






# Analyzing Cyber-Physical Systems in Cars: A Case Study

Harry H. Beyel<sup>1</sup>, Omar Makke<sup>2</sup>, Fangbo Yuan<sup>2</sup>, Oleg Gusikhin<sup>2</sup>  
and Wil M.P. van der Aalst<sup>1</sup>

<sup>1</sup>Chair of Process and Data Science, RWTH Aachen University, Aachen, Germany

<sup>2</sup>Global Data Insight & Analytics, Ford, Dearborn, U.S.A.


**Keywords:** Connected Car, Continuous Data, Sensor Data, Process Mining, Conformance Checking, Case Study.


**Abstract:** Cyber-physical systems connected to the internet are generating unprecedented volumes of data. Understanding cyber-physical systems' behavior using collected data is becoming increasingly important. Process-mining techniques consider sequences of events and thus can be used to check and verify how such cyber-physical systems operate. The data captured by cyber-physical systems are typically noisy and are not readily suitable for process mining. In this work, we present how a stream of connected-vehicle data can be transformed into an event log suitable for process mining. By applying different process-discovery techniques, we discover de-facto models that capture the behavior of an assistance system embedded in cars. We apply conformance-checking techniques and consult domain experts to find the best de-facto model. In addition, we apply conformance-checking methods to a preexisting, de-jure model that we transformed into a Petri net. We compare both models and point out differences. In this process, we show how we overcome challenges and highlight why applying process-mining techniques in the cyber-physical systems domain is valuable.


## 1 INTRODUCTION


Cyber-physical systems connect physical processes with computations. Computers monitor and control processes and react to changes, usually in real-time. Designing cyber-physical systems is challenging (Lee, 2008). The design process has become increasingly complex requiring multiple human experts from different fields to collaborate on designing today's systems. Such complex cyber-physical systems are, for example, embedded in cars. These cars are characterized by their continuous connectivity to support their many connected features, such as those related to autonomous driving, servicing, and energy management systems in electric vehicles. As a result, an abundance of data will be collected from these features. Analyzing these features and their underlying systems by using the connected-vehicle data plays a crucial role in improving them. An example of such a system is the hands-free driving assist system de-


veloped by Ford, embedded in connected vehicles. The system allows the vehicle to be driven hands-free and without any human intervention in predefined zones on a map as long as certain conditions are not violated. Vehicles equipped with this feature require connectivity and can receive over-the-air updates as often as needed to improve the feature's behavior based on its usage. Therefore, it is crucial to develop techniques that explain the feature's behavior as dictated by the data and to understand the sequence of events that led the feature controls to make a specific decision. Connected vehicles play a vital role in acquiring the usage data for such a feature from real-world scenarios. In this paper, we used test vehicles to collect the vehicle data. A data stream is recorded at a one-second rate and transmitted to the cloud using the vehicle's modem. Features of the collected data include an anonymized vehicle identifier, states and warnings of assistance systems, and a vehicle's speed. Besides checking if the system works as intended, there is an additional need to understand how drivers use the hands-free driving feature. A possible option to check the behavior is using process-mining techniques. However, the received connected-car data are not suitable for process mining since process-mining techniques require an event

<sup>a</sup> <https://orcid.org/0000-0002-6541-3848>

<sup>b</sup> <https://orcid.org/0000-0002-7295-751X>

<sup>c</sup> <https://orcid.org/0009-0008-9068-435X>

<sup>d</sup> <https://orcid.org/0000-0001-6943-4227>

<sup>e</sup> <https://orcid.org/0000-0002-0955-6940>

log as input. An event log consists of events that have three mandatory attributes: *case*, *activity*, and *timestamp*. Cases can be thought of as runs of a system, activities as actions taking place in a system, and timestamps mark when an event happened. In contrast to event logs, vehicle data does not contain these attributes initially; therefore, the data must be transformed. Process mining techniques can be grouped into process discovery, conformance checking, and process enhancement (van der Aalst, 2016). Process-discovery techniques reveal a comprehensible process model representing the event log's underlying process. Conformance-checking techniques quantify how well a process model represents the behavior contained in an event log. Process-enhancement techniques show additional insights into a process, for instance, the reasoning for decisions.

Process mining is typically applied in organizations to improve, e.g., financial or production processes. However, these techniques can also be applied in other areas, for example, smart home environments or in the software domain (van Eck et al., 2016b; Rubin et al., 2014; Astromskis et al., 2015; Maruster et al., 2008). Applying process-mining techniques in a car-related setting is interesting but challenging for numerous reasons. First, data collection is susceptible to *data quality* issues and noise. At any instant in time, thousands of vehicles send data under different conditions, such as software versions, sensors, ages, and connectivity conditions. An example of inaccurate information is GPS signals contradicting vehicle speed or temperature measurements taken while a vehicle is parked under the sun. Second, *privacy* and *security* requirements must be met. The collected data may contain information that could be used to identify individuals directly or indirectly, and in some cases, anonymization is performed on the vehicle by altering the signals to ensure privacy is maintained. Third, the *contextualization* of data is not trivial. The recorded data may not be well interpretable without additional context. For instance, when the speed of a car is low, potential reasons can be road conditions or a traffic jam. Without this contextualization, it is hard to understand the behavior of drivers.

In this work, we accomplish multiple goals. First, we transform connected-vehicle data to better suit the process mining requirements. Transforming connected-vehicle data into an event log requires several steps. An overview of the steps is depicted in Figure 1. In the remainder of this paper, we specify how we transformed connected-vehicle data into an event log. Second, we transform a flowchart-like behavioral model provided by domain experts into a formal process model represented as a Petri net. Dur-

ing the equivalent behavioral transformation from the provided model to a Petri net, we consult domain experts to ensure that all aspects are captured. Third, we apply conformance checking on this de-jure model using the transformed event log. Fourth, we discover a model from the transformed event log that explains the behavior well. In this process, we discover multiple process models. To pick the best de-facto model, we have to consider quality metrics and models' understandability for domain experts. Fifth, we explain the differences between the two models. By explaining the differences, we consult domain experts and pay attention to the data and algorithms we use.

In the remainder of this paper, we first introduce basic concepts in Section 2. We show and discuss related work in Section 3. In Section 4, we provide insights into how we transformed vehicles' data into an event log. In Section 5, we demonstrate how we create the de-jure model from a preexisting description. Moreover, we show how we pick the best de-facto model discovered by process-mining techniques. In Section 6, we point out differences between models and discuss them. We summarize and discuss our work in Section 7 and describe future work.

## 2 PRELIMINARIES

In this section, we formally introduce concepts that are the basics of the techniques we introduce later. Given a set  $X$ , a sequence  $\sigma \in X^*$  assigns an enumeration to elements of the set, i.e.,  $\sigma : \{1, \dots, n\} \rightarrow X$ . We denote this with  $\sigma = \langle \sigma_1, \dots, \sigma_n \rangle$ . In the remainder, we refer with  $\sigma_i$  to the sequence's  $i$ -th element.

We receive data recorded by vehicles. The data of vehicles are recorded on US highways at a sample rate of one second. The data contains information related to cars' software and sensors. In the following, we briefly explain the recorded data's most important features.

- *Vehicle*: Vehicle to which a datum belongs.
- *Journey*: A journey is created as soon as a recording of a vehicle takes place. Moreover, journeys are generated if highways are switched. Also, they are generated if a vehicle drives in a rest area or at a petrol station. This means that traveling from place A to B can involve multiple journeys. A journey is linked to a vehicle.
- *Run*: A run marks the beginning of hands-free driving and its turn-off. Multiple runs can be contained in one journey.
- *Speed*: Vehicle speed in km/h.

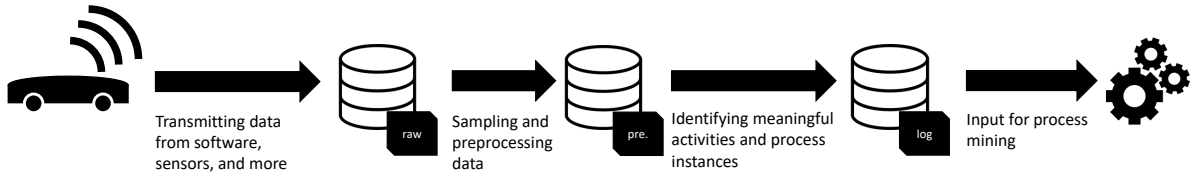


Figure 1: Overview of transforming connected-vehicle data to an event log.

- *State*: States of traffic jam assist. We consider four states: hands-free driving and states one, two, and three.
- *Warning*: Different warnings are stored in this feature. Ranging from no warning to warnings one, two, and three.
- *Timestamp*: Local timestamp of vehicle for each recording.

An example of such data is shown in Table 1. In the following, we formally introduce vehicle data.  $\mathcal{U}_{time}$  is the universe of timestamps,  $\mathcal{U}_{vehicle}$  is the universe of vehicles, and  $\mathcal{U}_{journey}$  is the universe of journeys.  $P$  is the set of states, and  $W$  is the set of warnings of the assistance system.  $\mathbb{R}_{\geq 0}$  is the set of real numbers greater or equal to zero.

**Definition 1** (Vehicle Data).  $\mathcal{U}_{vd}$  is the universe of vehicle data instances,  $\mathcal{U}_{v-att}$  is the universe of attribute names of vehicle data ( $\{vehicle, journey, run, speed, state, warning, time\} \subseteq \mathcal{U}_{v-att}$ ),  $\mathcal{U}_{v-val}$  is the universe of vehicle attribute values, and  $\mathcal{U}_{v-map} = \mathcal{U}_{v-att} \rightarrow \mathcal{U}_{v-val}$  is the universe of vehicle attribute value mappings. Vehicle data is a tuple  $V = (D, \pi_D)$  with  $D \subseteq \mathcal{U}_{vd}$  as the set of vehicle data instances,  $\pi_D \in D \rightarrow \mathcal{U}_{map}$ , such that,  $\forall d \in D$ :  $\{vehicle, journey, run, speed, state, warning, time\} \subseteq \text{dom}(\pi_D(d))$  and the following holds:

- *vehicle*:  $\pi_D(d)(vehicle) \in \mathcal{U}_{vehicle}$  is the vehicle of  $d$
- *journey*:  $\pi_D(d)(journey) \in \mathcal{U}_{journey}$  is the journey of  $d$
- *run*:  $\pi_D(d)(run) \in \mathcal{U}_{run}$  is the run of  $d$
- *speed*:  $\pi_D(d)(speed) \in \mathbb{R}_{\geq 0}$  is the speed of  $d$
- *state*:  $\pi_D(d)(state) \in P$  is the state of  $d$
- *warning*:  $\pi_D(d)(warning) \in W$  is the warning of  $d$
- *time*:  $\pi_D(d)(time) \in \mathcal{U}_{time}$  is the timestamp of  $d$

We assume that in journeys, vehicle data instances can be sorted by their timestamp from earliest to latest. This attribute and its nature are crucial for the later transformation. By referring to *RowID* as a vehicle-data-instance identifier, the example data set in Table 1 has the following journeys:  $\langle 1, 2, 3, 4, 5, 6, 7, 8 \rangle, \langle 9, 10, 11, 12 \rangle$ . The runs given these data are  $\langle 1, 2, 3, 4 \rangle, \langle 5, 6, 7 \rangle, \langle 9, 10, 11, 12 \rangle$ .

To later apply process-mining techniques, we need an event log. An event log has three main attributes: case identifiers, activity names, and timestamps.  $\mathcal{U}_{case}$  is the universe of case identifiers,  $\mathcal{U}_{act}$  is the universe of activity names. The following defines an event log.

**Definition 2** (Event Log).  $\mathcal{U}_{ev}$  is the universe of events,  $\mathcal{U}_{e-att}$  is the universe of event attribute names ( $\{case, act, time\} \subseteq \mathcal{U}_{e-att}$ ),  $\mathcal{U}_{e-val}$  is the universe of event attribute values, and  $\mathcal{U}_{e-map} = \mathcal{U}_{e-att} \rightarrow \mathcal{U}_{e-val}$  is the universe of event attribute value mappings. An event log is a tuple  $L = (E, \pi_E)$  with  $E \subseteq \mathcal{U}_{ev}$  as the set of events,  $\pi_E \in E \rightarrow \mathcal{U}_{e-map}$ , such that,  $\forall e \in E$ :  $\{case, act, time\} \subseteq \text{dom}(\pi_E(e))$  and  $\pi_E(e)(act) \in \mathcal{U}_{act}$  is the activity of  $e$ ,  $\pi_E(e)(time) \in \mathcal{U}_{time}$  is the timestamp of  $e$ , and  $\pi_E(e)(case) \in \mathcal{U}_{case}$  is the case of  $e$ .

Since we use conformance-checking techniques, we point out the following works to get familiar with the used techniques (Adriansyah et al., 2011; Adriansyah, 2014; Verbeek et al., 2001). Moreover, we use Petri nets to represent processes. The work presented in (van der Aalst, 2016; Reisig, 1985) provides an introduction.

### 3 RELATED WORK

In this section, we show and comment on related work. First, we focus on process mining in the Internet-of-Things (IoT), respectively, cyber-physical system domain. Second, we mention some case studies that created de-jure models and applied conformance-checking techniques to these.

Applying process mining to data gathered by cyber-physical systems, respectively, IoT devices, is not new. In (Janiesch et al., 2020), challenges and opportunities of combining IoT data with business process management are described. The first sketched challenge is to bridge the gap between low-level sensor data and event logs. The second named challenge is to correlate activities with process instances. Also, the work stresses designing trouble-free process models, for instance, by avoiding deadlocks and livelocks. In addition, the authors propose to use online confor-

Table 1: Example recording of vehicle data.

RowID	Journey	State	Warning	Timestamp
1	1	Hands-free	No warning	2023-01-01 13:37:37
2	1	Hands-free	No warning	2023-01-01 13:37:38
3	1	Hands-free	Warning 1	2023-01-01 13:37:39
4	1	State 1	Warning 1	2023-01-01 13:37:40
5	1	Hands-free	No warning	2023-01-01 13:37:41
6	1	Hands-free	No warning	2023-01-01 13:37:42
7	1	State 1	No warning	2023-01-01 13:37:43
8	1	State 1	No warning	2023-01-01 13:37:44
9	2	Hands-free	No warning	2023-01-02 20:08:02
10	2	Hands-free	No warning	2023-01-02 20:08:03
11	2	Hands-free	No warning	2023-01-02 20:08:04
12	2	State 2	No warning	2023-01-02 20:08:05

mance checking to check compliance rules during execution. An example of tackling those challenges is provided in (van Eck et al., 2016b). The authors segment sensor data in smaller time windows and compute relevant features for each segment, for example, the average value of an attribute. Based on these features, the created segments are clustered. In the next step, the clustered segments are labeled using domain knowledge, and activities are created. In our work, we do not rely on strict time windows. Nevertheless, we use sensor recordings to create additional information for each event, for instance, if it was recorded during a traffic jam. In (Koschmider et al., 2020), a general framework for discovering process models from sensor data is presented. This framework considers a sensor event log as input and proposes three steps to transform these data into an event log. First, correlating sensor events to an activity instance. Second, discovering process activities and their labels and sensor-level process models. Third, abstracting the data to represent the processes. We use a similar approach as shown in Figure 1. In (Makke and Gusikhin, 2021), parking space occupancy is monitored. To track the occupancy of parking space, a Petri net is discovered by applying process-mining techniques. Their model can be changed if the environment changes, for instance, when sensors fail.

In this work, we apply conformance-checking techniques to check how well de-jure and de-facto models represent reality. In other domains, for instance, the healthcare domain, a de-jure model is created from clinical guidelines and evaluated using conformance checking. Examples are (Xu et al., 2020; Grüger et al., 2021; Benevento et al., 2023).

## 4 GENERATING EVENT DATA

In this section, we present in greater detail how we transformed vehicles' data into an event log. An overview is shown in Figure 2. We sample the data and filter out noisy instances as depicted in Figure 2. Moreover, we enhance the data by creating new features and merging journeys. After identifying activities and process instances, we receive an event log that can be used as input for process-mining techniques. The structure of this section is as follows. First, we reveal how we sample, filter, merge, and enhance data. Second, we explain how we transformed the data into an event log.

### 4.1 Sampling, Filtering, Merging, and Enhancing Vehicles' Data

As shown in Figure 2, the first step is sampling the data. This effort has to consider erroneous data (negative odometer) and noisy data (temperature appears warmer than it is). Since the size of the collected data is large, it is difficult for most process-mining tools to process data at such a large scale. Journeys are randomly sampled from the collected data, and only journeys with useful information are considered to be part of the sampling pool, thus, minimizing the amount of erroneous data. The sampled data contains over 14 million instances, over 25,000 journeys, and records from 100 vehicles. The second step is filtering the data. Due to a large amount of data available, we found the results more revealing if we only keep journeys with hands-free engagements. Also, through several iterations, we identified some erroneous data, such as overlapping journeys. As pointed out previously, the data is sampled each second. As a result, a data collection event that takes 500 seconds has to

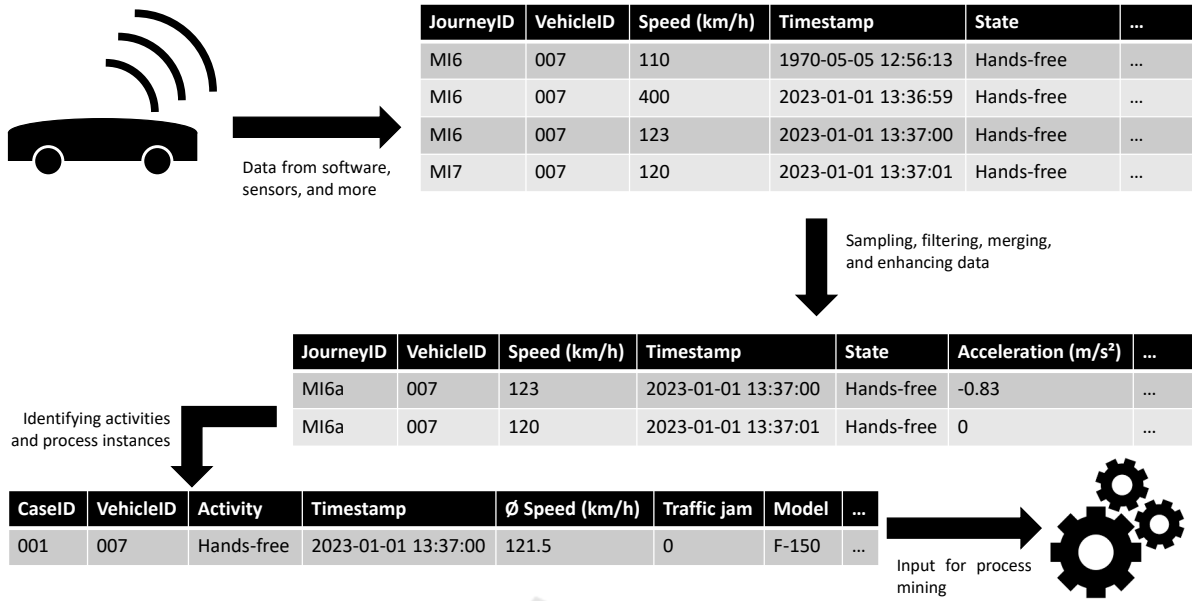


Figure 2: More-detailed overview of transforming connected-vehicle data to an event log.

have 501 entries in the data. Since this assumption does not always hold, we filter out journeys for which this assumption is violated. Therefore, we created a script that checks the data for flaws based on domain knowledge.

The third step is merging data. As introduced earlier, many triggers exist for creating a new journey. To link hands-free driving sessions which are temporally close to each other, we merge vehicle journeys of the same car if they are less than five minutes apart.

The fourth step is data enrichment which can be considered a feature-extraction phase. Consider acceleration as an example. We derive the acceleration from the recorded speed curve. For the last entry of a journey, we denote an acceleration of 0. A more complex data enhancement is performed when detecting if a vehicle is in a traffic jam situation. To detect traffic jams, we rely on two fundamental attributes: speed and acceleration. Our idea is based on the assumption that unusual acceleration and deceleration in combination with low speeds on highways in a short time indicate traffic jams. Also, we introduce new states and warnings which are beneficial for the later transformation phase.

After preprocessing, the data contains roughly 2.6 million instances, over 3,100 journeys, and records from 80 vehicles.

## 4.2 Transforming Vehicles' Data

Using the preprocessed data, activities, and process instances need to be identified. Since we aim to an-

alyze the hands-free driving system, activities have to be related to this system. In this work, we rely on domain knowledge and warnings and states of the assist system. Using the previously mentioned features, it is possible to create activities showing how the hands-free-driving feature works and how drivers interact with it. Different scopes for process instances are possible, for example, whole journeys. However, we decide to define process instances from starting the hands-free driving system to turning it off — we focus on runs. In the following, we present the formal framework for our transformation. First, we introduce maximal sequences. Second, we present how we transform connected-vehicle data into an event log.

To create an event log from connected-vehicle data, we have to create events. To create events, we introduce maximal sequences of vehicle data. In a normal sequence of vehicle data, all elements are sorted from earliest to latest with one second in-between — they are *continuous*. Also, they share the same journey, run, warning, and state — they are *homogeneous*. Maximal sequences of vehicle data share these properties. Additionally, if a vehicle data instance exists in the same journey and run one second before or after a sequence, the state or warning is different from those in the sequence — they are *maximal*. Given the data in Table 1, the following maximal sequences are obtained:  $\langle 1, 2 \rangle, \langle 3, 4 \rangle, \langle 5, 6 \rangle, \langle 7 \rangle, \langle 9, 10, 11 \rangle, \langle 12 \rangle$ . The following statements capture this.

**Definition 3** (Maximal sequences). *Let  $V = (D, \pi_D)$  be vehicle data. Maximal sequences are a tuple*

$M_V = (S, \pi_S)$  with  $S \subseteq D^*$  as the set of sequences over vehicle data instances,  $\pi_S \in S \rightarrow \mathcal{U}_{v-map}$  such that,  $\forall s \in S : \{run, state, warning, time\} \subseteq dom(\pi_S(s))$  and  $\pi_S(s)(run) = \pi_D(d_1)(run) \in \mathcal{U}_{run}$ ,  $\pi_S(s)(state) = \pi_D(d_1)(state) \in P$ ,  $\pi_S(s)(warning) = \pi_D(d_1)(warning) \in W$ , and  $\pi_S(s)(time) = \pi_D(d_1)(time) \in \mathcal{U}_{time}$ . Moreover,  $\forall s = \langle d_1, \dots, d_n \rangle \in S$ , the following requirements are fulfilled:

- *continuous*:  $\pi_D(d_i)(time) + 1 = \pi_D(d_{i+1})(time)$  for  $i \in \{1, \dots, n-1\}$
- *homogeneous*:  $\pi_D(d_i)(att) = \pi_D(d_j)(att)$ ,  $att \in \{journey, run, state, warning\}$ ,  $i, j \in \{1, \dots, n\}$
- *maximal*: Let  $A = \{journey, run\} \subset \mathcal{U}_{v-att}$  and  $B = \{state, warning\} \subset \mathcal{U}_{v-att}$  be sets of attributes. If there is a  $d_0 \in D$ , such that  $\pi_D(d_0)(time) + 1 = \pi_D(d_1)(time)$  and  $\pi_D(d_0)(a) = \pi_D(d_1)(a)$ , for all  $a \in A$ , then there is a  $b \in B$  such that  $\pi_D(d_0)(b) \neq \pi_D(d_1)(b)$ . If there is a  $d_{n+1} \in D$ , such that  $\pi_D(d_n)(time) + 1 = \pi_D(d_{n+1})(time)$  and  $\pi_D(d_n)(a) = \pi_D(d_{n+1})(a)$ , for all  $a \in A$ , then there is a  $b \in B$  such that  $\pi_D(d_n)(b) \neq \pi_D(d_{n+1})(b)$ .

After introducing maximal sequences, we still need to create events, respectively, an event log from the vehicle data. In general, we extract maximal sequences from the vehicle data. Then, we use these sequences to create events. The case of an event is determined by a sequence's run, the timestamp by the first element of a sequence, and the activity by applying domain knowledge captured with the function *map*. In the following, we provide a closed expression that captures this process.

**Definition 4** (Transformation). Let  $M_V = (S, \pi_S)$  be maximal sequences and  $L = (E, \pi_E)$  be an event log. Let *map* be a function that returns an activity given a sequence, i.e.,  $map : S \rightarrow \mathcal{U}_{act}$ . There exist a bijective function from  $S$  to  $E$ ,  $trans : S \rightarrow E$  such that  $\forall s \in S, \pi_S(s)(run) = \pi_E(trans(s))(case)$ ,  $map(s) = \pi_E(trans(s))(activity)$ , and  $\pi_S(s)(time) = \pi_E(trans(s))(time)$ .

By combining the above-introduced mechanisms, we transform enriched connected-vehicle data into an event log. For each run, we discover maximal sequences. Each maximal sequence is converted into an event. We create an event log by mapping every maximal sequence contained in vehicles' data to events. As a result, using an event log, we can compare multiple runs across vehicles. The received event log from our data consists of over 16,000 cases and over 38,000 events.

## 5 OBTAINING SYSTEM MODELS

In this section, we demonstrate how we construct the de-jure model based on a preexisting description. In addition, using the transformed connected-vehicle data, we discover the best de-facto model concerning quality metrics and understandability for domain experts. First, we introduce a behavioral model designed by humans to show the system's behavior and its Petri net abstraction. Second, we apply process-discovery techniques on the event log we received after applying the previously mentioned preprocessing and transforming techniques.

### 5.1 Constructing the De-Jure Model

To implement the feature of hands-free driving, a behavioral model in the form of a flow chart was designed by humans. A general view concerning the behavior of the hands-free driving feature is depicted in Figure 3. In this work, we focus only on parts of the

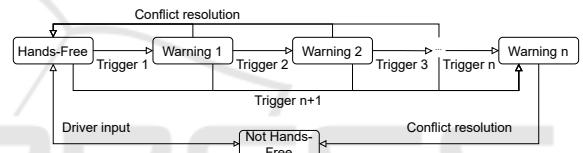


Figure 3: Flow chart showing the overall behavior of the hands-free driving feature.

system. The parts include a small selection of conflict resolutions and warnings. We focus on conflicts related to the absence of focus of a driver and three warning types. The chart showing the selected feature's behavior is depicted in Figure 4. The behavior

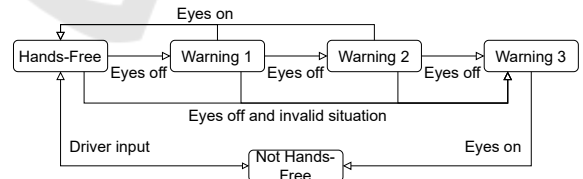


Figure 4: Flow chart for the selected conflict resolution strategies and warnings of the hands-free driving feature.

allowed by the model is as follows. Assume the starting state is hands-free driving without interruption. If the eyes are off the road, a first warning goes off. The driver can resolve this conflict by looking back on the road. A second warning appears if the conflict is not resolved, meaning eyes are still not looking on the road. Again, the situation can be solved by looking back on the road. If that is not done, a third warning happens. The system takes control of the car, including decelerating, and this situation can only be solved

by looking at the road. After a driver looks back on the road, the assistance system enters a non-hands-free-driving state. This takeover procedure can also happen at any time if an invalid situation for the system occurs and the eyes are not focused on the road. If conflicts are resolved before a third warning, the car continues driving hands-free without any warning. The model is translated into a Petri net by also consulting domain experts to check if the model and the recorded behavior align with each other, i.e., applying conformance-checking techniques. To convert this model into a Petri net, we assigned each state a place and a marking, as shown in (Reisig, 1985). The resulting Petri net is depicted in Figure 5. We used  $\tau$ -transitions since the reasons for starting the takeover can not properly be defined using a single transition and, therefore, can lead to an unreadable model. Moreover, after consulting domain experts, we introduced multiple transitions to reveal the input that drivers do to turn the hands-free driving mode off. By executing "State 1," "State 2," or "State 3," hands-free driving is disabled. The first conflict state is entered if the eyes are off, i.e., not focusing on the road. The conflict can be resolved if a driver looks back on the road, resulting in hands-free driving. The conflict escalates to the next state if a driver still does not look at the road. Again, the conflict can be resolved by looking back on the road. If it is not resolved, "Warning 3" happens, followed by "State 3" and turning the system off. This takeover procedure can also begin in other states. Instead of modeling this procedure in multiple states, we simplify the Petri net by using  $\tau$ -transitions, respectively, silent transitions. These transitions are represented as black-colored transitions in Figure 5. In Petri nets, silent transitions provide a flexible and expressive mechanism for modeling various aspects of process behavior. They enable the representation of internal actions, improving Petri nets' modeling capabilities. Silent transitions, for example, allow the synchronization of different parts of the Petri net. They can be used to control the execution of other transitions by requiring that certain conditions or constraints be met before proceeding. Silent transitions can act as synchronization points, waiting for specific conditions to be met before allowing subsequent transitions to fire. Using the formerly created event log, we apply conformance-checking techniques to this de-jure model. We load the de-jure model and event log in ProM (van Dongen et al., 2005). By using (Adriansyah et al., 2011), we observe a fitness score of 0.97, and by using (Adriansyah, 2014), we receive a precision score of 0.71 and a generalization score of 1.

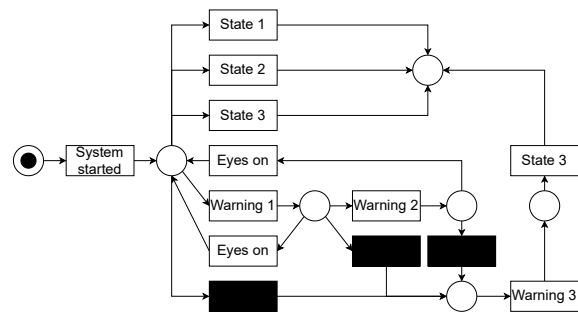


Figure 5: Petri net showing the behavior of the hands-free driving system based on human design. After executing "System started," the vehicle is in the state of hands-free driving.

## 5.2 Discovering the De-Facto Model

In contrast to constructing a Petri net from a pre-existing description, we apply process-discovery techniques to the former extracted event log (see Section 4.2). We apply different process-discovery techniques, each implemented in ProM (van Dongen et al., 2005). Furthermore, we evaluate the models' soundness, fitness, precision, and generalization capabilities using the implemented algorithms in ProM. Concerning the region miner, a transition system was mined by using the event name as a backward key and a set abstraction of size one. Moreover, self-loops were removed, and label-splitting was performed. As denoted in Table 2, some algorithms lead to non-workflow-net models or are unsound. These models are not considered to be good candidates for explaining the underlying behavior in the event log. Also, as shown in Table 2, filtering influences fitness and precision. More filtering leads to less fitting models, but the models are more precise. The generalization score is for every model the same. By considering the scores and after consulting domain experts, the model discovered by the region-based miner is chosen as the best de-facto model. The model is depicted in Figure 6. The model is not a directly-follows graph. First, transition names appear multiple times. Second, each state has a contextual meaning.

As shown in the model in Figure 5, after starting the system, "State 1," "State 2," and "State 3" can be executed. Moreover, the first and second warnings can be executed, as well as the procedure starting with the third warning. Executing "State 1," "State 2," or "State 3" leads to the non-handsfree-driving state. Firing any "Warning 1" transition leads to the state in which this warning is active. As described earlier, the conflict can be resolved with "Eyes on," or the conflict can escalate with "Warning 2." However, there is also the option of turning the hands-free driving sys-

Table 2: Conformance checking results for de-jure model and different algorithms and filters. To check soundness, WOFLAN is used (Verbeek et al., 2001). For fitness and alignments, (Adriansyah et al., 2011) is used. Concerning precision and generalization, the work in (Adriansyah, 2014) is used.

Algorithm	Filter parameter	WF-net	Sound	Fitness	Precision	Generalization
Alpha	-	No	-	-	-	-
Alpha +	-	Yes	No	-	-	-
Alpha ++	-	Yes	No	-	-	-
Region Miner	-	Yes	Yes	0.99	0.78	1.00
Inductive Miner	-	Yes	Yes	1.00	0.43	1.00
Inductive Miner Infrequent	0.2	Yes	Yes	0.96	0.78	1.00
Inductive Miner Infrequent	0.4	Yes	Yes	0.96	0.75	1.00
Inductive Miner Infrequent	0.6	Yes	Yes	0.96	0.75	1.00
Inductive Miner Infrequent	0.8	Yes	Yes	0.96	0.90	1.00
Directly-follows Miner	-	Yes	Yes	1.00	0.77	1.00
Directly-follows Miner	0.8 (paths)	Yes	Yes	0.85	1.00	1.00
Directly-follows Miner	0.6 (paths)	Yes	Yes	0.85	1.00	1.00
Directly-follows Miner	0.4 (paths)	Yes	Yes	0.58	1.00	1.00
Directly-follows Miner	0.2 (paths)	Yes	Yes	0.58	1.00	1.00
De-Jure Model	-	Yes	Yes	0.97	0.71	1.00

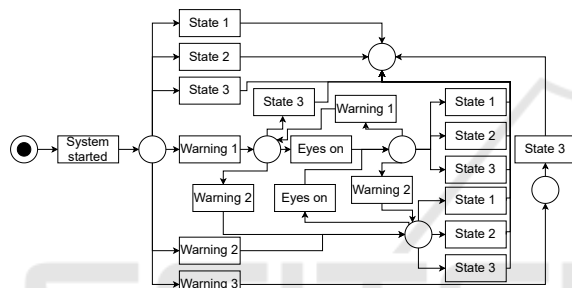


Figure 6: Petri net obtained by using the region-based miner. The underlying transition system was built using the event name as the backward key, set abstraction with size one, and removal of self-loops. Also, label-splitting was used.

tem off. By firing "Warning 2," all ways to deactivate hands-free driving mode are executable; therefore, conflict resolution may not occur. If, at any point, conflicts are resolved, the state is not the same as after starting the system. The difference between this conflict-resolved state and the initial hands-free driving state is the possible execution of "Warning 3." In the data, "Warning 3" is executed only after the system starts, without any conflict or conflict resolution. Therefore, this transition is only executable in this position.

## 6 EVALUATION

In this section, we evaluate our results. In particular, we compare the conformance-checking results of both models. Moreover, we discuss the differences between the models and highlight the reasons for these.

As mentioned earlier, the fitness score of the de-jure model is 0.97, with a precision score of 0.71 and a generalization score of 1. For the de-facto model, the fitness, precision, and generalization scores are 0.99, 0.78, and 1. While precision and generalization have similar scores, the difference in fitness is caused by different conflict-resolution strategies and by directly firing "Warning 2" before firing "Warning 1."

We presented the models to domain experts and discussed the differences. The first deviation deals with conflict resolution strategies. Applying process-mining techniques to the data shows that the system can be turned off during a conflict. This conflict resolution strategy contrasts the designed way. Designed is the conflict resolution strategy by looking back on the road. However, domain experts clarified that turning the system off is possible but was not captured in the state model we used for the de-jure model. The second deviation is skipping "Warning 1" and directly executing "Warning 2." Domain experts clarified that logging can be too imprecise to capture "Warning 1." Therefore, the first warning happens but is not captured. The third and last deviation focus on the takeover procedure. The de-facto model shows that this procedure can only happen if no other event occurs after the system starts. The reason is the underlying data. The procedure is executed after the system starts and only appears once in the data. As a result, this procedure can only be executed in the model after the system has been started. In general, the discovered process model captures more behavior than was initially captured by humans. The reason is that domain experts may skip notating some information at this level of design and prioritize clarity of the design by avoiding notation clutter. Although other documents are available, they are closer to the implementation level and do not suit our purpose.



## 7 DISCUSSION

In this section, we summarize our results and provide pointers for future work. In this work, we demonstrated how to enrich and transform continuous data recorded by cyber-physical systems into an event log. We applied process-discovery and conformance-checking techniques on the created event log, resulting in the best de-facto model for domain experts. Furthermore, we transformed an existing model describing the hands-free driving system into a Petri net. In addition, we applied conformance checking on the de-jure model using the created event log. We compared the models and explained their differences. As denoted, the de-facto model shows more behavior than the de-jure model, but domain experts explained that.

In general, we demonstrated that applying process mining in the cyber-physical systems domain is valuable. By applying conformance checking, we verified that the de-jure model is close to reality. In addition, by applying process-discovery techniques, we revealed that drivers' behavior differs from the behavior that the initial behavioral model assumed. As a result, we are confident that applying process-mining to other systems is beneficial to understand if the designed behavior matches reality and what the actual behavior looks like. To the best of our knowledge, we are the first that applied process-mining techniques on connected-vehicle data.

Nevertheless, there are still open challenges and pointers for future work. First, the de-facto model depicts multiple decision points, for example, firing "Warning 2" or "Eyes on." Thus, applying decision mining to find out why specific actions are taken seems interesting to understand the drivers' behavior better. The enhanced information may help to understand the behavior better. Second, exploring how long vehicles are in a particular state is interesting to get insights into the features' usage. Third, the takeover procedure is more complex and consists of multiple sub-processes. Modeling the procedure and preprocessing the data to capture this procedure are interesting to receive a complete picture of the interactions between driver and car. Fourth, cars transmit an enormous amount of data. The amount of received data will increase in the future. Therefore, there is a need to evaluate if our proposed approach can handle such volumes. Also, applying the techniques in a streaming setting seems beneficial, for example, to faster analyze the system. Finally, we modeled one feature of cars as one system. However, cars consist of multiple systems that interact with each other. An approach that reveals such interactions of systems is presented

in (van Eck et al., 2016a). Applying the idea of the mentioned work in this car-related setting can lead to more and deeper insights.

## ACKNOWLEDGEMENTS

This research is supported by Ford-RWTH Aachen University Alliance program. We thank the Alexander von Humboldt (AvH) Stiftung for supporting our research.

## REFERENCES

- Adriansyah, A. (2014). *Aligning observed and modeled behavior*. PhD thesis, Mathematics and Computer Science.
- Adriansyah, A., van Dongen, B. F., and van der Aalst, W. M. P. (2011). Conformance checking using cost-based fitness analysis. In *EDOC*, pages 55–64. IEEE Computer Society.
- Astromskis, S., Janes, A., and Mairegger, M. (2015). A process mining approach to measure how users interact with software: an industrial case study. In *ICSSP*, pages 137–141. ACM.
- Benevento, E., Pegoraro, M., Antoniazzi, M., Beyel, H. H., Peeva, V., Balfanz, P., van der Aalst, W. M. P., Martin, L., and Marx, G. (2023). Process modeling and conformance checking in healthcare: A covid-19 case study. In *Process Mining Workshops*, pages 315–327, Cham. Springer.
- Grüger, J., Geyer, T., Kuhn, M., Braun, S. A., and Bergmann, R. (2021). Verifying guideline compliance in clinical treatment using multi-perspective conformance checking: A case study. In *Process Mining Workshops - ICPM*, pages 301–313, Cham. Springer.
- Janiesch, C., Koschmider, A., Mecella, M., Weber, B., Burrattin, A., Di Ciccio, C., Fortino, G., Gal, A., Kanengiesser, U., Leotta, F., Mannhardt, F., Marrella, A., Mendling, J., Oberweis, A., Reichert, M., Rinderle-Ma, S., Serral, E., Song, W., Su, J., Torres, V., Weidlich, M., Weske, M., and Zhang, L. (2020). The internet of things meets business process management: A manifesto. *IEEE Systems, Man, and Cybernetics Magazine*, 6(4):34–44.
- Koschmider, A., Janssen, D., and Mannhardt, F. (2020). Framework for process discovery from sensor data. In *EMISA*, pages 32–38. CEUR-WS.org.
- Lee, E. A. (2008). Cyber physical systems: Design challenges. In *IEEE (ISORC)*, pages 363–369. IEEE Computer Society.
- Makke, O. and Gusikhin, O. (2021). Robust iot based parking information system. In *Smart Cities, Green Technologies, and Intelligent Transport Systems*, pages 204–227, Cham. Springer.
- Maruster, L., Faber, N. R., Jorna, R. J., and van Haren, R. J. F. (2008). A process mining approach to analyse

- user behaviour. In *WEBIST*, pages 208–214. INSTICC Press.
- Reisig, W. (1985). *Petri Nets: An Introduction*, volume 4 of *EATCS Monographs on Theoretical Computer Science*. Springer, Heidelberg.
- Rubin, V. A., Mitsyuk, A. A., Lomazova, I. A., and van der Aalst, W. M. P. (2014). Process mining can be applied to software too! In *ESEM*, pages 57:1–57:8. ACM.
- van der Aalst, W. M. P. (2016). *Process Mining - Data Science in Action, Second Edition*. Springer, Heidelberg.
- van Dongen, B. F., de Medeiros, A. K. A., Verbeek, H. M. W., Weijters, A. J. M. M., and van der Aalst, W. M. P. (2005). The prom framework: A new era in process mining tool support. In *Applications and Theory of Petri Nets*, pages 444–454, Heidelberg. Springer.
- van Eck, M. L., Sidorova, N., and van der Aalst, W. M. P. (2016a). Discovering and exploring state-based models for multi-perspective processes. In *BPM*, pages 142–157, Cham. Springer.
- van Eck, M. L., Sidorova, N., and van der Aalst, W. M. P. (2016b). Enabling process mining on sensor data from smart products. In *RCIS*, pages 1–12. IEEE.
- Verbeek, H. M. W., Basten, T., and van der Aalst, W. M. P. (2001). Diagnosing workflow processes using woflan. *Comput. J.*, 44(4):246–279.
- Xu, H., Pang, J., Yang, X., Ma, L., Mao, H., and Zhao, D. (2020). Applying clinical guidelines to conformance checking for diagnosis and treatment: a case study of ischemic stroke. In *BIBM*, pages 2125–2130. IEEE.

