturers, but quite detailed information is required. The presented platform was incorporated into Škoda Kodiaq and Škoda Enyaq Coupé RS iV. Thanks to shared vehicle platforms, it is pretty straightforward to implement the proposed solution in any VW group car.

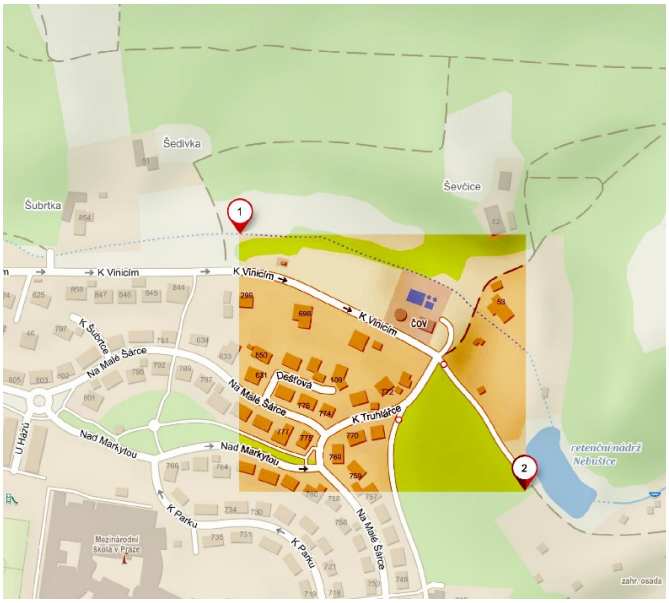

Figure 8. Points of interest solution.

The further development of the rapid prototyping platform is planned to focus on three areas. The first is the usage of SOME/IP services over the Automotive Ethernet within the Functionality Controller. The second is the extension of the platform hardware layer with LIN Gateways. They provide services similar to those of the CAN Gateways described above, but for the LIN networks. Finally, the HMI will be improved to support voice input (instead of the touch screen response, which distracts the driver a lot more).

The use cases described and implemented above were intended to evaluate the proposed platform. Future work will focus on implementing more complex functions using artificial intelligence. For example, the platform is capable to thoroughly monitor driver behaviour using available controls (steering wheel, pedals, button, touches on screens). This can create an interesting surface for AI deployment.

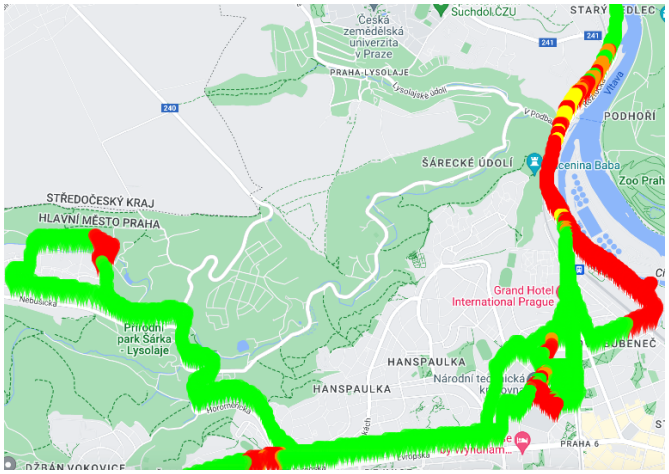

Figure 9. Map with air recirculation flap state prediction.

REFERENCES

[1] M. M. Antony , R. Whenish, Advanced Driver Assistance Systems (ADAS), Cham: Springer International Publishing, 2021, pp. 165–181.
DOI: 10.1007/978-3-030-59897-6_9

[2] A. G. Mariño, F. Fons, J. M. M. Arostegui, The future roadmap of in-vehicle network processing: A hw-centric (r-) evolution, IEEE Access, vol. 10, 2022, pp. 69223–69249.
DOI: 10.1109/ACCESS.2022.3186708

[3] J. Van den Herrewegen, Automotive firmware extraction and analysis techniques, Ph.D. dissertation, University of Birmingham, 2021. Online [Accessed 16 November 2024]
https://etheses.bham.ac.uk/id/eprint/11516/1/VandenHerrewegen2021PhD.pdf

[4] M. Zago, S. Longari, A. Tricarico, M. Carminati, M. G. Pérez, G. M. Pérez, S. Zanero, Recan–dataset for reverse engineering of controller area networks, Data in brief, vol. 29, 2020, p. 105149.
DOI: 10.1016/j.dib.2020.105149

[5] X. Pan, C. Zivkovic, C. Grimm, Virtual prototyping of heterogeneous automotive applications: matlab, systemc, or both?, Proc. of the 24th Asia and South Pacific Design Automation Conf., Tokyo, Japan, 21-24 January 2019, pp. 544–549.
DOI: 10.1145/3287624.3287629

[6] J. Sobotka, J. Novak, J. Pinkava, Rapid prototyping of vehicle software defined functions, 2023, 26th IMEKO TC4 Symp. and 24th Int. Workshop on ADC and DAC Modelling and Testing (IWADC), Pordenone, Italy, 20 - 21 September 2023, p. 88 – 91.
DOI: 10.21014/tc4-2023.20

[7] L. Schärtel, B. Reick, M. Pfeil, R. Stetter, Analysis and synthesis of architectures for automotive battery management systems, Applied Sciences, vol. 12, no. 21, 2022, p. 10756.
DOI: 10.3390/app122110756

[8] K. Belcarz, T. Białek, M. Komorkiewicz, and P. Zolnierczyk, Developing autonomous vehicle research platform – a case study, IOP Conference Series: Materials Science and Engineering, vol. 421, no. 2, 2018, p. 022002.
DOI: 10.1088/1757-899X/421/2/022002

[9] V. Bandur, G. Selim, V. Pantelic, M. Lawford, Making the case for centralized automotive e/e architectures, IEEE Transactions on Vehicular Technology, vol. 70, no. 2, 2021, pp. 1230–1245.
DOI: 10.1109/TVT.2021.3054934

[10] P. Nasahl, N. Timmers, Attacking autosar using software and hardware attacks, 2019 Embedded Security in Cars USA, 2019. Online [Accessed 16 November 2024]
https://core.ac.uk/download/pdf/211131590.pdf

[11] Speeduino website, Speeduino. Online [Accessed 16 November 2024]
https://speeduino.com/home/

[12] Elektrobit, Linux os for automotive, Tech. Rep., 2022. Online [Accessed 16 November 2024]
https://www.elektrobit.com/products/ecu/eb-corbos/linux/

[13] The Linux Foundation, Automotive Grade Linux (AGL), 0106-2024. Online [Accessed 16 November 2024]
https://www.automotivelinux.org/

[14] D. d. S. A. Loura, F. F. V. de Melo Ferreira, R. C. da Costa, Hardware-in-the-loop alternative approach for an ESP verification, SAE Technical Paper, Tech. Rep., 2022.
DOI: 10.4271/2021-36-0066

[15] J. Sobotka, J. Novák, Flexray ecu mission critical parameters measurement, Measurement, vol. 100, 2017, pp. 213–222.
DOI: 10.1016/j.measurement.2016.12.051

[16] S. Yong, Y. Ma, Y. Zhao, L. Qi, Analysis of the influence of can bus structure on communication performance, 5th EAI Int. Conf. IoT as a Service (IoTaaS), Xi'an, China, 16-17 November 2019, Proc. 5. Springer, 2020, pp. 405–416.
DOI: 10.1007/978-3-030-44751-9_34

[17] D. Martin, L. Völker, M. Zitterbart, A flexible framework for future internet design, assessment, and operation, Computer Networks, vol. 55, no. 4, 2011, pp. 910–918.
DOI: 10.1016/j.comnet.2010.12.015

[18] K. Relan, Building REST APIs with Flask: Create Python Web Services with MySQL, 1st ed. USA: Apress, 2019.
DOI: 10.1007/978-1-4842-5022-8

[19] L. Lombardo, Multi-platform solution for data acquisition, Acta IMEKO, vol. 12, no. 1, 2023, pp. 1–8.
DOI: 10.21014/actaimeko.v12i1.1475

[20] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen (+ another 35 author), TensorFlow: Large-scale machine learning on heterogeneous systems, 2015, Software available from tensorflow.org. Online [Accessed 16 November 2024]
https://www.tensorflow.org/static/extras/tensorflow-whitepaper2015.pdf

[21] Nominatim developer community, Nominatim documentation, 2/2024. Online [Accessed 16 November 2024]
https://nominatim.org/release-docs/latest/

[22] R. Olbricht, Overpass API User's Manual, 2/2024. Online [Accessed 16 November 2024]
https://dev.overpass-api.de/overpass-doc/en/