Adaptive Monitoring for Autonomous Vehicles using the HAFLoop Architecture

Edith Zavala[Ɨ*], Xavier Franch[Ɨ], Jordi Marco[±], Christian Berger[Ŧ]

{zavala, franch}@essi.upc.edu, jmarco@cs.upc.edu, christian.berger@gu.se

[Ɨ]*Services and Information Systems Engineering Department, Universitat Politècnica de Catalunya, Barcelona, Spain*

[±]*Computer Science Department, Universitat Politècnica de Catalunya, Barcelona, Spain*

[Ŧ]*Computer Science and Engineering Department, University of Gothenburg, Gothenburg, Sweden*

*Corresponding author

Service and Information System Engineering Department,

Polytechnic University of Catalonia,

Jordi Girona 1-3, 08030, Barcelona, Spain

zavala@essi.upc.edu

Word count: 7,001

Adaptive Monitoring for Autonomous Vehicles using the HAFLoop Architecture

Current Self-Adaptive Systems (SASs) such as Autonomous Vehicles (AVs) are systems able to deal with highly complex contexts. However, due to the use of static feedback loops they are not able to respond to unanticipated situations such as sensor faults. Previously, we have proposed HAFLoop (Highly Adaptive Feedback control Loop), an architecture for adaptive loops in SASs. In this paper, we incorporate HAFLoop into an AV solution which leverages machine learning techniques to determine the best monitoring strategy at runtime. We have evaluated our solution using real vehicles; results are promising and demonstrate the great potential of our proposal.

Keywords: self-adaptive systems; self-improvement capabilities; adaptive monitoring; autonomous vehicles; smart vehicles; machine learning

Introduction

Over the last decades, self-adaptive systems (SASs) have been subject of great research efforts in both industry and academic communities given their applicability in modern domains such as smart cities, mobile apps and autonomous vehicles (AVs) (B. H. C. Cheng et al. 2009; De Lemos et al. 2013; Weyns 2017; Krupitzer et al. 2015). SAS's adaptation is supported by the self-* capabilities: self-configuration, self-optimisation, self-healing and self-protection (IBM-Corporation 2006). Feedback control loops like the IBM's MAPE-K loop (see Fig. 1) (IBM-Corporation 2006; Kephart and Chess 2003) are typically adopted for supporting SASs' self-* capabilities. In MAPE-K, the self-* capabilities are managed by five elements: *Monitor*, *Analyse*, *Plan*, *Execute* and *Knowledge base*. The *Monitor* gathers runtime data from the managed software system and its environment by means of *Sensors*. The collected data is then analysed by the Analyse element and, if needed (e.g. analysis results show a violation of the requirements), a system adaptation scheduled by the *Plan* element. Finally, the adaptation is enacted by the

Execute element using *Effectors*. In order to operate properly, a *Knowledge base* containing data such as measurements, logs, adaptation policies, etc. is shared.

System models, static adaptation rules and objective functions are well-established concepts used by existing MAPE-K loops in order to manage the adaptation process of SASs (Lalanda, McCann, and Diaconescu 2013; Zavala, Franch, and Marco 2019). Adaption rules are typically of the form *if…then*. For instance, in the AV domain, examples of adaptation rules can be: *if* it is raining *then* reduce speed, *if* traffic is slow *then* adjust route to destination. In systems with highly dynamic environments such AVs, the traditional design of static feedback loops can lead to incompleteness or obsolescence of systems' models and adaptation rules, as well as non-optimal objective functions, particularly when runtime uncertainty is experienced (Krupitzer et al. 2017). Runtime uncertainty is one of the main factors challenging modern SASs (Zavala et al. 2018; Knauss et al. 2016; Mallozzi et al. 2019). One of the approaches for addressing this issue is to enable the adaptation of SASs' feedback loops. In this way, for instance, adaptation rules could be updated at runtime in order to better fit context, or system and user changing requirements. Krupitzer et al. (Krupitzer et al. 2016) have defined this process as self-improvement . Concretely, as defined by Krupitzer et al. (Krupitzer et al. 2016):

> *Self-improvement* of the adaptation logic (*the MAPE-K loop elements*) is the adjustment of the adaptation logic to handle former unknown circumstances or changes in the environment or the managed resources (*the managed element*)

[Figure 1 near here]

In a previous work, we have proposed a generic and reusable architecture, called HAFLoop (Highly Adaptive Feedback control Loop) (Zavala et al. 2020), for supporting the self-improvement capability of SASs. In this work, we instantiate such architecture for enabling adaptive monitoring in AVs, i.e., enabling the adaptation of the *Monitor* element of the MAPE-K loop. The *Monitor* is a crucial element since the quality of the monitored data, i.e., accuracy, freshness, completeness, etc., affects directly the performance of the rest of the elements of the loop (i.e., *Analyse*, *Plan*, *Execute* and the *Knowledge base*). For instance, in the AV domain, a de-calibrated of faulty sensor may provoke an erroneous adaptation decision, affecting driver's safety and/or comfortability. Runtime adaptation of the *Monitor* element can also be beneficial when resources are scarce. For instance, in hybrid or electric AVs, a runtime trade-off between the utility and the cost of sensors and monitoring services, may prolong the availability of the main AVs' features, e.g. the self-driving functionality. For the purposes of this work, we use the following generic definition of adaptive monitoring (Zavala, Franch, and Marco 2019):

> *Adaptive monitoring* is the ability a monitoring system has to modify its structure and/or behaviour in order to respond to internal and external stimuli such as changes in their execution context, functional and non-functional requirements, systems under monitoring or the monitoring system itself

In order to support adaptive monitoring in AVs, in this work we propose a solution that combines two HAFLoop feedback loops: (1) a level-1 loop, in charge of the AVs logic, (2) a level-2 loop, in charge of the self-improvement process. Machine learning techniques are integrated into the level-2 loop for adapting the *Monitor* element of the level-1 loop. The proposal is a step forward to address some of the most important challenges affecting current AVs such as sensor faults, energy-efficiency, vehicles' communication and uncertainty (Gruyer et al. 2017; Schwarting, Alonso-Mora, and Rus 2018; Van Brummelen et al. 2018; Zhu et al. 2017). The evaluation of our solution has been performed in the context of the Swedish research project SALI (SmArt seLf-driving vehIcle). A series of scenarios have been evaluated using real vehicles and the scaled city and rural road environments of the AstaZero test track (http://www.astazero.com/) in Sweden.

The remainder of the paper is structured as follows. Section II presents the background of this work. Section III describes our generic architecture HAFLoop and its instantiation in the AV domain for supporting adaptive monitoring. Section IV provides the details of the evaluation of our HAFLoop AVs solution, and discusses the results. Finally, conclusions and future work are presented in Section V.

Background

This work focuses on the domain of AVs and adds self-improvement capabilities in the form of adaptive monitoring for improving AVs' perception systems. Therefore, the background is divided into two parts: 1) perception systems for AVs, and 2) proposals to support self-improvement capabilities.

Autonomous Vehicles Perception

Autonomous vehicles (a.k.a. intelligent vehicles, driverless vehicles, or self-driving vehicles; AVs for short) operate autonomously by perceiving the environment and exhibiting a responsive action such as navigating, vehicle following, overtaking, parking or lane keeping (H. Cheng 2011). The process typically comprises four main stages, as shown in Fig. 2: (1) environment perception and modelling, (2) localization and map building, (3) path planning and decision-making, and (4) motion control (H. Cheng 2011). These stages can be mapped to the functionalities of the MAPE-K loop elements: *Monitor*, *Analyse*, *Plan* and *Execute*, respectively. As mentioned before, in this work we focus on the *Monitor* element adaptation, i.e., the perception stage of AVs. In this stage, data are usually collected by multiple sensors, such as camera, radio detection and ranging (Radar), light detection and ranging (LiDAR), and infrared sensors (Van Brummelen et al. 2018).

[Figure 2 near here]

Since the perception stage is the link between the real world and the rest of the AVs stages, it plays a crucial role (Gruyer et al. 2017; Guanetti, Kim, and Borrelli 2018; Zhu et al. 2017). Given its importance, over the last decade, a lot of research efforts have been dedicated to this stage, developing promising advances in AV technologies (Van Brummelen et al. 2018; Gruyer et al. 2017; Zhu et al. 2017). However, some perception challenges to attain the full potential of AVs remain open, either because they are relatively addressed or largely unaddressed by existing approaches (Van Brummelen et al. 2018). In an extensive overview, recently performed on the AVs' perception topic (Van Brummelen et al. 2018), we have identified a set of challenges that affect the performance of AVs in poor environmental conditions and uncertain and complex settings, as well as challenges regarding the reliability and energy-consumption of sensors and the utilisation of connectivity with other road entities. Other reviews on the AV domain have also remarked the importance of addressing some of these challenges (Gruyer et al. 2017; Zhu et al. 2017; Schwarting, Alonso-Mora, and Rus 2018). From these reviews, we have identified current open perception challenges affecting AVs and classified them in three main categories:

- Category 1. Technological perception challenges
    - **ChlP1.1.** Support AV perception in adverse weather conditions (Gruyer et al. 2017; Schwarting, Alonso-Mora, and Rus 2018; Van Brummelen et al. 2018; Zhu et al. 2017).
    - **ChlP1.2.** Support AV perception in poor lighting conditions (Zhu et al. 2017; Van Brummelen et al. 2018; Gruyer et al. 2017).
- Category 2. Modelling perception challenges

    - **ChlP2.1.** Support AV perception in complex urban environments (Van Brummelen et al. 2018; Schwarting, Alonso-Mora, and Rus 2018; Gruyer et al. 2017).
    - **ChlP2.2.** Reduce reliance on *a priori* data such as maps (Van Brummelen et al. 2018).
- Category 3. Functional perception challenges
    - **ChlP3.1.** Incorporate Vehicle-to-Vehicle (V2V) and Vehicle-to-Infrastructure (V2I) communication in AVs perception systems (Gruyer et al. 2017; Van Brummelen et al. 2018).
    - **ChlP3.2.** React to faulty sensors and uncertain sensor data (Van Brummelen et al. 2018; Gruyer et al. 2017; Schwarting, Alonso-Mora, and Rus 2018; Zhu et al. 2017).
    - **ChlP3.3.** Support energy-efficient AV perception (Van Brummelen et al. 2018).

The *Technological challenges* refer to the aspects constrained by the sensor technology available nowadays. For instance, current camera sensors generate poor-quality images that are difficult to exploit with rainy, foggy, sunny (dazzling or shining), or snow conditions (Gruyer et al. 2017). Similarly, some sensors are affected by the poor lighting conditions. One of the solutions to overcome this issue is the use of data fusion methods, i.e., combine information gathered through different sensors (Zhu et al. 2017; Van Brummelen et al. 2018; Gruyer et al. 2017). Part of the perception stage is the pre-processing of raw sensor data to generate models of the environment. In this regard, there are some *Modelling challenges* that are still open. For example, in cluttered environments AVs may model several interactions among different road users, failing in this task may provoke an unsafe state (Schwarting, Alonso-Mora, and Rus 2018).

Rule-based, probabilistic, learning-based, among other types of approaches have been proposed for modelling complex environments at runtime; however, there are aspects that still have to be addressed (Van Brummelen et al. 2018; Schwarting, Alonso-Mora, and Rus 2018; Gruyer et al. 2017).

The last category corresponds to the *Functional challenges*. These challenges refer to the aspects related to the (*adequate*) operation of the data gathering task of the perception stage, independently from the technology, i.e., sensor data completeness/redundancy, reliability and process optimisation. For example, in high-density traffic conditions, radar systems may pick up other vehicles' radar signals, causing false detections, interference and additional uncertainty (Hischke 2002; Schipper et al. 2015). Moreover, as more sensors are integrated into vehicles, e.g., for supporting data fusion algorithms, the risk of faults increases as well as the power consumption (Ren et al. 2018; Gawron et al. 2018). A popular approach to optimize AVs energy consumption is the adoption of V2V and V2I communications (Gawron et al. 2018; Guanetti, Kim, and Borrelli 2018); however, correctly supporting these capabilities in AVs remains still a challenge (Gruyer et al. 2017; Van Brummelen et al. 2018). In this work, we focus on the operational aspects of AVs' monitoring systems. Therefore, our contribution regarding current AVs challenges will be on ways to address the open *Functional perception challenges*.

Self-Adaptive Systems Self-Improvement

For many years, software solutions for SASs have relied on static feedback loops that manage the whole adaptation process. This implies that changes on the structure and behaviour of the loops' components are not possible at runtime. This vision has constrained SASs capabilities to respond to unpredictable events and in consequence limited their application. In extremely demanding domains such AVs, smart cities or mobile applications, this approach is not suitable;

in these domains, feedback control systems must be able to adapt at runtime as well. Motivated by the increased need of adaptive feedback loops, many researchers have proposed self-improvement solutions to address this issue. In a recent work, we have reviewed 26 approaches for supporting adaptive feedback loops (Zavala et al. 2020), and presented a new generic one, HAFLoop. We have found gaps and opportunities of contribution to the research field; for instance, the urgency of solutions that satisfy modern SASs' requirements such as decentralisation. Based on our findings, we have identified a series of challenges related to the flexibility, generalisability and reusability of the solutions that still need more research efforts to be addressed. Below, we summarize such challenges and organize them into three main categories:

- Category 1. Applicability of self-improvement challenges
  - **ChlS1.1**. Provide generic self-improvement solutions, so they can be reused in a variety of SASs.
  - **ChlS1.2.** Support software engineers in the whole software development lifecycle.
- Category 2. Capabilities of self-improvement challenges
  - **ChlS2.1**. Support reactive (reaction after a change) as well as proactive (action before it) adaptation.
  - **ChlS2.2**. Support changes on both the structure and the parameters of SASs' feedback loops.
- Category 3. Engineering of self-improvement challenges
  - **ChlS3.1**. Support different software components' settings, i.e., centralized, decentralized, hybrid.

The *Applicability of self-improvement challenges* refer to the generalisability and completeness of the proposals. Generic and flexible enough software solutions would structure and accelerate the development process of adaptive loops as well as facilitate the evolution of SASs over time (Zavala et al. 2020). From the analysed approaches, there is a tree-layer based approach, proposed by Perrouin et al. (Perrouin et al. 2012) that tries to address this issue by supporting a variety of adaptation types; however, the proposal provides only a vision while details for engineering such vision are missing. In general, most of the approaches studied do not provide guidelines for designing, implementing, maintaining and evolving their self-improvement solutions in different SASs. On the other hand, the *Capabilities of self-improvement challenges* refer to the functionality supported by the solutions. Concretely, there are two desirable behaviours: first, respond to context changes by adapting SASs feedback loops (reactive adaptation) and predict the need for adaptation before a context change (proactive adaptation); second, respond adequately to context changes either reorganizing feedback loops components (structure adaptation) and/or tuning loop parameters (parameter adaptation).

Most of the current self-improvement approaches do not completely support both behaviours (Zavala et al. 2020). MORPH (Braberman et al. 2017; 2015) is one of the approaches that considers reactive and proactive adaptation as well as structure and parameter adaptation. The proposal consists of a reference architecture for goal-oriented SASs that modifies SASs configuration and loops behaviour in order to satisfy a series of goals, at runtime. The solution relies on a goal model described by the user and enriched by inferences from logs that describes the system state, goals and environment assumptions. The operation of many SASs can be modelled and driven by high-level goals; however, in modern and complex SASs the application of this proposal could not be feasible due to engineering challenges. The *Engineering self-*

improvement challenges affecting modern SASs are mainly provoked by its complexity. Modern SASs applications such as smart cities, mobile apps and smart vehicles require the interoperability of heterogeneous components as well as its decentralisation.

Most of the approaches analysed in our previous work (Zavala et al. 2020) consider only the adaptation of centralized feedback loops and only 3 out of 26 support both centralized and decentralized loops. Unfortunately, the adoption or adaptation of centralized solutions for decentralized settings is not a straightforward task. Therefore, there is a need of flexible solutions that can be adopted in a variety of SASs settings. Motivated by these research gaps, we proposed HAFLoop, an architectural solution that aims at providing generic and reusable software structures for supporting engineers in the systematic development of adaptive feedback loops for modern SASs. In the next section, we summarize our proposal HAFLoop and exemplify how it can be adopted by demanding SASs such the AVs for improving the operation of their feedback loops through adaptation.

Adaptive Monitoring for Autonomous Vehicles

Current AVs' perception systems are constantly affected by runtime uncertainty, e.g., they are exposed to unexpected faults, runtime limited resources such as energy, and the unpredictable behaviour of other vehicles. In order to overcome such situations, a first step will be to address the *Functional perception challenges* affecting AVs (see Section 2.1). The adoption of self-improvement solutions, like HAFLoop, may help AVs to reach this goal. In this section, we focus on how HAFLoop could improve AVs operation. Firstly, we provide an overview of the HAFLoop generic architecture and describe the main characteristics that allow our approach to address current self-improvement challenges (see Section 2.2). Secondly, we propose an architectural solution for AVs using HAFLoop feedback loops and explain the mechanisms that address current *Functional perception challenges*.

The HAFLoop Architecture

HAFLoop (Zavala et al. 2020) is an architectural solution that extends the IBM's MAPE-K loop reference model for enabling the adaptation of the elements of the loop (*Monitor*, *Analyse*, *Plan*, *Execute*, *Knowledge base*), at runtime. HAFLoop proposes generic components and subcomponent that are reused by the different elements, and the mechanisms that allow them to coordinate their normal operation (i.e., monitoring, analysing, planning, executing and managing knowledge) with their own adaptation process. Fig. 3 provides an overview of a HAFLoop MAPE-K element (henceforth, HAFLoopElement). A HAFLoopElement is organized in four layers: (1) *Communication layer*, (2) *Message processing layer*, (3) *Logic layer* and (4) *Knowledge layer*.

The *Communication layer*, as its name indicates, is in charge of the HAFLoopElement communication with external entities; it consists of a *Sender* and a *Receiver* components. The *Message processing layer* is in charge of preparing input and output messages; it is composed of a *Logic selector* and a *Message processor* components. The *Logic layer* corresponds to the actual HAFLoopElement-specific logic, i.e., the logic for monitoring, analysing, planning, executing or managing knowledge, as well as the logic for being adapted. This layer is composed of the *Functional logic* and the *Adaptation logic* components. The *Functional logic* is the main part of the element and it has to be developed for each SAS while the rest of the components can be reused and/or extended (addressing **ChlS1.1**). Finally, the *Knowledge layer* is in charge of managing the HAFLoopElement's knowledge and it is composed of a *Knowledge manager* component.

[Figure 3 near here]

HAFLoop utilizes policies for decoupling HAFLoopElement´s components operation through separation of concerns, i.e., each component has all the information it requires to operate correctly. Therefore, they can be deployed in a variety of SASs settings, from centralized to fully-decentralized (addressing **ChlS3.1**). Policies are utilized at runtime for driving the HAFLoopElement's adaptation process, e.g., in case of an *Analyse* HAFLoopElement, a change on the algorithm used to analyse sensor data is indicated in its policies. Policies can also be used by system owners to configure their adaptive feedback loops, in terms of both behaviour and structure. The adaptation of policies can be triggered proactively and/or reactively that will depend on each SAS implementation, i.e., the architectural solution is decoupled from the specific approaches allowing SASs owners to test, implement and combine different proposals for addressing **ChlS2.1**.

Internally, each HAFLoopElement component operates with three generic subcomponents: (1) a *Message manager*, (2) a *Component policy manager*, and (3) a *Component policy* (see Fig. 4). While the first subcomponent is in charge of receiving, processing and sending component's operation-related messages, the last two are in charge of managing the adaptation process of the component's policy, at runtime. Policy adaptation consists of three main steps: (1) variables adjustment, (2) change notification, (3) component structural and/or parameter adaptation enactment (as response to the policy variables adjustment). Some structural adaptations such as HAFLoopElements and HAFLoopElement components reorganisation (addition, removal, substitution, etc.) are already considered in the proposal (for addressing **ChlS2.2**).

[Figure 4 near here]

In order to address **ChlS1.**2, in the proposal of HAFLoop (Zavala et al. 2020), the engineering process was supported with the development of a framework and a series of guidelines for adopting HAFLoop[1]. The framework for Java-based applications implements the generic functionalities of HAFLoop and it can be reused in any type of SAS. Although the framework only covers Java-based applications, it also serves as an example for implementing HAFLoop in other languages. In this work, we use the Java-based HAFLoop framework for implementing a self-improvement loop and develop an ad-hoc instance of HAFLoop in C++ for supporting AVs smart behaviour.

HAFLoop for Autonomous Vehicles

In order to support adaptive monitoring in AVs, we have adopted a 2-feedback loop-based approach. A level-1 loop is in charge of managing the AV adaptation process (i.e., the smart behaviour) while a level-2 loop is in charge of level-1 loop's *Monitor* element adaptation. The adaptation of level-1 loop *Monitor* is enabled as a step forward to address the open *Functional perception challenges* affecting current AVs. For demonstrating the feasibility of our vision, we have extended the vehicle software environment OpenDLV[2] with the principles of HAFLoop, i.e., we have enable adaptation capabilities to the main AV's feedback control system. Henceforth we call this extended version HAFLoop-OpenDLV.

1. https://github.com/edithzavala/loopa
2. https://github.com/edithzavala/opendlv/tree/feature.smartcar

Concretely, OpenDLV is an open source software stack of microservices for distributed robotic and automotive systems, written in C++, that supports the development of AVs, in real and simulation environments. It supports the whole AVs cycle (see Fig. 2) allowing engineers to test different logics in each phase. It also provides a series of baseline self-driving algorithms. For real vehicles, OpenDLV implements a series of low-level hardware-software interfaces that allow the interaction with vehicles' sensors and actuators. In this work, we will test our solution in a real vehicle with the following sensors: an Applanix POS GPS/INSS, a Velodyne LiDAR HDL32e and an Axis camera. In order to enrich contextual data, apart from these sensors, we will include data gathered through V2V communications (complying with **ChlP3.1**) and weather and traffic monitoring cloud services. HAFLoop-OpenDLV software module becomes the AV level-1 loop mentioned before. The architecture of this loop is shown in Fig. 5. Below, we describe each of its components:

[Figure 5 near here]

- Level-1 Monitor (HAFLoopMonitor)
    - **GPS**[3] is the interface of an Applanix POS GPS/INSS, this component provides vehicle's position data (i.e., latitude and longitude, see Fig. 6a).
    - **LiDAR**[4] is the component that interacts with a VelodyneLiDAR HDL32e, it provides 360º 3D point cloud data (see Fig. 6b). The LiDAR input is used to determine frontal, gear and lateral distances.
    - **Camera**[5] interfaces with an Axis camera in order to provide image data (see Fig. 6c). After receiving the image data, a post-processing that simulates frontal distance calculation is performed.

3. https://github.com/chalmers-revere/opendlv-device-gps-pos
4. https://github.com/chalmers-revere/opendlv-device-LiDAR-hdl32e
5. https://github.com/chalmers-revere/opendlv-device-camera-opencv

  - **V2V** component provides simulated V2V communication data, two types of messages are considered: CAM (ETSI 2011) and DENM (ETSI 2010).
  - **City reporter** component consumes the service offered by HERE (https://developer.here.com/) through API. This service provides real-time traffic data. The City reporter also consumes data from the OpenWeatherMap service (https://openweathermap.org/) in order to gather weather data.

The different subcomponents of the *HAFLoopMonitor,* referred as *Monitors* in Fig. 5*,* are independent one from each other. Therefore, a *Monitor* can be removed, enabled or disabled without interfering in the operation of the rest of subcomponents. In this instance of HAFLoop-OpenDLV, the *HAFLoopMonitor* supports both structural and parameter adaptations (addressing **ChlS2.2**). Regarding the AVs main stages illustrated in Fig. 2, the *HAFLoopMonitor* performs the *Environment perception and modelling* stage which is the most relevant to this work.

[Figure 6 near here]

- Level-1 Analyse, Plan and Execute (HAFLoopAnalyse, HAFLoopPlan and HAFLoopExecute)
  - **Lane follower** is a complex component that appears in three of the loop elements. This component is included for simulation purposes and has been extended from an existing one available at the OpenDaVINCI software vehicles' environment (https://opendavinci.readthedocs.io). The extended version of the lane follower[6] supports a context-aware AV. This component has three main functionalities: 1) follow a lane, 2) overtake objects moving slow or static, 3) re-evaluate route to destination based on traffic and runtime events (e.g., a crash). The

6. https://github.com/edithzavala/OpenDaVINCI/tree/feature.smartvehicle/automotive/miniature/lanefollower

logic of this component encompasses the 3 last stages of an AV process (see Fig. 2).

- Level-1 Knowledge base (HAFLoopKnowledge)
    - **Odsupercomponent** creates an UDP multicast session in order to allow the communication of all the rest of components. Apart from that, it distributes to the other components their initial policies.
    - **Adaptation data manager** forwards to the level-2 loop the monitoring data gathered through sensors and services.
    - **Adaptation enactor manager** receives requests for adaptation from the level-2 loop and sends requests for change to the Monitor. Then, policy variables are adjusted and the Monitor structure and/or parameters adapted.

The level-2 feedback loop, in charge of self-improvement, i.e., adapting the level-1 loop's *Monitor*, has been developed using the Java-based HAFLoop framework. The architecture of the level-2 loop[7] is shown in Fig. 7. Below, we describe each of its components:

[Figure 7 near here]

- Level-2 Monitor (HAFLoopMonitor)
    - **Monitoring data thresholds' checker** revises the correctness of the morning data, based on policies. Data out of thresholds could indicate, for example, a sensor fault (enabling the AV to address **ChlP3.2**, **ChlS2.1**); if that is the case, an alert is triggered. All the data gathered is persisted by the Knowledge base.

7. https://github.com/edithzavala/ksam-loopa

    - **Battery inspector** continuously checks the vehicle's battery level (consider a hybrid AV). If the vehicle starts to run out of battery, an alert is sent to the Analyse element before it gets totally depleted (enabling the AV to proactively address **ChlP3.3**, **ChlS2.1**). The battery level data can be visualized at runtime thanks to the Knowledge base element.
- Level-2 Analyse (HAFLoopAnalyse)
    - **Analysis alerts manager** this component receives alerts from the Monitor element. According the alert type, e.g., a fault or a battery issue, it analyses time series data (stored by the Knowledge base) in order to determines the sensors and monitoring services (from AV's level-1 loop) that will be used in the near future. In order to accomplish this task, it relies on a Data miner component. In case an adaptation is required, it sends an alert to the Plan element.
    - **Data miner** this component uses the data mining Weka tool (Machine Learning Group at the University of Waikato 2016) and existing models (created during a training phase), in order to forecast: (1) the position of vehicles in the near future, (2) the usage of the self-driving functionality such future position(s). Weka offers a variety of classification and clustering algorithms. In this work, we have selected a couple of these algorithms based on previous experiences (Zavala et al. 2015; 2018; Zavala, Franch, and Marco 2019; Zavala et al. 2020), particularly based on the promising results obtained in a series of experiments performed in the smart vehicles domain, in simulation environments.
- Level-2 Plan (HAFLoopPlan)

    - **Planner alerts manager** performs a trade-off between the cost of keeping a monitor (monetary and power), the data it provides, and its relevance regarding the self-driving functionality usage, every time it receives an alert from the Analyse element. Cheaper monitors are preferred if they (or a combination of them) are able to gather the data required. In the case of a battery critical situation, the Planner sacrifices coverage is and a restricted self-driving functionality is supported. When coverage is not sufficient and the risk is high, a request for take-over is sent to the user (disabling the self-driving functionality). The resulting list of monitors to adapt, and their configuration, is sent to the Execute element.
- Level-2 Execute (HAFLoopExecute)
    - **Adaptation request sender** receives the requests for adaptation triggered by the Plan element, transforms them into actionable tasks that can be understood by the Managed Element ME (i.e., in this case, using the communication protocols and format required by the level-1 loop) and sends them.
- Level-2 Knowledge base (HAFLoopKnowledge)
    - **Data store** persist runtime data produced by the MAPE elements. In this implementation, the data gathered by the Monitor element is in .arff files, the format used by Weka. The rest of data, e.g., list of active monitors, active functionalities, data mining prediction, etc., are stored in different runtime variables. This component is also connected to the Runtime data dashboard component for data visualization.

- **Runtime data dashboard** uses the monitoring tool Graphite (http://graphite.readthedocs.io) the Grafana platform (https://grafana.com/) (see Fig. 8) in order to provide a real-time visualization of the monitoring data. Fig. 8 shows the different variables we have visualized, these includes: the calls performed to the different level-2 loop elements, the data mining self-driving functionality usage prediction, the measurements of the monitoring sensors and services as well as their health (i.e., whether they are working properly or not).

[Figure 8 near here]

Evaluation

Motivated by the current challenges affecting AVs perception (see Section 2.1), we have developed a research project called SALI (https://azopenresearch.fluidreview.com/res/p/A0034/). In the context of the SALI project, we have evaluated our solution through a series of experiments conducted in (controlled) real traffic environments, at the facilities of the test ground AstaZero in Sweden. Concretely, experiments have been executed in two test areas: a scaled city (see Fig. 9a) and a rural road (see Fig. 9b).

The components of the AVs level-1 and level-2 loops (see Fig. 5 and Fig. 7) have been implemented as a series of microservices which have been containerized using Docker (https://www.docker.com/) for facilitating, among other tasks, the deployment process. While the level-1 loop was deployed and run on the vehicle's machine during the experiments, the level-2 loop was deployed on an external machine. Concretely, a computer with an IntelR CoreTM i7-7700HQ CPU @ 2.80GHz and 16,0GB of RAM. During the evaluation, both machines were connected through a local area network using an Ethernet connection.

The evaluation aimed at assessing the correctness and timeliness of our solution for supporting adaptive monitoring in modern SASs such the AVs, in realistic environments. In order to do so, a series of use cases have been executed using three real cars: two Volvo XC90s (henceforth Snowfox and Greyfox) and one Volvo V40. Specifically, we have run three use cases: a sensor fault, battery issues and a road accident with uncertain sensor data. Apart from the vehicles mentioned before, a soft vehicle (see left-hand yellow half-vehicle in Fig. 10) has been utilized for the road accident use case at the AstaZero city area. Our solution has been deployed on the Snowfox car, while, the other vehicles have been used for setting up the scenarios.

[Figure 9 near here]

[Figure 10 near here]

Preparation Activities

The evaluation of our solution has consisted of testing our implementation in eight scenarios of our three use cases (two scenarios per use case). Table 1 provides the description of the scenarios while Fig. 11-13 illustrate them. For the sensor fault and battery issues use cases, the AV has to determine whether the self-driving functionality is going to be used or not. In order to enable this feature, a training phase has been conducted for modelling user's preferred route and self-driving functionality usage (see initial set-up vignette in Fig. 12 and Fig. 13). These two use cases have been executed on the rural road while the road accident use case in the city area. The IBk (Aha, Kibler, and Albert 1991) and JRip (Kotsiantis 2007; Tan, Steinbach, and Kumar 2005) algorithms from the Weka tool (Machine Learning Group at the University of Waikato 2016) have been used for training the AV; the first one for position data and the second one for modelling the self-driving functionality usage. The patterns generated by this training phase are illustrated in Fig. 14.

[Figure 11 near here]

[Table 1 near here]

[Figure 12 near here]

[Figure 13 near here]

[Figure 14 near here]

In order to configure the AVs HAFLoop feedback loops, we have set an initial set of policies. Table 2 and Table 3 show a simplified version of the level-1 and level-2 loop policies, respectively. For the sake of simplicity, some policy variables related to elements' internal structure such as the list of components' recipients, are not shown.

[Table 2 near here]

[Table 3 near here]

Scenarios Execution and Results

The experiments have consisted on the repetitive execution of different scenarios. A total of 30 executions have been done. For analysing the evaluation results, we have explored two aspects of the self-improvement process: the response time and the adequacy of the adaptations. The response time has been divided into:

- **Level-2 loop response time**. This is the time elapsed since a challenging factor (from Table 1) is detected by the *Monitor* element, until a decision of *no adaptation required* or an adaptation request is sent.

- **Level-1 loop response time**. This is the time required by the level-1loop for enacting an adaptation since a request is received.
- **Data mining response time**. This refers to the prediction process time, i.e. the time required for predicting next vehicle's positions and the usage of the self-driving functionality in such positions. This time is part of the *Level-2 loop response time*; however, we consider that is interesting to report it separately in our benchmarking.

Regarding the self-improvement adequacy; first, we have evaluated the correctness of the enactment process, i.e., ensure that the expected adaptations, described in Table 1, are the ones enacted; second, if enacted, we check whether it has been enacted timely (from a human-perspective). As a conclusion of the execution of all the scenarios, it can be confirmed that the enactment of all the adaptation decisions have been done correctly and as expected. From a human-perspective, it has also been concluded that the adaptations have been timely enacted. Timelines has been evaluated as follows: for the road accident, the adaptation should have been executed before the intersection is reached (see Fig. 11); for the sensor fault and battery issues use cases, the adaptation is enacted within the segment of the route where the self-driving functionality is required (according to the patterns shown by Fig. 14).

The adequacy of the self-improvement functionality has also been evaluated in terms of prediction correctness. From the results, we have concluded that the prediction of the vehicle's position and the self-driving usage has also been accurate and timely. For instance, when the vehicle was starting the journey our solution predicted that the functionality would be used while when the vehicle was almost at the end of the segment where self-driving is typically used (see Fig. 14), the prediction indicates the functionality would not be required in the near future. Thanks to the correctness of the predictions, the vehicle has made the correct adaptation decisions. Table 4 shows the resulting average response time (in milliseconds) for the data mining module and the level-2 loop. The response time of cases where data mining has been required is reported separately as well as of the scenarios that have configured a minimum number of alert iterations for the *Monitor* and *Analyse* elements. The standard deviation of the response times is provided as well. In Fig. 15, we have plotted a detailed response time per scenario execution. The x-axis indicated the number of the execution, while the y-axis the response times.

The average level-2 loop response times go from 249,83 ms to 781,5 ms, for the cases that have not required data mining. For the rest of the cases, the time goes from 3971,75 ms to 4421,5 ms. Particularly, in the second use case, the data mining module spent in average 3624 ms. On the other hand, the use cases considering waiting iterations have experienced an increment of around 500 ms in their response time; this variable should be further investigated as it may affect the performance of the whole solution. However, we consider that its usage may depend on each application. The benefit of using an architectural solution as HAFLoop is that SASs' owners can put their efforts on investigating domain-specific variables like this one, instead of wasting time building the generic functionalities of an adaptive loop.

Regarding the adaptations' enactment, in $\mathbf{us_2}$ and $\mathbf{us_6}$ the process has implied the adaptation of OpenDLV components (Camera and V2V) and the City reporter (Traffic service). The adaptation in these use case have managed sequentially, i.e., first the City reporter and then the OpenDLV components, resulting in a smaller level-2 loop response time for the City reporter than for OpenDLV components (see Fig, 15). This is an implementation issue that can easily be fixed with parallelism. In this work, we have considered the average of both response times.

Table 5 shows the level-1 loop's average response time (in milliseconds). Response times are shown by component adapted: Camera, V2V or City reporter service. The de/activation of the V2V and the Camera has been simulated by software components; therefore, the response time for the adaptation enactment may be different in real cases. That is not the case of the City reporter which connects to a real service and enacts the adaptation changing its configuration. As in previous table, the standard deviation of the response times has been included. The response time per scenario execution is detailed in Fig. 16. A different graph is provided per monitoring component adapted. The x-axis show the number of execution, while response time is indicated by the y-axis.

[Table 4 near here]

[Table 5 near here]

[Figure 15 near here]

In the case of the City reporter, one can notice that the response time of the second adaptation, in $\mathbf{us_2}$, is much smaller than the response time of the first adaptation (almost 20 times). This systematic difference could be attributed to the service communication protocol which might take more time when the service and the client are connected for the first time; however, we cannot make any conclusions since the low-level details of the framework used for this purpose (i.e., Spring boot - https://spring.io/projects/spring-boot) are hidden from our perspective and out of the scope of this work. The execution of more experiments could improve our understanding of this phenomenon, for example adding a 3rd (or more) adaptation(s).

The results of this evaluation are promising. The execution of the scenarios in a realistic environment not only shows the feasibility but also the benefits of supporting adaptive loops in modern SASs such the AVs. The adoption of HAFLoop has allowed our AV to accurately and timely adapt at runtime and deal with unexpected situations such as faults and limited resources. Among the most important benefits of using HAFLoop in AVs, there is the possibility of supporting self-improvement capabilities and adapt their perception system at runtime. That is, address current *Functional perception challenges* affecting the AV domain. Moreover, since HAFLoop decouples AVs' normal operation (Level-1 loop) from the self-improvement process (Level-2 loop); the adoption of our solution does not affect AVs' self-driving functionality response time.

[Figure 16 near here]

Threats to Validity

- **Internal validity**. In order to reduce this threat, we have interpreted our response time and adaptation adequacy results quantitatively using descriptive statistics in order to determine tendencies and dispersion. An accidental bug in our code can also be considered a threat to internal validity. In order to reduce this unavoidable threat, we have tried to use well-established frameworks and tool such as the previous mentioned Spring boot or Gradle (https://gradle.org/) and Docker, etc.
- **Construct validity**. A threat to construct validity in our evaluation could be that some of the vehicle's dynamics and adaptation enactment processes have been simulated. Due to this decision, some aspects could have not been measured, e.g., the time required by the Camera to physically de/activated. For reducing this threat, we have included in this evaluation a real service (i.e., City reporter) that has been adapted both structurally and parametrically. This adaptation case demonstrates a more realistic runtime adaptation process.

- **External validity**. The external validity threat is about how generalizable are our conclusions. The evaluation presented in this this work has been conducted in the domain of AVs and the results show the feasibility and benefits of adopting HAFLoop in this extremely demanding domain. However, the scenarios for running this evaluation are simple compared to the highly complex situations a real AV has to face in real life. Given such evaluation characteristic, the generalisation of our results ignoring other factors is limited, not only to the domain, but also to the application of our solution in this specific domain. More experimentation could reduce this threat; however, due the great diversity of execution contexts, in the SASs' field it will always exist.

Conclusion and Future Work

This work has addressed the runtime adaptation of SASs' feedback control loops, particularly the adaptation of the Monitor element for responding to changes in the environment and the system itself. In order to support the adaptation process, we have adopted HAFLoop, a generic architectural solution for supporting SASs' self-improvement. We have identified open research challenges affecting AVs perception systems as well as challenges affecting the research field of SASs' self-improvement. Although great efforts have been done in both fields, there are not state-of-the-art solutions that address all the functional challenges affecting them. Motivated by this fact, we have implemented a solution for a real vehicle. The solution leverages machine learning techniques to determine the best adaptation decision based on user's behaviour and context data. We have validated our proposal in a series of use cases scenarios and in real (controlled) environments at the AstaZero test track.

The evaluation results are promising and demonstrate not only the feasibility but the benefits of adopting our proposal, in the domain of AVs. A next step for this work would be the development of solutions for adapting the rest of the MAPE-K elements. Each element operates in a different way and has a different objective; therefore, we consider that element-specific approaches should be developed. For instance, the frequency parameter that has been adapted in this work in the *Monitor* element may not be relevant for the *Plan* element. Finally, regarding the evaluation of our generic proposal HAFLoop, we consider that experiments with different settings should still be investigated. For instance, in other domains or using other techniques in the domain of AVs, e.g., other data mining algorithms.

Acknowledgment

Thanks to the Chalmers Revere vehicle laboratory for the technical support provided during the implementation and experimentation phases described in this work. This work was supported by the Swedish organisations AstaZero and SAFER under Grant Project A-0034; Campus Energia (UPC) under program Ajuts de mobilitat internacional; the Spanish project GENESIS under Grant TIN2016-79269-R; the Mexican council CONACYT.

References

Aha, David W., Dennis Kibler, and Marc K. Albert. 1991. “Instance-Based Learning Algorithms.” *Machine Learning* 6 (1): 37–66. https://doi.org/10.1023/A:1022689900470.

Braberman, Victor, Nicolas D’Ippolito, Jeff Kramer, Daniel Sykes, and Sebastian Uchitel. 2015. *MORPH: A Reference Architecture for Configuration and Behaviour Self-Adaptation*. *Proceedings of the 1st International Workshop on Control Theory for Software Engineering - CTSE 2015*. Vol. 1. https://doi.org/10.1145/2804337.2804339.

Braberman, Victor, Nicolas D’Ippolito, Jeff Kramer, Daniel Sykes, and Sebastian Uchitel. 2017. “An Extended Description of MORPH: A Reference Architecture for Configuration and Behaviour Self-Adaptation.” *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 9640 LNCS: 377–408. https://doi.org/10.1007/978-3-319-74183-3_13.

Brummelen, Jessica Van, Marie O’Brien, Dominique Gruyer, and Homayoun Najjaran. 2018. “Autonomous Vehicle Perception: The Technology of Today and Tomorrow.” *Transportation Research Part C: Emerging Technologies*. Elsevier. https://doi.org/10.1016/j.trc.2018.02.012.

Cheng, B H C, R De Lemos, H Giese, P Inverardi, and J Magee. 2009. “No Title.” *Software Engineering for Self-Adaptive Systems*, 1–26.

Cheng, Hong. 2011. *Autonomous Intelligent Vehicles : Theory, Algorithms, and Implementation*. Springer.

ETSI. 2010. “ETSI EN 302 637-3 Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 3: Specifications of Decentralized Environmental Notification Basic Service.” *Etsi* 1: 1–73. http://portal.etsi.org/chaircor/ETSI_support.asp.

———. 2011. “Intelligent Transport Systems (ITS) - Vehicular Communications - Basic Set of Applications - Part 2 : Specification of Cooperative Awareness Basic Service.” *History* 1: 1–22. https://www.etsi.org/deliver/etsi_en/302600_302699/30263702/01.03.02_60/en_30263702v010302p.pdf.

Gawron, James H., Gregory A. Keoleian, Robert D. De Kleine, Timothy J. Wallington, and Hyung Chul Kim. 2018. “Life Cycle Assessment of Connected and Automated Vehicles: Sensing and Computing Subsystem and Vehicle Level Effects.” *Environmental Science and Technology* 52 (5): 3249–56. https://doi.org/10.1021/acs.est.7b04576.

Gruyer, Dominique, Valentin Magnier, Karima Hamdi, Laurène Claussmann, Olivier Orfila, and Andry Rakotonirainy. 2017. “Perception, Information Processing and Modeling: Critical Stages for Autonomous Driving Applications.” *Annual Reviews in Control* 44: 323–41. https://doi.org/10.1016/j.arcontrol.2017.09.012.

Guanetti, Jacopo, Yeojun Kim, and Francesco Borrelli. 2018. “Control of Connected and Automated Vehicles: State of the Art and Future Challenges.” *Annual Reviews in Control* 45 (May): 18–40. https://doi.org/10.1016/j.arcontrol.2018.04.011.

Hischke, M. 2002. “Collision Warning Radar Interference,” 13–18. https://doi.org/10.1109/ivs.1995.528250.

IBM-Corporation. 2006. “An Architectural Blueprint for Autonomic Computing.” *IBM White Paper* 36 (June): 34. https://doi.org/10.1021/am900608j.

Kephart, Jeffrey O, and David M Chess. 2003. “The Vision of Autonomic Computing.” *IEEE Computer Society* 36 (1): 41–50.

Knauss, Alessia, Daniela Damian, Xavier Franch, Angela Rook, Hausi A. Múller, and Alex Thomo. 2016. “Acon: A Learning-Based Approach to Deal with Uncertainty in Contextual Requirements at Runtime.” *Information and Software Technology* 70: 85–99. https://doi.org/10.1016/j.infsof.2015.10.001.

Kotsiantis, Sotiris B. 2007. “Supervised Machine Learning: A Review of Classification Techniques.” *Informatica* 31: 249–68. https://doi.org/10.1115/1.1559160.

Krupitzer, Christian, Julian Otto, Felix Maximilian Roth, Alexander Frommgen, and Christian Becker. 2017. “Adding Self-Improvement to an Autonomic Traffic Management System.” In *Proceedings - 2017 IEEE International Conference on Autonomic Computing, ICAC 2017*, 209–14. IEEE. https://doi.org/10.1109/ICAC.2017.16.

Krupitzer, Christian, Felix Maximilian Roth, Martin Pfannemuller, and Christian Becker. 2016. “Comparison of Approaches for Self-Improvement in Self-Adaptive Systems.” In

Proceedings - 2016 IEEE International Conference on Autonomic Computing, ICAC 2016, 308–14. IEEE. https://doi.org/10.1109/ICAC.2016.18.

Krupitzer, Christian, Felix Maximilian Roth, Sebastian Vansyckel, Gregor Schiele, and Christian Becker. 2015. “A Survey on Engineering Approaches for Self-Adaptive Systems.” *Pervasive and Mobile Computing* 17 (PB): 184–206. https://doi.org/10.1016/j.pmcj.2014.09.009.

Lalanda, P, J A McCann, and A Diaconescu. 2013. “No Title.” *Autonomic Computing: Principles, Design and Implementation*.

Lemos, R De, H Giese, H A Müller, and M Shaw. 2013. “Software Engineering for Self-Adaptive Systems: A Second Research Roadmap.” *Software Engineering for Self-Adaptive Systems II* 7475.

Machine Learning Group at the University of Waikato. 2016. “Weka 3: Data Mining Software in Java.” 2016. http://www.cs.waikato.ac.nz/ml/weka/.

Mallozzi, Piergiuseppe, Patrizio Pelliccione, Alessia Knauss, Christian Berger, and Nassar Mohammadiha. 2019. “Autonomous Vehicles: State of the Art, Future Trends, and Challenges.” In *Automotive Systems and Software Engineering: State of the Art and Future Trends*, edited by Yanja Dajsuren and Mark van den Brand, 347–67. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-12157-0_16.

Perrouin, Gilles, Brice Morin, Franck Chauvel, Franck Fleurey, Jacques Klein, Yves Le Traon, Olivier Barais, and Jean Marc Jezequel. 2012. “Towards Flexible Evolution of Dynamically Adaptive Systems.” *Proceedings - International Conference on Software Engineering*, 1353–56. https://doi.org/10.1109/ICSE.2012.6227081.

Ren, Jing, Moustafa El Gindy, A.N. Ouda, Haoxiang Lang, and Amr Mohamed. 2018. “Literature Survey for Autonomous Vehicles: Sensor Fusion, Computer Vision, System Identification and Fault Tolerance.” *International Journal of Automation and Control* 12 (4): 555. https://doi.org/10.1504/ijaac.2018.095104.

Schipper, Tom, Silvia Prophet, Marlene Harter, Lukasz Zwirello, and Thomas Zwick. 2015. “Simulative Prediction of the Interference Potential Between Radars in Common Road Scenarios.” *IEEE Transactions on Electromagnetic Compatibility* 57 (3): 322–28. https://doi.org/10.1109/TEMC.2014.2384996.

Schwarting, Wilko, Javier Alonso-Mora, and Daniela Rus. 2018. “Planning and Decision-Making for Autonomous Vehicles.” *Annual Review of Control, Robotics, and Autonomous Systems* 1 (1): annurev-control-060117-105157. https://doi.org/10.1146/annurev-control-060117-105157.

Tan, Pang-Ning, Michael Steinbach, and Vipin Kumar. 2005. *Introduction to Data Mining,*. Pearson Publishing.

Weyns, Danny. 2017. “Software Engineering of Self-Adaptive Systems: An Organised Tour and Future Challenges.” *Handbook of Software Engineering*, 1–41.

Zavala, Edith, Xavier Franch, and Jordi Marco. 2019. “Adaptive Monitoring: A Systematic Mapping.” *Information and Software Technology* 105 (January): 161–89.

https://doi.org/10.1016/j.infsof.2018.08.013.

Zavala, Edith, Xavier Franch, Jordi Marco, and Christian Berger. 2020. "HAFLoop: An Architecture for Supporting Highly Adaptive Feedback Loops in Self-Adaptive Systems." *Future Generation Computer Systems* 105 (April): 607–30. https://doi.org/10.1016/j.future.2019.12.026.

Zavala, Edith, Xavier Franch, Jordi Marco, Alessia Knauss, and Daniela Damian. 2015. "SACRE: A Tool for Dealing with Uncertainty in Contextual Requirements at Runtime." In *23rd IEEE International Requirements Engineering Conference (RE)*, 278–79. IEEE. https://doi.org/10.1109/RE.2015.7320437.

———. 2018. "SACRE: Supporting Contextual Requirements' Adaptation in Modern Self-Adaptive Systems in the Presence of Uncertainty at Runtime." *Expert Systems with Applications* 98 (May): 166–88. https://doi.org/10.1016/j.eswa.2018.01.009.

Zhu, Hao, Ka Veng Yuen, Lyudmila Mihaylova, Henry Leung, Reuse Unless, Patents Act, White Rose, Takedown If, and White Rose. 2017. "Overview of Environment Perception for Intelligent Vehicles." *IEEE Transactions on Intelligent Transportation Systems* 18 (10): 2584–2601. https://doi.org/10.1109/TITS.2017.2658662.