Institute of Software Engineering
Empirical Software Engineering Group

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Exploring Runtime Monitoring Techniques in the Automotive Domain for Advanced Driver-Assistance Systems

Andrijana Radic

Course of Study:	Informatik
Examiner:	Prof. Dr. Stefan Wagner
Supervisor:	Eva Zimmermann, M.Sc., Pavel Nedvědický, Ing.
Commenced:	June 14, 2023
Completed:	December 14, 2023

Abstract

In the real world, a dynamic and unpredictable environment, an "Advanced Driver-Assistance System" (ADAS) should be safe for itself, pedestrians, and other obstacles. One possible approach to ensure the safety of ADAS is "Runtime Monitoring" (RM). In this context, additional formal safety mechanisms are added to the system. This thesis aims to explore RM for ADAS. Our main contributions consist of three parts. Firstly, we conducted a rapid review to find relevant RM techniques for ADAS. We provided detailed information on our search query and the filter criteria for the papers. We extracted the information from 16 remaining relevant papers and applied an existing taxonomy to classify the techniques. Secondly, we defined the hardware criteria given by the "Robot Operating System" (ROS)-based "Autonomous Research Vehicle" (ARV) called *Mecabot TX*, the use case for the prototype, and four safety requirements for our runtime monitor. In our use case, the system performed "Advanced Emergency Braking" (AEB) without the intervention of a driver. Based on that, we then chose one of the 16 techniques for our prototype. The technique we evaluated to be optimal was *rtamt* that relied on specifications written in "Signal Temporal Logic" (STL). Thirdly, we implemented one passive runtime monitor for each safety requirement using *rtamt*. We proposed an architecture consisting of an object tracker, the AEB logic, and the runtime monitors. We verified our monitors and conducted an experiment to test the implementation. We identified that the runtime monitors successfully detected multiple violations for each safety requirement during the test run. By analyzing the violations, we gained helpful insights for debugging the system and improvements for its safety. Therefore, our work paves the way for future research in the area of RM for ADAS.

Kurzfassung

In der realen Welt, einer dynamischen und unvorhersehbaren Umgebung, sollten fortschrittliche Fahrerassistenzsysteme (ADAS) die eigene Sicherheit sowie die Sicherheit für andere Verkehrsteilnehmer wie beispielsweise Fußgänger gewährleisten. Ein möglicher Ansatz, um die Sicherheit solcher Fahrerassistenzsysteme zu gewährleisten, ist "Runtime Monitoring" (RM). Dabei werden zusätzliche formale Sicherheitsmechanismen in das System integriert. Das Ziel dieser Arbeit ist die Erforschung des Einsatzes von RM im Kontext fortschrittlicher Fahrerassistenzsysteme. Unser Hauptbeitrag lässt sich in mehrere Teile gliedern. Zum einen führten wir eine "Rapid Review" durch, um relevante RM-Techniken im Kontext von Fahrerassistenzsystemen zu identifizieren. Dabei beschrieben wir detailliert unsere verwendete Suchanfrage und unsere Filterkriterien. Wir extrahierten Informationen aus den 16 verbliebenen relevanten wissenschaftlichen Arbeiten. Zudem wandten wir eine vorhandene Taxonomie zur Klassifizierung dieser identifizierten RM-Techniken an. Des Weiteren definierten wir die Hardwarekriterien des auf dem "Robot Operating System" (ROS) basierenden autonomen Fahrzeugroboters namens *Mecabot TX*, unseren Anwendungsfall für den Prototyp und vier Sicherheitsanforderungen für unseren Laufzeitmonitor. In unserem Beispiel führte ein Notbremsassistenzsystem (AEB) eine autonome Notbremsung ohne das Eingreifen des Fahrers durch, sobald ein relevantes Hindernis vor dem Fahrzeugroboter erkannt wurde. Daraufhin wählten wir eine der 16 Techniken für unseren Prototyp basierend auf unserer Klassifikation der RM-Techniken und den erhobenen Kriterien aus. Die von uns als optimal bewertete Technik lautete dabei *rtamt*, bei der die Spezifikationen in "Signal Temporal Logic" (STL) definiert werden. Unter Verwendung der *rtamt* Bibliothek implementierten wir für jede spezifizierte Sicherheitsanforderung einen passiven Laufzeitmonitor. Unsere finale Architektur bestand aus einer Objekterkennung, dem autonomen Notbremssystem und den Laufzeitmonitoren. Wir verifizierten die Implementierung unserer Monitore und führten ein Experiment mit dem Gesamtsystem in einer Laborumgebung durch. Dabei stellten wir fest, dass die implementierten Laufzeitmonitore während des Testlaufs erfolgreich mehrere Verstöße gegen jede unserer vier Sicherheitsanforderungen erkannten. Durch die manuelle Analyse dieser Verstöße erhielten wir hilfreiche Einblicke zur Fehlerbehebung des Systems und für mögliche Verbesserungen der Sicherheit unseres Fahrerassistenzsystems. Somit legt unsere Arbeit den Grundstein für zukünftige Forschungen im Bereich des RM für Fahrerassistenzsysteme, die an unsere Ergebnisse anknüpfen.

Contents

List of Figures

List of Tables

List of Listings

Acronyms

AD Autonomous Driving. 19

ADAS Advanced Driver-Assistance System. 9, 17

AEB Advanced Emergency Braking. 9, 17

ARV Autonomous Research Vehicle. 18

BPMN Business Process Model and Notation. 33

COCO Common Objects in Context. 27

DeepSORT Simple Online and Realtime Tracking with a Deep Association Metric. 27

LiDAR Light Detection and Ranging. 21

RM Runtime Monitoring. 11, 18

ROS Robot Operating System. 24

RR Rapid Review. 9, 31

RV Runtime Verification. 22

SORT Simple Online and Realtime Tracking. 27

SR Safety Requirement. 11, 53

STL Signal Temporal Logic. 23

YOLO You Only Look Once. 24

1 Introduction

To introduce this thesis, we first explain the motivation behind our work in Section 1.1. Next, we describe the problem statement in Section 1.2. Finally, we give an overview of our thesis structure in Section 1.3.

1.1 Motivation

In our day-to-day life, we are surrounded by a multitude of vehicles. Either they are private vehicles like cars, industrial vehicles like trucks, or public transport vehicles like buses. With this diverse setting on the road from different drivers to different sized vehicles, it is difficult to maintain safety. As an example, vehicles often have blind spots, where the drivers do not see other road users [Bau22]. Achieving road safety is challenging, e.g., due to the complexity of human behavior, blind spots, and unpredictable environmental conditions [Bau22; Eur21]. With 2.4 million reported road traffic accidents in Germany in 2022, there is still much room for improvements in decreasing this amount [Sta23a]. The causes of the accidents are diverse. There were driver-related accidents, e.g., due to intoxicated drivers, inappropriate velocity, or insufficient safety distance [Sta23b]. Other causes for the accidents were, e.g., harsh weather conditions like rain, snow, ice, and fog, but also, e.g., wild animals on the roads [Sta23c]. To decrease the number of road accidents and to increase the overall road safety, there is research done both in the industry [MMGP19] and in scientific communities targeting automated driving.

The SAE On-Road Automated Driving (ORAD) Committee [SAE21b] defines different levels of automated-driving (autonomous) systems. The different levels describe systems that either assist the driver while driving the vehicle or systems that fully drive on their own (in all or only in restricted conditions). With higher levels, the driver's importance and control over the vehicle decrease. Contrary to that, with higher levels, the vehicle and its systems gain importance and control [SAE21a; SAE21b].

An "Advanced Driver-Assistance System (ADAS)" [Eur21, p. 2] is such a system that assists the driver while driving [Eur21]. An example of such an ADAS is, e.g., an "Advanced Emergency Braking (AEB)" [Eur21]. The AEB helps with braking if the distance to an obstacle falls below a safety distance [Eur21]. An ADAS has several advantages, e.g., they increase the reaction time and the perception of the driver. Collected and processed sensor information is sent to the driver, which can increase the driver's perception regarding the blind spots, if they are correctly placed and working. Additionally, some ADAS can take over the car in a critical moment (as the AEB), e.g., if the driver is distracted [Eur21].

Overall, increasing the road safety is of great importance for everyone and everything on the road, from pedestrians to cyclists and the drivers of the vehicles themselves.

1.2 Problem Statement

We have noticed that there is still improvement possible regarding road safety and that there is work done to use ADAS to improve the road safety. But using ADAS in the real world to increase the road safety leads us to a problem, that we will explore with this thesis. As the aim of adding driving automation to a car, e.g., by using an ADAS, is to provide an additional safety instance to support the driver, the ADAS itself should be safe and reliable. One possible approach to ensure the safety of ADAS is “Runtime Monitoring (RM)” [BFFR18, p. 1]. By using RM, additional formal safety mechanisms are added to the system. The mechanisms can be useful regarding debugging and testing of the ADAS [BFFR18]. Therefore, it increases the safety of the ADAS as it can be tested with multiple approaches (RM and more). As there is still a lack of research and literature reviews on RM regarding ADAS, this thesis aims to explore RM for ADAS on an “Autonomous Research Vehicle (ARV)” [KCDK15, p. 102].

1.3 Structure

In the following, we outline the structure of this thesis.

Chapter 2 – Background and Foundations Here, we explain the relevant information that is needed for the reader to understand the rest of this thesis.

Chapter 3 – Related Work In this chapter, we summarize the influence of the conference on RM and briefly describe a taxonomy for RM.

Chapter 4 – Study Design Next, we describe our research questions and briefly summarize the used methodology.

Chapter 5 – Rapid Review The first part of the main part is the rapid review. We describe the process and results in this chapter.

Chapter 6 – Prototype Technique Selection The second part of the main part is the technique selection for the prototype. This includes defining requirements and a use case for the prototype.

Chapter 7 – Prototype Implementation The last part of the main part is the implementation description, where we explain our realized architecture as well as the verification of the monitors, and the description and evaluation of our experiment.

Chapter 8 – Discussion Here, we evaluate this thesis in the context of the research questions and bring the results into context. Furthermore, we describe the threats to validity.

Chapter 9 – Conclusion and Outlook In the end, we summarize our work, its benefits and limitations, possible future work and the lessons we learned with this thesis.

2 Background and Foundations

In this chapter, we first provide necessary background information for understanding the general topic of this thesis in Section 2.1. Secondly, we provide foundational information to understand this thesis and its outcomes in Section 2.2.

2.1 Background

We start with providing insights into relevant background knowledge of this thesis. The relevant topics include: "Autonomous Driving (AD)" [ZBK20, p. 172], "Advanced Driver-Assistance System (ADAS)" [Eur21, p. 2], and "Runtime Monitoring (RM)" [BFFR18, p. 1].

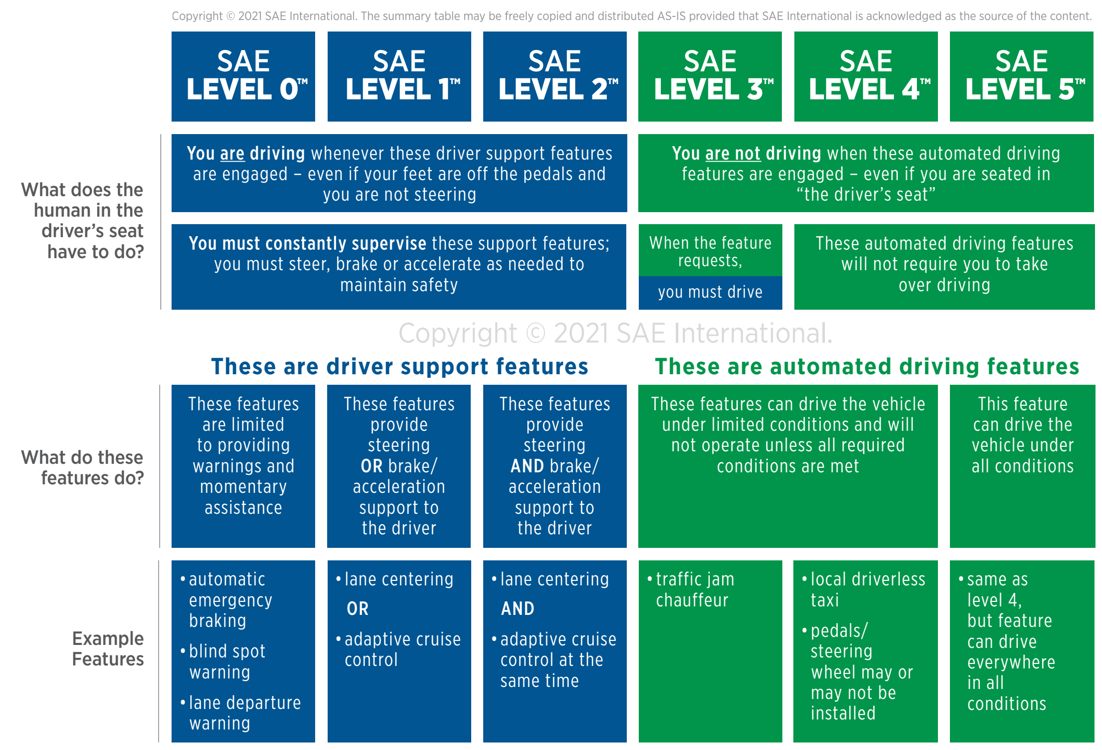
2.1.1 Driving Automation

The SAE On-Road Automated Driving (ORAD) Committee [SAE21b] describes different levels of automated-driving (autonomous) systems. The six different levels are separated by the degree of control by the driver of the vehicle. At the lowest level, the driver has full control over the vehicle. At the highest level, the vehicle has full control without interruptions by the driver. The levels in between are the spectrum between the lowest and highest level. In the following, we describe the levels based on the standard provided by SAE On-Road Automated Driving (ORAD) Committee [SAE21a; SAE21b] in Figure 2.1:

Level 0 This level is called "no driving automation" [SAE21b]. At this level, the driver has an active role while driving, even though there are features involved. Those features are called "driver support features" [SAE21a]. As they are only for the support of the driver while driving, they have to be constantly supervised by the driver. The driver may have to intervene to maintain safety, e.g., by braking. The support features at this level warn and assist the driver for short moments [SAE21a; SAE21b].

Level 1 This level is called "driver assistance" [SAE21b]. At this level, the driver has an active role while driving, even though there are features involved. Those features are called driver support features. As they are only for the support of the driver while driving, they have to be constantly supervised by the driver. The driver may have to intervene to maintain safety, e.g., by braking. The support features at this level help with slowing down / speeding up or steering [SAE21a; SAE21b].

Level 2 This level is called "partial driving automation" [SAE21b]. At this level, the driver has an active role while driving, even though there are features involved. Those features are called driver support features. As they are only for the support of the driver while driving, they have to be constantly supervised by the driver. The driver may have to intervene to maintain safety,



	SAE LEVEL 0™	SAE LEVEL 1™	SAE LEVEL 2™	SAE LEVEL 3™	SAE LEVEL 4™	SAE LEVEL 5™
What does the human in the driver's seat have to do?	**You are driving** whenever these driver support features are engaged – even if your feet are off the pedals and you are not steering			**You are not driving** when these automated driving features are engaged – even if you are seated in "the driver's seat"		
	You must constantly supervise these support features; you must steer, brake or accelerate as needed to maintain safety			When the feature requests, you must drive	These automated driving features will not require you to take over driving	
	These are driver support features			These are automated driving features		
What do these features do?	These features are limited to providing warnings and momentary assistance	These features provide steering **OR** brake/ acceleration support to the driver	These features provide steering **AND** brake/ acceleration support to the driver	These features can drive the vehicle under limited conditions and will not operate unless all required conditions are met		This feature can drive the vehicle under all conditions
Example Features	• automatic emergency braking • blind spot warning • lane departure warning	• lane centering **OR** • adaptive cruise control	• lane centering **AND** • adaptive cruise control at the same time	• traffic jam chauffeur	• local driverless taxi • pedals/ steering wheel may or may not be installed	• same as level 4, but feature can drive everywhere in all conditions

Figure 2.1: Driving automation levels introduced by the SAE International [SAE21a], cropped

e.g., by braking. The support features at this level help with slowing down / speeding up and steering. Which means that not only one feature needs to be included, but multiple features with different scopes [SAE21a; SAE21b].

Level 3 This level is called "conditional driving automation" [SAE21b]. At this level, the driver has a passive role while driving. The features are called "automated driving features" [SAE21a]. The features at this level can only work when all the conditions for it are met. It can also generally only take over driving under limited conditions. When those conditions are not met, the driver is requested to drive instead [SAE21a; SAE21b].

Level 4 This level is called "high driving automation" [SAE21b]. The features are called automated driving features. At this level, the driver always has a passive role while driving. The automated driving features can only work when all the conditions for it are met. It can also generally only take over driving under limited conditions. When those conditions are not met, the driver is not requested to drive instead. There may not even exist a way that allows manually steering the wheel or pushing the pedals [SAE21a; SAE21b].

Level 5 This level is called "full driving automation" [SAE21b]. The features are called automated driving features. At this level, the driver has always a passive role while driving because the vehicle can drive on its own under all conditions and in all places [SAE21a; SAE21b].

As already mentioned, the levels zero to two provide driver support features and levels three to five provide automated driving features. As the driver support features are relevant to our topic, we describe it further in the next section (see Section 2.1.2).

2.1.2 Advanced Driver-Assistance Systems

In the section before, we introduce the different levels of AD features. The vehicle gains more control at levels higher than zero. The system that takes over control in levels zero to two is called an ADAS. At levels three to five, they are AD systems [Eur21].

One relevant ADAS that we need is “Advanced Emergency Braking (AEB)” [Eco14]. AEB systems are systems that provide momentary assistance to avoid forward collisions of the vehicle with obstacles. This assistance includes automatically detecting the obstacles and decelerating the vehicle to avoid said collisions or reduce the damage by said collisions [Eco14].

The European Commission states that ADAS improve the road safety, especially ADAS at level zero, have the highest potential of increasing the road safety. Additionally, they state some challenges with such systems. The main challenges are regarding the interaction between the driver and ADAS. This includes distraction issues of the driver, as the driver has to drive and simultaneously monitor the ADAS. Additional challenges are the trust in the system, transparency, and having the understanding of all the features the ADAS provides. Besides the interaction challenges of driver and ADAS, there are also technical challenges stated, e.g., accuracy of the detections and range limitations of sensors. All these challenges have to be addressed to benefit from those safety mechanisms [Eur21].

Sensors

We describe the previously mentioned sensors further. The main purpose of sensors is to collect environmental data [Eur21]. The collected data can then be used in, e.g., an ADAS. The sensor data and the range of sensors depend on the specific sensor [RCG22]. Examples of such sensors are “Light Detection and Ranging (LiDAR)” [RCG22, p. 1], camera, and radar [RCG22]. The range of sensors can be a limitation for systems [Eur21]. Therefore, when sensors are used for observing the environment, then oftentimes multiple sensors and sensor types are used to solve range limitations and for fulfilling the requirements the system has [RCG22]. In the following, we briefly describe the mentioned sensor examples.

LiDAR LiDAR is an abbreviation of “Light Detection and Ranging” [RCG22, p. 1]. The basic idea of a LiDAR is to collect information about the environment and transform them into 3D digital representations. Mainly, it calculates distances to obstacles. It does that by sending laser pulses. These pulses then reach the obstacle and return to the LiDAR. With the timely information, it calculates the distance to the obstacle in real-time [RCG22].

Camera Cameras are used for detecting objects and classifying objects. The camera images create visual descriptions of the environment. A huge factor to consider when using camera images is that they are easily affected by foggy, dark, or rainy weather conditions [RCG22].

Radar Similar to cameras, if radar sensors are positioned strategically around the vehicle, it can detect obstacles from all angles. As LiDAR sensors, radar sensors also use pulses and wait for their return. Contrary to LiDAR sensors, radar sensors work with radio waves. Radar sensors provide, e.g., precise and direct distance measurements. Additionally, radar sensors are unaffected by variations in lighting [RCG22].

With the knowledge on what ADAS are and how sensors are integrated into them, we can take a look at RM itself and how it is connected to ADAS.

2.1.3 Runtime Monitoring

As described by Bartocci et al. [BFFR18], “Runtime Verification (RV)” [BFFR18, p. 1] describes the process of checking the correctness of a property during runtime. There are inconsistencies with the definition of RM. Sometimes in research, it is described as a synonym of RV. Other times, RM is described as more specific than RV as it is the observation and active interaction with a system when checking for the correctness of properties. In this thesis, we use RM and RV synonymous. Generally, RM is useful regarding the testing and debugging of systems. Additionally, it guarantees the safety and the robustness of systems. We only name a few examples mentioned by Bartocci et al. [BFFR18].
In Section 2.1.2, we already mentioned that ADAS benefit from safety mechanisms [Eur21]. As RM guarantees the safety of systems, we think that RM can be useful in ADAS. Therefore, we look further into RM. RM can be separated into three parts that have to be done [BFFR18]:

Specification Wanted and unwanted system behavior is described in the first part [BFFR18].

Generation A monitor is generated based on the defined specifications in the second part [BFFR18].

Connection The connection of the monitor to the system is created in the third part [BFFR18].

In the next sections, we further describe the mentioned parts of RM.

Specification

In the following section, we define the information we need to define the wanted and unwanted system behavior. This includes defining events, traces, properties and specifications itself [BFFR18].

Event Any observation concerning the system is an event. Therefore, events are like a snapshot of the system and its behavior at a given time. Events can be about the internal behavior or the external system behavior [BFFR18].

Traces A series of events is called trace. It describes system behavior over time instead of just in a specific moment [BFFR18].

Property A set consisting of traces is called a property. The set may be finite. A property is unique as well as language independent. There are explicit and implicit properties [BFFR18].

Specification It describes the textual formalization of the property of a system [BFFR18].

Language Features Specifications are written in a specification language. Some of the relevant features of such a language are: is it executable or declarative, is it finite or infinite, and the time. Executable specifications are specified and then they are immediately executable. Declarative specifications are specified, then the executable object is generated from this specification. An example for each are: state machines (executable) and temporal logic (declarative). The next relevant point is whether the traces are finite or infinite. There are languages that satisfy the demands of finite traces and others for infinite traces. An example for each are: state machines (finite) and temporal logic (infinite). Generally, sometimes infinite traces are parted into sections of finite traces to evaluate them in real-time. Time constraint mechanisms can be added in both, declarative and executable languages. E.g., in declarative languages, you define that some value should change in the next x seconds [BFFR18].

Temporal Logic In the following, we describe one declarative language as an example. We already mentioned temporal logic previously. "Linear Temporal Logic (LTL)" [BFFR18, p. 7] is the base variant of the temporal logics. There are two types of LTL, LTL with future operators and LTL with past operators. Future and past are referring to the time. Future operators that are directly available are *next* and *until*. Operators that are derived from them are *eventually* and *always*. The semantics of the operators is intuitive from the name. Therefore, we do not describe this further. The past operators that are directly available are *previous* and *since*. The language features that are previously described for temporal logics are valid for LTL [BFFR18].
There are multiple variants of temporal logics that provide additional expressiveness compared to LTL. To us, "Signal Temporal Logic (STL)" [BFFR18, p. 9] with past operators and future operators is of relevance. Compared to LTL, STL has signals as input. Signals are multiple pairs of the following form: (time, value). STL also provides a quantitative value for a formula, the robustness. The robustness describes how much a signal satisfies or violates the specification (formula). The provided information on STL and its formal definition can be found in [NY20].

Generation

We now summarize relevant knowledge on the generation of such monitors, especially how such a monitoring setup looks like and how different approaches can be used for the setup [BFFR18].

Standard Setup The standard setup consists of the monitor, the system to observe and instrumentation between monitor and system. The instrumentation is the connection between monitor and system [BFFR18].

Monitor As previously described, monitors are executable and are generated from specifications. The monitor is the component that observes the system at runtime. It checks whether the defined properties are violated or satisfied. With enough gathered information, monitors may evaluate a decision (verdict). It can not be withdrawn and is forwarded to other components that can handle the decision. Monitors are generally trusted and should be correct themselves. The automatic generation of monitors from specifications increases this as monitor code is standardized. Monitors typically have minimal, or ideally no, impact on the operation of the observed system [BFFR18].

Instrumentation We already described this as the connection between the monitor and the system that is observed. It defines which information of the system's execution become observable for analysis by the monitor. The instrumentation is responsible for filtering relevant and observable data from irrelevant and hidden data. The relevant and observable data is sent to the monitor [BFFR18].

Programming There exists "Monitor-Oriented Programming" [BFFR18, p. 14], we explain a few aspects of it here. It is a design concept in which the monitors code and the observed systems code are as separated as possible. Contrary to the monitors we talked about until now, monitors are a key aspect to raise violations and to act on these violations. The actions of the monitor can be that it suppresses unwanted behavior or even adapt to such behavior of the system in the future [BFFR18].

Design Details We introduced the general setup of such a monitor and the system to observe. Within this setup, multiple design choices are made. We explain some design choices to give an idea of what they are about. The first relevant design choice is whether the monitor should be online or offline. Online monitors evaluate the information while the system is running. Offline monitors evaluate the information after the system finishes its execution. Both approaches have advantages and disadvantages that need to be considered when deciding which approach is appropriate. The second design choice one can make is that the execution of the monitor and system can be synchronous. That means that the system provides relevant information to the monitor and then waits until the monitor evaluates a result until it continues. Alternatively, with the asynchronous approach, the system does not have to wait. Mechanisms to guarantee synchronicity have a higher interference with the system. The decisions on which design choice depend on the use case and requirements one wants to fulfill [BFFR18].

Connection

For creating the connection between the system and the monitor, instrumentation is needed. We already introduced instrumentation previously, but we want to add some additional information. Generally, the connection between monitor and system varies depending on the observed system. The system can be hardware or software. For this thesis, we focus on software systems. For monitoring such systems, additional code is needed, as there is the need to define the outputs the system should provide to the monitor. There are limitations concerning instrumentation for software systems. If the instrumentation code is not working or not executed, then there is relevant information missing. In settings where maintaining real-time requirements is crucial to satisfy safety-critical demands, such an outcome might be fatal [BFFR18].

As we now have the knowledge on the characteristics of ADAS, RM, and on how RM is connected to ADAS, we can now continue with some foundations to understand what was done in this thesis.

2.2 Foundations

After summarizing the relevant information for understanding the topic of this thesis, we summarize information to fully understand the contents of this thesis, especially the implementation. This includes information that is not common knowledge in software engineering. The relevant topics are the "Robot Operating System (ROS)" [AĐHM23, p. 1], "*rtamt*" [NY20, p. 1], and "*darknet_ros*" [Bje18] with "You Only Look Once (YOLO)v3" [RF18, p. 1].

2.2.1 Robot Operating System

"ROS is a collection of open-source robotics libraries and tools centered around robotics software development" [AĐHM23, p. 1]. ROS finds application in a wide range of areas [AĐHM23]. Examples for such areas are: "navigation tasks" [AĐHM23, p. 13], autonomous vehicles, and "home and medical care" [AĐHM23, p. 13]. For this thesis, we focus on ROS 1 [AĐHM23]. There are many concepts at different levels that could be explained, e.g., "the Filesystem level, the Computation Graph level, and the Community level" [Ope22]. As we only need the ROS Computation Graph level for understanding this thesis, we focus on the concepts at this level.

Computation Graph Level

The ROS processes together form a decentralized network. The processes collaborate on data processing tasks. The components of such a graph include nodes, topics, and more. They each contribute data through diverse mechanisms [Ope22].

Nodes Nodes represent operational entities executing computational tasks. ROS' design exhibits modularity at a detailed level. Therefore, a typical robot control system consists of numerous individual nodes. E.g., the wheel speed is one node, etc. [Ope22].

Master Firstly, the ROS is responsible for registering the names of the graphs components. Secondly, it provides a lookup within the graph. Nodes rely on the master, e.g., to find one another and communicate with one another. Without it, such functionalities would be unavailable [Ope22].

Parameter Server It enables centralized storage of data using key-value pairs. The master integrates the parameter server [Ope22].

Messages A message constitutes a structured data format with typed fields. These messages support standard types (primitive) such as booleans, integers, and floats, along with arrays of these types. Additionally, nesting is possible. Nodes exchange messages to communicate with each other [Ope22].

Topics The previously introduced messages are exchanged via a publish and subscribe mechanism. Nodes that are interested in specific data subscribe to the corresponding topic. Producers of the data publish messages of the data to the topic. The subscribed node then receives the messages and thus the data. The publishing and subscribing is not restricted to a one-to-one relationship. Multiple publishers can publish to one topic, and multiple subscribers can subscribe to a topic. The publishers and subscribers do not know about each other, e.g., how many publishers write to the subscribed topic or how many subscribers read the topic [Ope22].

Services The publish / subscribe mechanism, explained in topics, is not always ideal. If a communication is wanted with request and response, publish / subscribe is not suited. Therefore, there are services provided by nodes. Services are message pairs (request_message, response_message). A node can send a request to another node that makes the service available and waits for the response [Ope22].

Bags Bags provide data storage for messages [Ope22].

In the next section, we describe the RM library used for our prototype.

2.2.2 Runtime Monitoring Library - *rtamt*

As stated by Ničković and Yamaguchi [NY20], *rtamt* is a library with which one can implement online monitors out of “STL” [NY20, p. 1] specifications. Additionally, robustness is provided. It supports future operators as well as past operators in the specifications. It generates the monitors from the specifications automatically [NY20].

Robustness The inputs that *rtamt* should receive are the STL specification and (time, value) pairs. The robustness describes how much the values satisfies or violates the specification. The robustness is a value between [-1,1], where a one describes a full satisfaction and minus one describes a full violation at a specific time [NY20].

Components There are three main components of *rtamt*: the specification, the frontend, and the backend. The first part was already introduced, a “declarative specification language” [NY20, p. 2] is, e.g., STL. The specification is written in such a language. The frontend processes the specification and creates the monitor. To achieve this, it parses the formal specification into a specific data structure (“abstract parse tree” [NY20, p. 2]) and then transforms it into a STL grammar. This is then utilized to create the monitor. The backend consists of the actual algorithm of the monitor that is used for evaluation [NY20].

STL Support The *rtamt* library is compatible with multiple variants of STL. The variant that is relevant to us is “input bounded-future STL (bfSTL)” [NY20, p. 2]. It is characterized by the restrictions regarding multiple operators: “*eventually*, *always* and *until*” [NY20, p. 2]. The restrictions are regarding the intervals concerning these operators. For offline monitors, intervals can be unbounded. For online monitors, they need to be bounded [NY20].

Online Monitors The earlier introduced parsing of the specification and the resulting transformation into a grammar is not usable for online monitors. Hence, the specification is transformed from a “bfSTL formula into an equi-satisfiable past STL formula” [NY20, p. 3]. These formulas only use past-time operators. The evaluation of the formula is delayed until reaching the boundary. This guarantees that all required inputs for calculating the robustness are accessible [NY20].

ROS Support The *rtamt4ros* library represents a ROS integration of *rtamt*. To integrate *rtamt* with ROS, the specification has to be transformed to match ROS. One possibility is to define the variables, its types and the topics of the specification in a *.stl* file. The result is not generalized, it is specification specific. This then associates one variable with one ROS topic [NY20].

In the next section, we describe the object detector for our prototype.

2.2.3 Object Detector - *darknet_ros* with YOLOv3

darknet_ros is an implementation of multiple YOLO [RDGF16] object detection models to use within ROS. YOLO is a neural network used for the detection of objects in real-time. It uses camera images for the detection. The result of the detector can be seen in the images as bounding boxes with the classes attached to the boxes.

YOLOv3 YOLOv3 is one object detection model to use with *darknet_ros*. Its tiny weights are especially interesting because of their best performance. There are updated versions of YOLO, but they are not of use for this thesis.

Dataset "Common Objects in Context (COCO)" [Bje18] is the dataset the YOLOv3 model was trained on. It contains the following class labels: "person, bicycle, car, motorbike, aeroplane, bus, train, truck, boat, traffic light, fire hydrant, stop sign, parking meter, bench, cat, dog, horse, sheep, cow, elephant, bear, zebra, giraffe, backpack, umbrella, handbag, tie, suitcase, frisbee, skis, snowboard, sports ball, kite, baseball bat, baseball glove, skateboard, surfboard, tennis racket, bottle, wine glass, cup, fork, knife, spoon, bowl, banana, apple, sandwich, orange, broccoli, carrot, hot dog, pizza, donut, cake, chair, sofa, pottedplant, bed, diningtable, toilet, tvmonitor, laptop, mouse, remote, keyboard, cell phone, microwave, oven, toaster, sink, refrigerator, book, clock, vase, scissors, teddy bear, hair drier, toothbrush" [Bje18].

Next, we provide information on the object tracker used in our prototype.

2.2.4 Object Tracker - *sort-deepsort-yolov3-ROS*

The provided COCO classes and bounding boxes by *darknet_ros* with YOLOv3 are needed for the object tracker. The "*sort-deepsort-yolov3-ROS*" [ily20] project provides a ROS implementation for both the "Simple Online and Realtime Tracking (SORT)" [BGO+16; ily20] tracker and its extension "Simple Online and Realtime Tracking with a Deep Association Metric (DeepSORT)" [WB18; WBP17]. These implementations are compatible with the YOLOv3 object detector. In the following, we conceptually explain the idea behind SORT and how DeepSORT extends SORT.

SORT SORT is a tracking algorithm. Conceptually, its main purpose is to track objects (give them IDs) and identify them as the same object regarding their bounding boxes, their position, and their movement between frames [BGO+16]. Ideally, the tracker can give the same ID to the same objects over multiple frames [BGO+16]. Exceptions for this are when objects are fully covered by other objects within the frame, or if they leave the frame and return to it [ily20]. Then, they are always considered as new objects with new IDs [ily20].

DeepSORT DeepSORT is an extension of SORT. The extension enables tracking objects even though they were covered for some time [WBP17]. This is not used in this thesis. Therefore, we do not describe this further.

After introducing the relevant background information and foundations, the next chapter gives an overview of related work regarding RM.

3 Related Work

In this chapter, we summarize a topic related conference as well as related work regarding RM. We explain why our work is relevant and how it is different from the work done in previous studies.

3.1 Conference

An important contribution to the field of RM is provided by the "International Conference on Runtime Verification" [Spr23b]. RM in general, is useful for increasing the reliability, safety, and robustness of systems [CL18]. These systems can be hardware systems as well as software systems [CL18]. One of the RM conference's goals is to introduce new lightweight RM techniques [CL18]. As RM can be applied in a broad variety of areas, the conference introduces techniques, e.g., for cyber-physical systems [DN20] and for the security of systems (e.g., intrusion detection [DTCL21]). Besides monitoring such systems, analyzing and guiding the system's behavior at runtime is also of relevance [DN20]. The reviewed conference proceedings are published in a book named "Runtime Verification" [Spr23b]. The book itself is published as part of the "Lecture Notes in Computer Science"[Spr23a] series [Spr23b].

3.2 Taxonomy

In the RM conference of 2018, Falcone et al. [FKRT18] published "a taxonomy for classifying runtime verification tools" [FKRT18, p. 241]. They provide a great contribution to not only specifying a terminology regarding RM and finding similarities between existing tools, but also to provide a classification overview of existing tools. As of 2018, they have classified 20 RM tools using their taxonomy, but without contacting the authors of the tools to do the classification [FKRT18].

In 2021, they provide refinements regarding the terminology of the classification. Additionally, they provide an extension of that paper, increasing the amount of classified tools to 60 instead of 20. They define seven main categories for their classification. They are visualized in Figure 3.1. The seven main categories are connected to the RV node in the middle. For easier understanding, we describe the categories as questions, partly based on each other [FKRT21]:

Specification What do we want to check at runtime [FKRT21]?

Monitor What checks the specification at runtime [FKRT21]?

Deployment How do we implement the monitor? How does the monitor get the needed information [FKRT21]?

Reaction What does the monitor do following the evaluation of the given specification [FKRT18]?

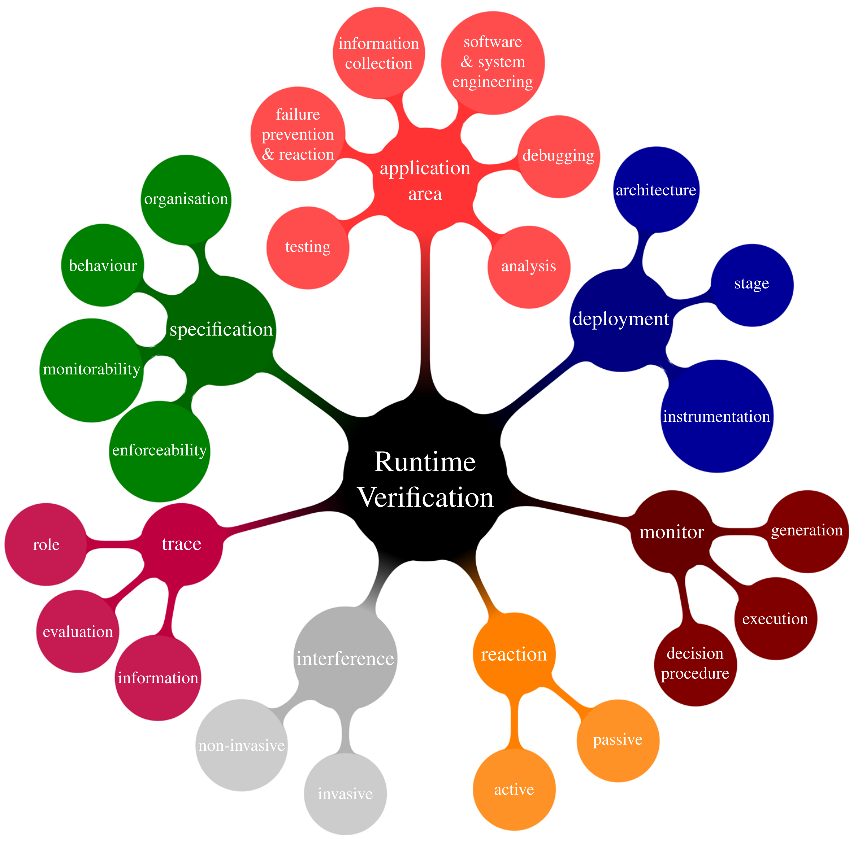


Figure 3.1: Classification overview of the seven main categories and partly subcategories as mind map from [FKRT21]

Trace What information of the system is collected at runtime [FKRT21]?

Interference How much does the tool effect the system [FKRT21]?

The seven main categories may have subcategories that are connected to the main categories. The subcategories are also defined in the papers and the included mind map (see Figure 3.1). For creating their classification overview, Falcone et al. [FKRT21] relied on contacting the authors of the tools to classify their own tools within a survey. The classification results show the classification of the tools while the survey was done. As tools can develop and change over time, their classification may change [FKRT21].

Even though there is the RM conference, finding domain-specific contributions is still challenging. In our case, finding ADAS specific RM techniques is challenging. To the best of our knowledge, there does not exist a literature review regarding RM in the automotive domain. Therefore, there is still great potential in RM in this domain. Additionally, we use the classification taxonomy provided by Falcone et al. [FKRT21] to apply it to our retrieved papers in Chapter 5.

In the next chapter, we provide insights on our study design. We focus on our research questions and our methodology.

4 Study Design

In this chapter, we describe the study design. We describe our research questions for this thesis and summarize the methodology we use to answer each research question.

4.1 Research Questions

RQ1 What are the current state-of-the-art Runtime Monitoring (RM) techniques in the automotive domain, especially for Advanced Driver-Assistance System (ADAS)?

RQ2 Which RM technique is optimal in the context of our prototype?

- **RQ2.1** What are the key factors that had to be considered during selection?

RQ3 How can the chosen RM technique be practically implemented in our prototype?

- **RQ3.1** What are the key considerations and challenges involved in the implementation process?

4.2 Methodology

This section describes the methods we use to answer the previously defined research questions. For answering RQ1, we conduct a "Rapid Review (RR)" [CPS20, p. 357] to collect information on current monitoring techniques for ADAS. We also categorize the found information using a taxonomy provided by Falcone et al. [FKRT21] to help with further evaluation. All of this can be found in Chapter 5. For RQ2, we evaluate the provided hardware and the use case to define the context of our prototype. We then evaluate the possible tools using the context and categorized tools. This can be found in Chapter 6. For RQ3, we implement the use case on the provided hardware. The relevant information can be found in Chapter 7.

5 Rapid Review

With this whole chapter, we answer the following research question, which is first defined in Section 4.1:

RQ1 What are the current state-of-the-art RM techniques in the automotive domain, especially for ADAS?

We give detailed information on the methodology in Section 5.1 and show the results in Section 5.2.

5.1 Methodology

To answer RQ1, we perform a literature review. Due to the limitations in time and reviewers we have for this thesis and the effectiveness of RRs in the decision-making process connected to a practical problem, we choose to do a RR instead of a "Systematic Review" [CPS20, p. 359] as stated by Cartaxo et al. [CPS20].
We visualize the complete workflow of the RR using "Business Process Model and Notation (BPMN)" [Obj23] in Figures 5.1 and 5.2. In the following, we explain the search strategy, the selection procedure, the inclusion and exclusion criteria, the quality appraisal, and the synthesis procedure of the RR [CPS20]. Those parts are also the milestones [Hil23] in Figures 5.1 and 5.2. In the figures, the steps before a milestone are explained in the milestone directly after. E.g., the activities: "Search for Keywords", "Test Queries", "Use Query", and "Duplicate Removal" are explained in the first milestone "Search Strategy". In Figure 5.1, there is the main process visualized. The mentioned sub-processes and call activity are displayed in Figure 5.2. Parallel to the work on the RR, we document the decisions in the next parts (activity: "Create Documentation").

5.1.1 Search Strategy

We start with the activity "Search for Keywords" as displayed in Figure 5.1. We use the title of this thesis to collect keywords without extended prior knowledge on the topic. This results in the following keywords: "ADAS", "Automotive", and "Runtime monitoring". After doing further research in Google Scholar [Goo23] to find synonyms and other related words, we end up with a list of topic-related keywords as visualized by the data object in Figure 5.1. With the keywords, we test multiple combinations in form of queries in scientific databases (activity: "Test Queries") and finally, we proceed with the following search query:

Search Query

```
      Abstract/Title: (
                                 "ADAS"
                              OR "Autonomous driv*"
                              OR "Autonomous vehicle*"
                              OR "Autonomous research vehicle*"
                              OR "Driver assistan*"
                              OR "Driver-assistan*"
                              OR "Automated driv*"
)   AND Title:        (
                                 "Runtime monitoring"
                              OR "Real-time monitoring"
                              OR "Realtime monitoring"
                              OR "Runtime verification"
                              OR "Real-time verification"
                              OR "Realtime verification"
)
```

This is not the exact query for each search engine but a generalized version of it. Even though the following keywords may seem relevant to the topic, we exclude the following keywords because of numerous false positives: “RV”, “ARV”, and “automotive”. We exclude them because the abbreviations also abbreviate words unrelated to the topic and the keyword “automotive” is too general. The query consists of two parts, reflecting our two main focuses: ADAS and RM. We look for RM-related keywords in the title of the papers, as RM should be the primary focus of the results. With ADAS-related keywords, we extend the focus by also searching in the title and the abstract. With explicit exact searches regarding “ADAS”, we try to prevent false positives because of the fuzzy searches of search engines in scientific databases. Normally in RRs, there are only one or few sources used to find papers [CPS20]. As there does not exist a large quantity of literature on our topic, we choose to look into multiple scientific databases. We use the following scientific databases:

- Scopus [Els23]
- IEEE Xplore [Ins23]
- Web of Science [Cla23]
- Lens [Cam23]

In Figure 5.1, we combine the databases as one symbol. On July 17, 2023, we collect 67 relevant entries with the query (Scopus: 19 entries, IEEE Xplore: 9 entries, Web of Science: 11 entries, and Lens: 28 entries; activity: “Use Query”). As we use multiple search engines, we have to filter the results (sub-process: “Filter Results”). The first filtering step is duplicate removal (activity: “Remove Duplicates”). After duplicate removal, there are 36 entries left (see Table 5.1). In the same table, there is an Identifier (ID) for each paper, the title, the publication year, the reference, and the exclusion criterion. We also describe where we found each entry in the scientific databases (see source/s column). There are multiple entries in the column if there are multiple sources for one paper (duplicates). For reproducibility, we mention where exactly we found each paper, even if they are found multiple times. Overall, we get 67 source entries for the 67 initial entries that we found in the different search engines. E.g, we have three source entries for paper ID 1 as we found the paper in IEEE, Scopus, and Web of Science. These are then three out of the 67 entries. In Figure 5.2, we have now reached the “Search Strategy” milestone. We continue with the “Selection Procedure”.

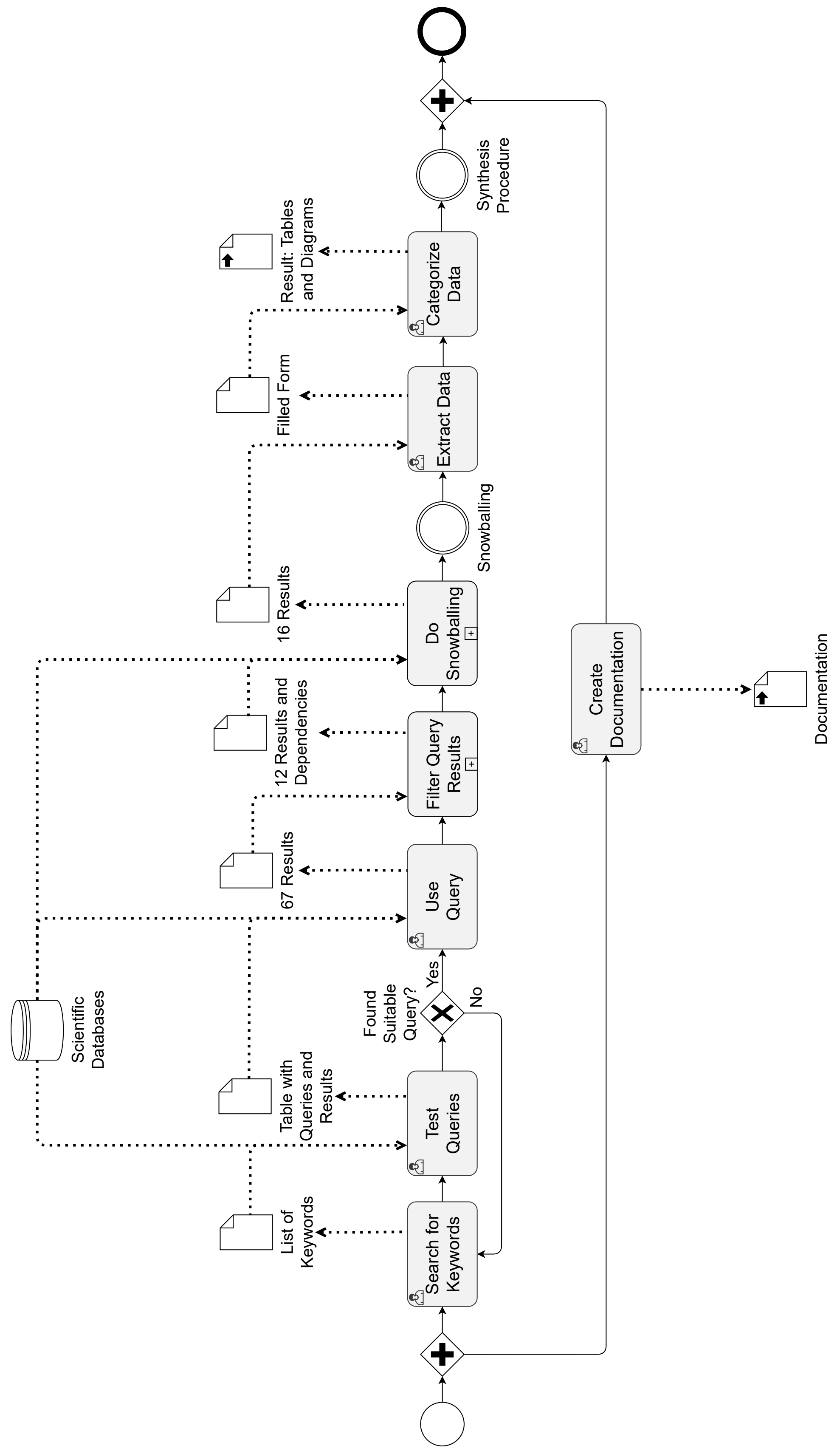


Figure 5.1: RR main-process in BPMN with process milestones [Hil23]

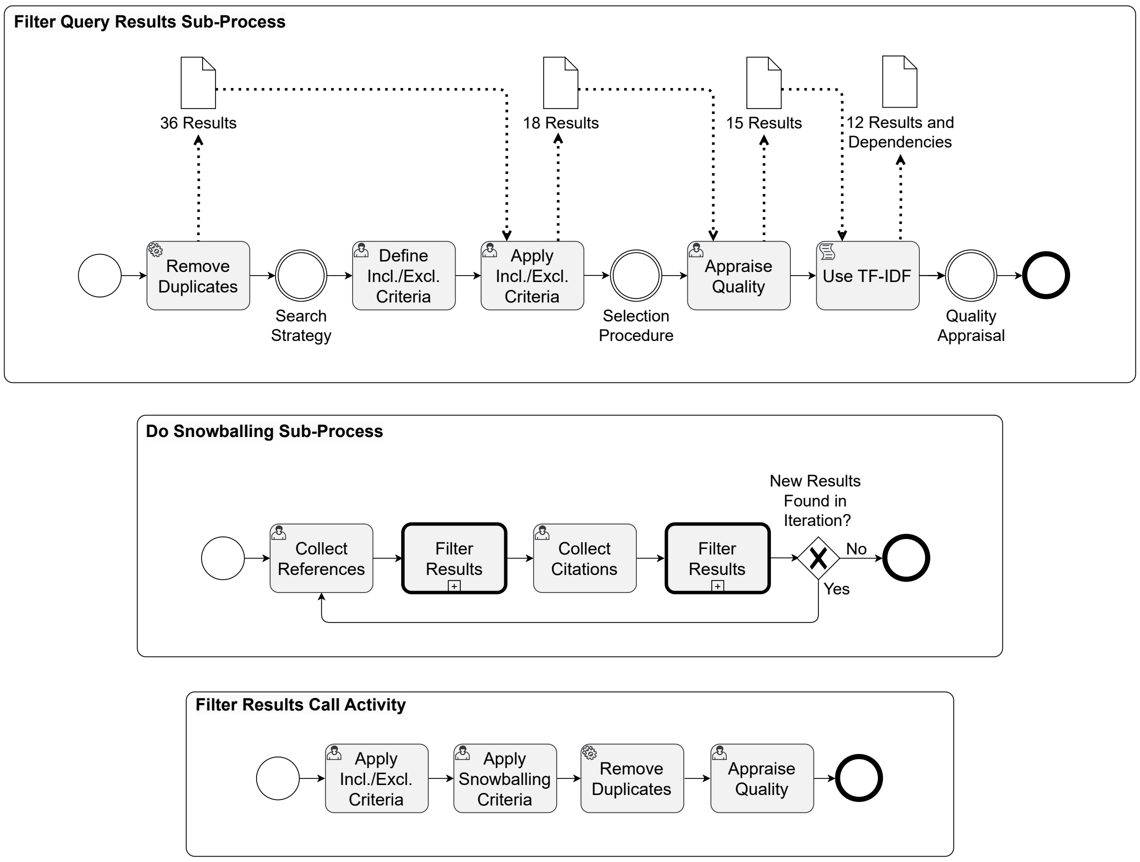

Filter Query Results Sub-Process
36 Results
18 Results
15 Results
12 Results and Dependencies
Remove Duplicates
Search Strategy
Define Incl./Excl. Criteria
Apply Incl./Excl. Criteria
Selection Procedure
Appraise Quality
Use TF-IDF
Quality Appraisal
Do Snowballing Sub-Process
Collect References
Filter Results
Collect Citations
Filter Results
New Results Found in Iteration?
No
Yes
Filter Results Call Activity
Apply Incl./Excl. Criteria
Apply Snowballing Criteria
Remove Duplicates
Appraise Quality

Figure 5.2: RR sub-processes and call activity in BPMN with process milestones [Hil23]

5.1.2 Selection Procedure

With the 36 entries left, we define inclusion and exclusion criteria, as some entries may not be suitable for our research (activity: “Define Incl./Excl. Criteria”). We define and apply the following criteria in that order (activity: “Apply Incl./Excl. Criteria”):

Year All the entries with publication year before 2013 are excluded.

Language All entries in a language that is not English or German are excluded.

Bug The exact search of Lens was not working perfectly at that time, we excluded the false positives caused by that. In more detail, instead of “ADAS”, there were papers found with “Ada”. We exclude them.

Conference We exclude entries if, i.e., they represent a collection of all scientific papers presented at the conference rather than a single work.

In Table 5.1, we list all 36 entries. In the column “Exclusion Criterion”, we mention the criterion that leads to the exclusion. In total, we have 18 papers left. We have now reached the “Selection Procedure” milestone in Figure 5.2. Next, we continue with the “Quality Appraisal”.

ID	Title	Year	Ref.	Source/s	Exclusion Crite-rion
1	“Real-Time Monitoring With Timing Side Information”	2023	[YCP23]	IEEE/S/WoS	Focus
2	“Early Concept Evaluation of a Runtime Monitoring Approach for Safe Automated Driving”	2022	[MČSP22]	IEEE/S	
3	“Pembuatan Sistem Real Time Monitoring Pengukur Oil Layer Pada Vertical Continuous Tank di Pabrik Kelapa Sawit Pekawai Kalimantan Barat”	2022	[Han22]	L	Lang.
4	“Towards Runtime Monitoring of Complex System Requirements for Autonomous Driving Functions”	2022	[GKS+22]	S	
5	“TSI-Aided Real-Time Monitoring of Brownian Motions: A Rate-Latency-Distortion Perspective”	2022	[YCP22]	IEEE/S/WoS	Focus
6	“Computation and Communication Co-Design for Real-Time Monitoring and Control in Multi-Agent Systems”	2021	[TBCM21]	IEEE/L/L/S/ WoS	Focus
7	“Provably-Robust Runtime Monitoring of Neuron Activation Patterns”	2021	[Che21]	IEEE/S/WoS	
8	“Real-time Monitoring of Autonomous Vehicle’s Time Gap Variations: A Bayesian Framework.”	2021	[KA21]	L	Qual.
9	“Trace-Length Independent Runtime Monitoring of Quantitative Policies”	2021	[DTCL21]	IEEE/L/S/WoS	
10	“20th International Conference on Runtime Verification, RV 2020”	2020	[DN20]	S	Conf.
11	“A Smartphone-Based Adaptive Recognition and Real-Time Monitoring System for Human Activities”	2020	[QSA20]	L	Bug
12	“Assuring the Safety of End-to-End Learning-Based Autonomous Driving through Runtime Monitoring”	2020	[GZWR20]	IEEE/L/S/ WoS	
13	“Enabling Real-Time Monitoring of Intrabody Networks Through the Acoustic Discovery Architecture”	2020	[PHCR20]	L	Bug
14	“Formal Runtime Monitoring Approaches for Autonomous Vehicles.”	2020	[SUPR20]	L/S	
15	“Runtime Verification of Autonomous Driving Systems in CARLA”	2020	[ZBK20]	S/WoS	
16	“Uncertainty modeling and runtime verification for autonomous vehicles driving control: A machine learning-based approach”	2020	[ALZ+20]	L/S/WoS	
17	“18th International Conference on Runtime Verification, RV 2018”	2019	[CL18]	S	Conf.
18	“RANCANG BANGUN SISTEM REAL TIME MONITORING GAS BERBAHAYA PADA PETERNAKAN AYAM BROILER BERBASIS INTERNET OF THINGS DAN DATA LOGGER”	2019	[Yuf19]	L	Lang.
19	“Real-Time Verification for Distributed Cyber-Physical Systems”	2019	[TNM+19]	L/L	Qual.
20	“APLIKASI REAL-TIME MONITORING KEHADIRAN KARYAWAN TERINTEGRASI DENGAN FINGERPRINT SYSTEM PADA UNIVERSITAS DEHASEN BENGKULU”	2018	[AR18]	L	Lang.
21	“INVITED: Runtime Monitoring for Safety of Intelligent Vehicles”	2018	[WKLS18]	IEEE/L/L/S/ WoS	
22	“IoT on Heart Arrhythmia Real Time Monitoring”	2018	[AM18]	L	Lang.
23	“NORTH - Non-intrusive observation and Runtime Verification of cyber-physical systems”	2018	[RCL+18]	S	
24	“ISoLA (2) - Assuring the Safety of Advanced Driver Assistance Systems Through a Combination of Simulation and Runtime Monitoring”	2016	[MHR16]	L/ S/ WoS	
25	“Desain Real-Time Monitoring Berbasis Wireless Sensor Network Upaya Mitigasi Bencana Erupsi Gunungapi”	2015	[Pam15]	L	Lang.
26	“Formal Contracts for Runtime Verification Support in the Ada Programming Language”	2015	[MPPP15]	L	Bug
27	“RV - A Case Study on Runtime Monitoring of an Autonomous Research Vehicle (ARV) System”	2015	[KCDK15]	L/ S/ WoS	
28	“Ada-Europe - Towards a Runtime Verification Framework for the Ada Programming Language”	2014	[MPPP14]	L	Bug
29	“Dependable ADAS by combining design time testing and runtime monitoring”	2014	[MRS14]	S	Qual.
30	“Runtime Verification of Linux Kernel Modules Based on Call Interception”	2011	[RS11]	IEEE/ S	Year
31	“Ada-Europe - Runtime verification of java programs for scenario-based specifications”	2006	[XLX+06]	L	Year
32	“Concurrent runtime monitoring of formally specified programs”	1993	[SM93]	L	Year
33	“An application generator for a family of real-time monitor and control systems”	1990	[BY90]	L	Year
34	“Prototype Real-Time Monitor: Ada Code.”	1987	[Sco87a]	L	Year
35	“Prototype Real-Time Monitor. Executive Summary.”	1987	[Sco87b]	L	Year
36	“Prototype Real-Time Monitor: Requirements”	1987	[DLP+87]	L	Year

S = Scopus, L = Lens.org, WoS = Web of Science, Lang. = Language, Conf. = Conference, Qual. = Quality Appraisal

Table 5.1: List of all papers found with the query

5.1.3 Quality Appraisal

To guarantee the quality of the papers, we only use papers that already passed some type of quality control, e.g., in the form of peer reviews. The papers we exclude because of that can be found in Table 5.1 with the exclusion criterion “Quality Appraisal”. In the case of ID 29, we can not find helpful information about whether the symposium peer reviews its papers. Thus, we are pessimistic and exclude the paper. After this, we have 15 papers left. In the “Use TF-IDF” activity, we implement a script which counts the occurrences of all words in our papers (in the 15 results) and sorts the keywords by their “*TF-IDF*” [SW10] scores. By calculating this, we gain insight into the contents of the papers. We ignore stopwords, that do not reflect the contents of the papers. We also fix some formatting issues and ignore punctuations. We do not singularize specific keywords. All the information on the keywords is listed in Listing A.1. We found three papers that are not primarily about RM. Therefore, we exclude them (Exclusion Criterion: “Focus” in Table 5.1). Before the final decision to exclude the papers, we check the contents of the papers to make sure not to exclude relevant papers. We highlight the remaining papers in gray in Table 5.1. We add the common keyword distribution in Section 5.2 (see Table 5.3). In Figure 5.2, we have now reached the “Quality Appraisal” milestone.

5.1.4 Snowballing

On August 28, 2023, we use all remaining papers and Google Scholar [Goo23] for snowballing (sub-process: “Do Snowballing”) [Woh14]. The process of snowballing involves collecting the references (backward snowballing) and filtering the results, as well as collecting the citations (forward snowballing) and again filtering the results [Woh14]. If new results are found, we repeat the process for the new papers until no new results are found [Woh14]. The corresponding activities are called: “Collect References”, “Filter Results”, and “Collect Citations”. We describe the “Filter Results” call activity during backward and forward snowballing further:

1. Filter based on previously defined inclusion and exclusion criteria (activity: “Apply Incl./Excl. Criteria”).
2. Filter based on title or journal. Our RM keywords have to be visible either in the title or in the published journal (activity: “Apply Snowballing Criteria”).
3. Filter based on title or abstract. Our ADAS keywords have to be visible either in the title/ abstract or in the published journal (activity: “Apply Snowballing Criteria”).
4. Remove duplicates (activity: “Remove Duplicates”).
5. Filter based on previously defined quality criterion (activity: “Appraise Quality”). If it is not obvious from the reference directly whether the result is satisfying our quality criterion, we do further research on the results for including or excluding them to the best of our knowledge.

Our snowballing criteria are analogous to our search query, where we have one RM part (step two) and one ADAS part (step three). As we conduct a RR instead of a systematic review, we are rigorous with our criteria compared to Wohlin [Woh14]. Therefore, we focus on the words that are included in the query to decide if a paper needs to be excluded or included in both parts (RM and ADAS) [Woh14]. We want to collect only additional results the query missed. In this case, the paper still matches our defined criteria but is, e.g., published in a different database. After filtering the references and citations based on the previously described procedure, we collect one result from backward snowballing (ID 38) and two results from forward snowballing (ID 37 and ID 40) in the first iteration. In the second iteration, we collect one paper from backward snowballing (ID 39). In the third iteration, we collect no papers. All found papers during snowballing, including in which iteration and in which snowballing step they are found, are visualized in Table 5.2. We list the 16 final papers in Table 5.1 and Table 5.2, all highlighted in gray, as previously described. We recreate and replace the final common keywords table with contents from all remaining papers (see Table 5.3).

ID	Title	Year	Ref.	Snowballing	Iteration
37	"PerceMon: Online Monitoring for Perception Systems"	2021	[BDH+21]	Forward	1
38	"Run-Time Safety Monitoring Framework for AI-Based Systems: Automated Driving Cases"	2019	[OKS19]	Backward	1
39	"Evaluating Perception Systems for Autonomous Vehicles Using Quality Temporal Logic"	2018	[DADF18]	Backward	2
40	"Hierarchical Non-intrusive In-situ Requirements Monitoring for Embedded Systems"	2017	[SL17]	Forward	1

Forward = Forward Snowballing, Backward = Backward Snowballing

Table 5.2: List of added papers during snowballing

5.1.5 Synthesis Procedure

We see the data extraction and categorization of the data as parts of the data synthesis (activities: "Extract Data" and "Categorize Data" in Figure 5.1). We use a form, which is introduced and used by Falcone et al. [FKRT21] to extract and visualize the data. We change the sequences of columns within the form, to make it more useful to us. Additionally, while filling out the form, we already categorize the data (see Table 5.5 and Table 5.6). We had a similar experience as Falcone et al. [FKRT21], while filling out the form, as there are cases where a distinction between categories is not clear. We do all the categorization to the best of our knowledge and leave fields with a "?" where we are not sure, as there is no sufficient information in the papers to categorize it. Falcone et al. [FKRT21] contact the original authors of the papers and tools to categorize their own tools. We do not have the time to contact the authors. Further, we create a dependency graph for the papers (see Figure 5.3).

We have now reached the "Synthesis Procedure" milestone in Figure 5.1. The output of the whole RR are the resulting tables and diagrams (see Table 5.1, Table 5.2, Figure 5.3, Table 5.3, Table 5.4, Table 5.5 and Table 5.6). With that, we reach the end of the RR. In the following section, there is a description of the results.

5.2 Results

With our previously described RR, we collect 16 relevant papers (see gray entries in Table 5.1 and Table 5.2). We try to make the process and the individual steps as reproducible as possible. We now include the findings from our RR.

5.2.1 Common Keywords Within Papers

In Table 5.3, we visualize the occurrences of the 20 most significant keywords by their *TF-IDF* scores in the source papers (see gray entries in Table 5.1 and Table 5.2). In the table, we have the paper IDs listed and the 20 most significant keywords by *TF-IDF*. We mark when a keyword is in a paper with the ○ symbol. We also include in how many papers a keyword is found as the sum column. We also include how many keywords are found in each paper as a sum row. The keywords are sorted by their *TF-IDF* scores with "RM" at in the top row and "bounding box" in the bottom row. The keywords that are in 15 out of the 16 papers are "RV" and "autonomous vehicle". "RV" , compared to "autonomous vehicle" can be found in paper ID 27. In return, "autonomous vehicle" can be found in paper ID 7. The keywords with the least amount of papers are: "traffic situation", "driving style", "acc system", and "bounding box". The keywords "traffic situation" and "bounding box" share source paper ID 39. The keywords "traffic situation" and "acc system" share source paper ID 24. The paper with the highest sum of keywords found is paper ID 15. Throughout the papers, there are at least six keywords found in each paper. The keyword "acc system" stands for "Adaptive Cruise Control (ACC)". Throughout the 20 keywords, we have no keyword that does not match either RM or ADAS aspects. Therefore, we expect that all remaining papers contents are related to our topic.

ID/ Keyword	**2**	**4**	**7**	**9**	**12**	**14**	**15**	**16**	**21**	**23**	**24**	**27**	**37**	**38**	**39**	**40**	**sum**
runtime monitoring	○	○	○	○	○	○	○		○	○	○	○	○	○		○	14
neural network			○		○	○	○	○					○	○	○		8
traffic situation											○				○		2
runtime verification	○	○		○	○	○	○	○	○	○	○	○	○	○	○	○	15
temporal logic		○		○		○	○	○	○	○		○	○		○	○	11
system requirement		○					○		○	○	○	○	○			○	8
autonomous vehicle	○	○	○	○	○	○	○	○	○	○	○		○	○	○	○	15
machine learning			○		○		○	○						○	○		6
autonomous driving	○	○	○	○	○	○	○	○	○		○		○	○	○		13
monitoring algorithm				○		○	○					○		○			5
driving style					○			○									2
runtime monitor	○	○	○	○		○	○		○		○	○					9
anomaly detection				○	○				○					○			4
training data			○		○		○	○						○			5
case study	○			○	○		○	○	○	○	○	○		○	○	○	12
perception system			○				○						○		○		4
data set			○					○			○						3
acc system							○				○						2
test case		○					○				○						3
bounding box													○		○		2
sum	6	8	9	9	10	8	15	10	9	6	11	7	9	10	10	6	

2: [MČSP22], **4**: [Han22], **7**: [Che21], **9**: [DTCL21], **12**: [GZWR20], **14**: [SUPR20], **15**: [ZBK20], **16**: [ALZ+20], **21**: [WKLS18], **23**: [RCL+18], **24**: [MHR16], **27**: [KCDK15], **37**: [BDH+21], **38**: [OKS19], **39**: [DADF18], **40**: [SL17]

Table 5.3: Occurrence of most significant keywords (by *TF-IDF* [SW10]) scores in source papers

5.2.2 Paper Dependencies

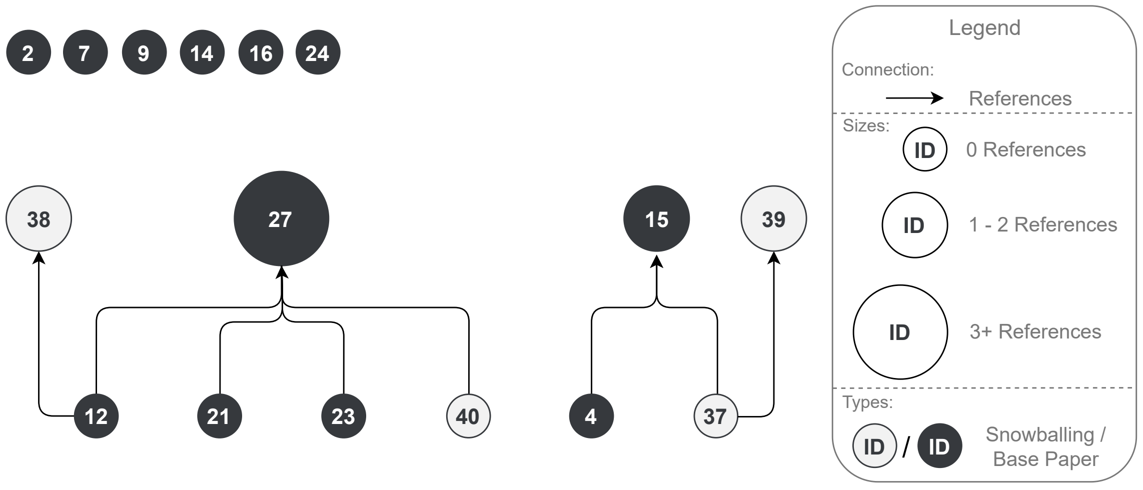
We analyze the dependencies between the papers to gain more information on the relevance of the individual papers. We visualize the dependencies in the graph in Figure 5.3. In the graph, the papers are visualized as nodes. With a number (“ID”) in each node, we specify which paper is represented by the node (by ID). The numbers and further information on the papers can be found in Table 5.1 and Table 5.2. In the graph, there are two types of nodes, three sizes of nodes, and one connection type. There are two types of nodes, base nodes and snowballing nodes. Each type represents when a paper was found, either it is a base paper or it was found during snowballing. The three sizes of nodes are: small, medium, and large. Each size represents the relevance of the paper for the other 15 papers. We describe this further:

Small node A paper is represented with a small node if none of the 15 other papers reference it. Therefore, the small node has no incoming edges.

Medium node A paper is represented with a medium node if one or two of the 15 other papers reference it. Therefore, it has one or two incoming edges.

Large node A paper is represented with a large node if three or more of the 15 other papers reference it. Therefore, it has three or more incoming edges.

The edges visualize the references. We have four papers referencing the paper with ID 27 (ID 12, ID 21, ID 23, and ID 40). The paper with ID 12 also references the paper with ID 38. We also have two papers referencing ID 15 (ID 4 and ID 37). The paper with ID 37 also references ID 39. Therefore, we have three medium nodes (ID 15, ID 38, and ID 39) and one large node (ID 27). The remaining nodes are small nodes. We expect that the referenced papers are foundational work. Thus, we think that the connected papers may be similar regarding their topics.



2: [MČSP22], **4**: [Han22], **7**: [Che21], **9**: [DTCL21], **12**: [GZWR20], **14**: [SUPR20], **15**: [ZBK20], **16**: [ALZ+20], **21**: [WKLS18], **23**: [RCL+18], **24**: [MHR16], **27**: [KCDK15], **37**: [BDH+21], **38**: [OKS19], **39**: [DADF18], **40**: [SL17]

Figure 5.3: Dependency graph to visualize the dependencies between the papers

5.2.3 Technique Classification

In a RR, there is the synthesis procedure. It combines the data extraction and categorization of the retrieved data. We add these results in Table 5.5 and Table 5.6. We use abbreviations from now on, which we list in Table 5.4. In Table 5.4, we have two columns. We call the left column “Column” and the right column“Values”. Entries in the “Column” column are names of the columns and sub-columns in, e.g., Table 5.5. The “Values” column describes all the values the columns and sub-columns can have. The main columns are “Specification”, “Monitor”, “Deployment”, “Interference”, “Reaction”, “Trace”, and “ADAS application”. In the Table 5.4, there exists one extra column, which is called “General”. It defines generally usable abbreviations throughout multiple columns. As we use the taxonomy by Falcone et al. [FKRT21], most of the abbreviations are by them. We extend the abbreviations, e.g., by adding the general value “x supported”, which is then explained in the table. For further explanation on the columns and sub-columns, see Chapter 3 or the paper by Falcone et al. [FKRT21].

After introducing the abbreviations, we display our extraction and categorization of the data in Table 5.5 and Table 5.6. We now explain the alterations we make to the form by Falcone et al. [FKRT21]. In Table 5.5, we make the following alterations: Contrary to Falcone et al. [FKRT21], we add the “organization” column to the table, even though we also have only the same value in the column. We keep the “behavior” sub-column to distinguish it from the “organization” column, as we also keep that column. We separate the “output” column into “information” and “frequency”. We add the “tool” column to the “decision-procedure”, instead of a separate column for the tool. We divide the “stage” column into an “offline” and an “online” column. We combine the “synchronisation” [FKRT21] column and “placement” [FKRT21] column that belong to the “online” column. We combine those columns into “online” to make the connection more clear.

In Table 5.6, we add the “formalism & fragment”, merged from “Specification formalism” [FKRT21] and “Supported fragment” [FKRT21]. We also extend the “application area” [FKRT21] by adding the “tool” column and by adding the “focus” column, “environment” column, the “communication” column, and the “task” column. We have the following questions in mind while filling out these columns:

Focus What is the main focus of the work?

Environment In what environment do they use the tool in?

Communication How does communication happen between components?

Task What is the purpose of the work?

Most columns from Table 5.6 contain further textual information, besides just the abbreviations. The gray marked entries in both tables are open-source tools. We gather 16 relevant papers and collect 16 approaches for RM that can be used in the automotive domain. We categorize the data and visualize it using the form tables. We continue with an analysis of our extracted and categorized data in the next section.

Column	**Values**
Specification	
organization	c = central, d = decentral
implicit	ms = memory safety,
	c = concurrency,
	sec = security
data	p = propositional, s = simple parametric, c = complex parametric
output information	v = verdict, r = robustness, q = quality
output frequency	sng(_) = a single _, seq(_) = a sequence of _
logical time	tot = total order, par = partial order
physical time	di = discrete, de = dense, none = no time
modality	f = future and current, p = past and current, c = current
paradigm	d = declarative, o = operational
Monitor	
generation	e = explicit, i = implicit
execution	i = interpreted, d = direct
properties of the decision procedure	s = soundness, c = completeness, i = impartiality, a = anticipation
Deployment	
stage	on = online, off = offline
synchronisation	sync = synchronous, async = asynchronous
architecture	c = centralised, d = decentralised
placement	out = outline, in = inline,
instrumentation	sw = software
Interference	
Interference	in = invasive, ni = non-invasive
Reaction	
active	ex = exception, r = recovery, ro = rollback, en = enforcement
passive	so = specification output, e = explanations, st = statistics
Trace	
information	e = events, s = states, i/o = input and output, si = signal
sampling	et = event-triggered, tt = time-triggered
evaluation	p = points, i = intervals
precision	p = precise, i = imprecise
model	f = finite trace model, i = infinite trace model
ADAS application	
focus	pv = property verification, fp = failure prevention & reaction,
	td = testing and debugging
General	
	all = all features supported, none = no features supported
	na = not applicable, ? = insufficient information
	x supported = x not applied but supported by the tool

Table 5.4: Classification abbreviations: The table is first introduced by Falcone et al. [FKRT18; FKRT21], as well as extended by us.

ID	Specification									Monitor					Deployment				Interference	Reaction		Trace				
	organization	behavior								decision procedure			generation	execution	stage		architecture	instrumentation		active	passive	information	sampling	evaluation	precision	model
		implicit	explicit							realisation	tool	properties			offline	online										
			data	output		time		modality	paradigm																	
				info.	freq.	log.	phys.																			
2	c	none	c	v	sng	?	?	f	d	analytical	SDE-V	c	i	d	no	?,sync	c	none	in	r	so	s	et	p	p	i
4	c	none	c	v	sng	?	?	f,p	d	analytical	?	c,s,i,a	e	i	no	?,?	c	none	in	none	so,st	s	?	p	p	i
7	c	none	p	v	sng	none	none	c	o	symbolic reasoning	na	s	i	d	no	inline,sync	c	na	in	none	so	i/o	et	p	i	f
9	c	none	p	v	seq	?	di	p	d	dynamic programming	?	c,s	i	i	no	outline,async	c	LogicDroid: sw, ROS: none	in,ni	en	so	s,e	et	i	p	i
12	c	none	s	v	sng	?	?	c	d	analytical	Dependability Cage	?	?	?	no	?,?	c	none	in	r	so	s	?	p	p	i
14	c	none	s	v	sng	?	di	c	o	automata-based	easy-rte	c,s	e	d	no	outline,sync	c	none	in	en	so	e	et	p	p	i
15	c	none	s	v,r	seq	?	di,de supported	f,p	d	analytical	rtamt	?	e	i	supported	outline,sync	c	none	ni	none	so,st	e	tt,et supported	i	p	i
16	c	none	s	r	seq	?	di	c	d	automata-based	UPPAAL-SMC	c,s	e	i	none	?,sync	c	none	in	en	so	i/o	tt	p	p	i
21	c	none	s	v,r	seq	tot	de	f	d	analytical	Breach	?	i	i	yes	?,sync	c	none	in	none	so,e	e,s	et	i	p	i
23	c	c, ms supported sec supported	c	v	seq	?	?	c	d	?	compatible w/ Cheddar	?	?	?	no	inline,?	c	none	ni	none	so	e	et,tt	p	i	f,i
24	c	none	c	v	sng	?	?	c	d	analytical	?	?	e	d	no	outline,?	c	none	ni	none	so	i/o	et	p	p	i
27	c	none	p	v	seq	tot	di	f,p	d	dynamic programming	AgMon/EgMon	s,a	i	i	no	outline,sync	c	none	ni	none	so	e	tt	p	p	i
37	c	none	c	v,r	sng	?	?	f,p	d	analytical	PerceMon	c,s	i	i	yes	outline,?	c	none	ni	none	so	i/o	et	p	p	i
38	c	none	s	v	sng	?	di	c	d	analytical	SMF	**?**	i	i	no	?,?	c	none	in	r	so	i/o	tt	p	p	i
39	c	none	c	v,q	sng	?	?	f	d	analytical	Persephone based on S-TaLiRo	?	i	i	yes	supported	c	none	ni	none	so	i/o	et	p	p	i
40	c	none	p	v	sng	?	?	c	o	?	NIRM hardware monitor	?	e	d	none	outline,?	c	none	ni	none	so,st	si,e	et	p	p	i

2: [MČSP22], **4**: [Han22], **7**: [Che21], **9**: [DTCL21], **12**: [GZWR20], **14**: [SUPR20], **15**: [ZBK20], **16**: [ALZ+20], **21**: [WKLS18], **23**: [RCL+18], **24**: [MHR16], **27**: [KCDK15], **37**: [BDH+21], **38**: [OKS19], **39**: [DADF18], **40**: [SL17]

Table 5.5: Classification - Part 1: Gray highlighted entries use open-source tools

ID	Specification	ADAS application			
	formalism & fragment	focus	environment	communication	task
2	SDE-V	pv	IPG CarMaker, TTTech's RazorMotion Automotive Prototype ECU, PC	?	check ADAS output trajectory to avoid collisions and avoid entering undrivable areas
4	Visual TSCs, SVR, TER	pv	simulation in CARLA, also supports real-world vehicles	CARLA, ROS	pass-by maneuver monitoring
7	abstractions for neuron activation patterns in DNNs	pv	ARV	?	detect abnormal camera images in physical laboratory setting (race track)
9	MTLcnt: extension of ptMTL (past-time MTL variant)	pv	LogicDroid, autonomous vehicle simulation platform (based on ROS)	USB, CAN	malware detection, intrusion detection
12	direct implementation	pv	ARV	?	ensure collision-free autonomous driving through parcour
14	FA or VDTA	pv	ARV with Raspberry Pi as master and Arduino Uno as slave for steering	?	ensure correct steering to keep vehicle within lane
15	STL	pv	simulation in CARLA	CARLA	find ACC parameters and continuosly check ACC performance requirements
16	informal visual stohChart(p) requirements translated to formal PCTL properties	pv	ARV, experiments with simulated scenario	?	ensure safety of LCA decisions
21	Parametric STL, online monitoring: restricted	pv,td	Unity, Simulink, MATLAB	LAN, MQTT	CPMS,FPS
23	TL	pv	FPGA hardware monitor with RTEMS target	BUS	online job scheduling errors detection, planned: experiments with CPS
24	ATS, mapping functions	pv	trained in simulations and then applied to real-world industrial prototype $\rightarrow$ loop	?	monitor safe operation of LCA
27	MTL, bounded future modalities	pv	ARV	USB, CAN	check safety requirements (mode transitions, heartbeats)
37	TQTL, STQL	pv	simulation in CARLA + ROS wrapper for CARLA, OpenSCENARIO	CARLA, ROS	check consistency of detections and smoothness of bounding box trajectories for all objects in dangerous driving situations
38	Safety Profile	pv,fp	ARV	?	monitor safety of ADAS for lane and obstacle detection
39	TQTL	pv	monitor applied on KITTI image benchmark dataset for autonomous driving scenarios	na	check consistency of detections for all objects in subsequent image frames
40	UML sequence diagrams translated to HRMGs	pv	on-chip NIRM hardware implemented on FPGA with Microblaze processor	BUS	detect timing, dependency, synchronization and sensor failures in multiple autonomous vehicle driving scenarios

2: [MČSP22], **4**: [Han22], **7**: [Che21], **9**: [DTCL21], **12**: [GZWR20], **14**: [SUPR20], **15**: [ZBK20], **16**: [ALZ+20], **21**: [WKLS18], **23**: [RCL+18], **24**: [MHR16], **27**: [KCDK15], **37**: [BDH+21], **38**: [OKS19], **39**: [DADF18], **40**: [SL17]

Table 5.6: Classification - Part 2: Gray highlighted entries use open-source tools

5.2.4 Technique Classification Analysis

Here, we describe how we analyze the retrieved techniques. First, we provide an overview on the distribution of the classified taxonomy values. Then, we compare the taxonomy classification values of each paper with all other papers to retrieve pairwise similarity scores.

Contents of the Papers

As we have textual information in form of papers, direct analysis is not possible. We show the extraction and categorization results of the data in the previous section. Generally, the contents of the 16 papers are all related to the topic of the thesis, as expected due to no outliers in Table 5.3. We now analyze the classification results. We implement a script to analyze the contents of our data where possible. We omit the following columns: “formalism & fragment”, “environment”, and “task”. We visualize our results in Table 5.7 and Table 5.8. We left the main categories/ columns to match the structure of Table 5.5 and Table 5.6. The smaller tables, within Table 5.7 and Table 5.8, are the sub-columns of the classification results. Table 5.7 contains the columns: “Specification”, “Monitor”, and “Deployment”. Table 5.8 contains the columns: “Interference”, “Reaction”, “Trace”, and “ADAS application”. With the script, we count how often we got which result in the sub-column, e.g., in the “specification” column, we have “c” as a result 100% of times throughout the 16 papers. If a column has multiple values as an entry, we split them at the comma and count them as separate entries. E.g., the paper with ID 23 has “c,ms supported,sec supported” as an entry in Table 5.5. Besides the “organization” column, the “architecture” column also has “c” as entry in 100% of the entries. As we rounded the values, sometimes they do not add up to 100%. We have several columns where we have insufficient information in at least $\frac{1}{3}$ of cases. The columns are: “logical” (logical time) with approximately 81%, “physical” (physical time) with approximately 47%, “properties” with approximately 33%, “online” with approximately 42%, and “communication” with approximately 33%. The other fields are mostly assignable. We also have multiple columns where we often have “none” as entry, e.g., “implicit” with approximately 83%, “instrumentation” with approximately 82%, and “active” with 62%. In the “tool” column, there are always unique answers that are then evenly distributed with 6.25%, except where we have no sufficient information to categorize them. In the “interference” column, almost half of the results (approximately 47%) has “ni” as entry and the remainder has approximately 53%.

Similarities Between Papers

In Table 5.9, we evaluate how similar pairs (ID 1, ID 2) of papers are in our classification. We always choose two papers and compare the row entries of the papers from Table 5.5 and Table 5.6 with each other. The “similarity” column contains the similarity values. They are rounded for better readability. The column “ID 1” contains the ID of the first paper of the pair, and the column “ID 2” contains the second paper of the pair. E.g., we compare the papers with ID 2 and ID 15. They have a similarity of around 32%. To evaluate the similarity, we always compare the values of the same column with each other. E.g., ID 2 and ID 15 both have “d” as “paradigm” entry, which increases their similarity. Columns, where the entries are not the same, decrease the papers’ similarity.

In total, we get 120 pairs, which we separate into three smaller tables for readability in Table 5.9. The lowest similarity has the pair (7,15) with around 16%. The pairs (12,38) and (37,39) have the highest similarities with 71% each. The gray highlighted entries are entries that have a connection in Figure 5.3 through a reference. We observe that the lowest similarity value is not one of the gray highlighted entries, but both of the highest values are highlighted entries, as they have a connection in Figure 5.3. As a reminder, ID 12 references ID 38 and ID 37 references ID 39. All the gray highlighted entries have a similarity of at least 32%. The highest similarity in the non-highlighted entries is 65%. There are five pairs with this value: (2,12), (2,24), (24,37), (24,39), and (24,40).

Specification

organization	**%**
c	100.00

implicit	**%**
none	83.33
c	5.56
sec supported	5.56
ms supported	5.56

data	**%**
c	37.50
s	37.50
p	25.00

information	**%**
v	75.00
r	20.00
q	5.00

frequency	**%**
sng	62.50
seq	37.50

logical	**%**
?	81.25
tot	12.50
none	6.25

physical	**%**
?	47.06
di	35.29
none	5.88
de supported	5.88
de	5.88

modality	**%**
c	40.00
f	35.00
p	25.00

paradigm	**%**
d	81.25
o	18.75

Monitor

realisation	**%**
analytical	56.25
dynamic programming	12.50
automata-based	12.50
?	12.50
symbolic reasoning	6.25

tool	**%**
?	18.75
SDE-V	6.25
na	6.25
Dependability Cage	6.25
easy-rte	6.25
rtamt	6.25
UPPAAL-SMC	6.25
Breach	6.25
compatible w/ Cheddar	6.25
AgMon/EgMon	6.25
PerceMon	6.25
SMF	6.25
Persephone based on S-TaLiRo	6.25
NIRM hardware monitor	6.25

properties	**%**
?	33.33
s	29.17
c	25.00
a	8.33
i	4.17

generation	**%**
i	50.00
e	37.50
?	12.50

execution	**%**
i	56.25
d	31.25
?	12.50

Deployment

offline	**%**
no	62.50
yes	18.75
none	12.50
supported	6.25

online	**%**
?	41.94
sync	22.58
outline	22.58
inline	6.45
async	3.23
supported	3.23

architecture	**%**
c	100.00

instrumentation	**%**
none	82.35
na	5.88
LogicDroid: sw	5.88
ROS: none	5.88

Table 5.7: Analysis of the classification - Part 1

Interference

interference	%
in	52.94
ni	47.06

Reaction

active	%
none	62.50
r	18.75
en	18.75

passive	%
so	80.00
st	15.00
e	5.00

Trace

trace information	%
e	36.84
i/o	31.58
s	26.32
si	5.26

sampling	%
et	55.56
tt	27.78
?	11.11
et supported	5.56

evaluation	%
p	81.25
i	18.75

precision	%
p	87.50
i	12.50

model	%
i	88.24
f	11.76

ADAS application

focus	%
pv	88.89
td	5.56
fp	5.56

communication	%
?	33.33
CARLA	14.29
ROS	9.52
USB	9.52
CAN	9.52
BUS	9.52
LAN	4.76
MQTT	4.76
na	4.76

Table 5.8: Analysis of the classification - Part 2

The difference of 6% between 71% and 65% is a difference of about 1.9 entries. The two highest similarities can be explained due to the fact that one paper is based on the other. The one paper is then a foundation for the other paper. Which means that the foundational paper is further extended, as the other paper may have worked on the future work aspects of the foundational paper. The smaller similarities can be explained due to extending the foundational paper for entirely new aspects not covered in the foundational work. We expected all connected papers to have a high similarity, which is not the case. For instance, even papers that are not connected through a reference have a higher similarity than those papers that are connected. But the highest similarities are still within the connected papers.

The information provided in the tables Table 5.5 and Table 5.6 summarize the available tools and techniques since 2013 in the automotive domain, especially for ADAS. Therefore, they represent the answer to our research question RQ1. Using these results, we now proceed with deciding which tool to use for the prototype in the next chapter.

ID 1	ID 2	similarity
2	4	58%
2	7	45%
2	9	42%
2	12	65%
2	14	55%
2	15	32%
2	16	45%
2	21	42%
2	23	39%
2	24	65%
2	27	42%
2	37	55%
2	38	58%
2	39	58%
2	40	45%
4	7	32%
4	9	39%
4	12	61%
4	14	45%
4	15	48%
4	16	42%
4	21	35%
4	23	39%
4	24	61%
4	27	45%
4	37	58%
4	38	52%
4	39	52%
4	40	48%
7	9	32%
7	12	42%
7	14	48%
7	15	16%
7	16	32%
7	21	23%
7	23	32%
7	24	48%
7	27	39%
7	37	35%
7	38	45%
7	39	35%
7	40	42%
9	12	35%
9	14	45%
9	15	35%
9	16	45%
9	21	35%
9	23	29%
9	24	42%
9	27	52%
9	37	42%
9	38	42%
9	39	39%
9	40	32%
12	14	55%
12	15	39%
12	16	48%
12	21	35%
12	23	48%
12	24	61%
12	27	42%
12	37	45%
12	38	71%
12	39	48%
12	40	45%
14	15	39%
14	16	61%
14	21	29%
14	23	35%
14	24	58%
14	27	45%
14	37	42%
14	38	55%
14	39	39%
14	40	52%
15	16	42%
15	21	48%
15	23	35%
15	24	45%
15	27	48%
15	37	48%
15	38	39%
15	39	45%
15	40	42%
16	21	39%
16	23	32%
16	24	48%
16	27	45%
16	37	45%
16	38	58%
16	39	42%
16	40	39%
21	23	23%
21	24	35%
21	27	39%
21	37	45%
21	38	42%
21	39	48%
21	40	29%
23	24	52%
23	27	42%
23	37	39%
23	38	35%
23	39	42%
23	40	45%
24	27	45%
24	37	65%
24	38	58%
24	39	65%
24	40	65%
27	37	48%
27	38	52%
27	39	45%
27	40	39%
37	38	48%
37	39	71%
37	40	48%
38	39	52%
38	40	39%
39	40	48%

2: [MČSP22], **4**: [Han22], **7**: [Che21], **9**: [DTCL21], **12**: [GZWR20], **14**: [SUPR20], **15**: [ZBK20], **16**: [ALZ+20], **21**: [WKLS18], **23**: [RCL+18], **24**: [MHR16], **27**: [KCDK15], **37**: [BDH+21], **38**: [OKS19], **39**: [DADF18], **40**: [SL17]

Table 5.9: Similarities of the classification in % from Table 5.5 and Table 5.6: Gray highlighted entries are connected in Figure 5.3

6 Prototype Technique Selection

With this whole chapter, we answer the following research questions, which are first defined in Chapter 4:

RQ2 Which RM technique is optimal in the context of our prototype?

> **RQ2.1** What are the key factors that had to be considered during selection?

To answer RQ2.1, we collect information on the provided hardware in Section 6.1 and information on the prototype's use case in Section 6.2. The hardware requirements and the use case are relevant factors for tool selection. With knowledge on them, we have then defined a context and requirements for the prototype. With this, we then filter the 16 tools on more detailed factors and provide one optimal tool for the prototype in Section 6.3 to answer RQ2.

6.1 Hardware - *Mecabot TX*

For our exploration of RM in the automotive domain, we have the opportunity to work with an ARV, the "*Mecabot TX*" [Rob23a]. The following information on the *Mecabot TX* is extracted from the *Roboworks* [Rob23a] online shop:

Applications "Autonomous driving[,] autonomous mobile robot[,] SLAM[,] navigation[,] mobile manipulation[,] multi robot system[,] human robot interaction[,] mobile computer vision[,] edge computing over 4G/5G" [Rob23a]

Dimensions "Length: 41.05 cm[,] Width: 40.7 cm[,] Height: 15.3 cm" [Rob23a]

Included Components The components of the robot can be divided into multiple categories, which are given below.

- **Hardware** "1x ROS controller Jetson TX1[,] 1x chassis battery[,] 1x battery charger" [Rob23a]
- **Sensors** "Orbbec depth camera[,] Leishen LSLiDAR" [Rob23a]
- **Operating System and Software** "onboard Ubuntu [18.04,] ROS1 [Melodic] [and] STM32 driver[,] [...] remote control app[,] drivers and software installers" [Rob23a]
- **ROS Packages and Source Codes** "STM32 chassis source codes" [Rob23a]
- **Actuators** "omnidirectional mecanum wheels with 100 mm diameter[,] 35W servo power" [Rob23a]

We investigate this, as it is relevant to know what hardware we have available for our prototype. The tool selection may differ depending on the hardware the robot provides. Next, we define the use case for our prototype.

6.2 Prototype Use Case

In this section, we present our prototypical use case to demonstrate the usage of runtime monitors for ADAS systems. The prototype is deployed onto a real-world ARV called *Mecabot TX*. The ARV is further described in Section 6.1. The aim of our AEB ADAS system is to perform AEB in case of an obstacle appearing in front of the ARV. In detail, the system is based on RGB camera images that are used to check for pedestrians, cars, etc. in a certain sub area of the image (called the AEB danger zone). Whenever an obstacle of a particular COCO class (e.g., person, car, ...) is detected within this AEB danger zone, the AEB system produces an emergency braking signal to stop all wheels of the ARV immediately. This corresponds to a full emergency braking maneuver. If there is no such obstacle within the AEB danger zone, the AEB outputs a safe driving signal and sets a constant positive speed on all wheels.

The danger zone and example COCO classes are visualized in Figure 6.1. Obstacles with a green border are safe obstacles and obstacles with a red border are unsafe. We have two examples in Figure 6.1, on the left are safe obstacles in the danger zone and on the right are unsafe obstacles. In Figure 6.1, the safe obstacles are: cars outside the danger zone, a person outside the danger zone and obstacles that are within the danger zone, but the overlap is under a threshold. Additionally, we consider COCO classes like donuts as safe, even though they are fully within the danger zone, as we do not consider them as a safety threat. On the right image in Figure 6.1, we visualize the unsafe obstacles, where the obstacles are overlapping with the danger zone (overlap of at least the chosen threshold).

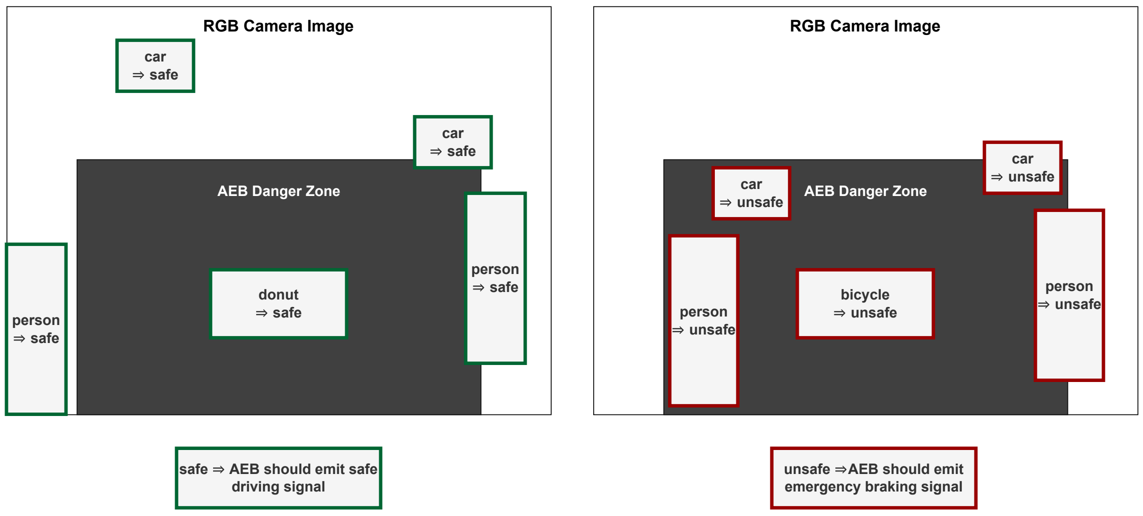


Figure 6.1: AEB danger zone illustration

The AEB is intended to work in a dynamic environment where obstacles can appear in front of the ARV anytime. Therefore, the goal of the AEB system is to prevent the ARV from hitting any unsafe obstacle in front. The AEB system works solely perception-based, e.g., by analyzing RGB camera images. The idea behind this perception-based AEB ADAS is to simulate a complex perception- & AI-based system for AD functions, as, e.g., used in real-world autonomous vehicles by different companies [MMGP19]. These perception- and AI-based systems are known to be error-prone, as already discussed in Chapter 2. Therefore, we additionally set up runtime monitors to monitor the decisions of the AEB ADAS at runtime regarding several Safety Requirements (SRs). We present the SRs for the RMs:

SR1 The AEB system should produce a signal (safe driving or emergency braking) each [50,150] ms (real-time samples).

SR2 If the current distance of the ARV to the nearest object in front is less than the current safety distance, then the AEB ADAS should emit an emergency braking signal within the next 500 ms. In example, when an emergency brake is required due to immediate danger, the AEB ADAS must produce an emergency braking signal quickly (no false negatives).

SR3 If the AEB ADAS emits an emergency braking signal, then the current distance to the nearest object in front must be less than the current safety distance. In other words, when the AEB ADAS produces an emergency braking signal, there must be an immediate danger in front of the ARV (no false positives).

SR4 Whenever the AEB system emits an emergency braking signal, then there may not occur a safe driving signal within the next 2000 ms before the ARV reaches zero velocity (no stopping of emergency braking maneuver).

6.2.1 Data Flow Visualization

Based on the given hardware and SRs, we visualize a possible data flow (connection arrows) in Figure 6.2. Details may change in the actual implementation. The sensors and outputs are parts of the ARV. We have the sensor components on the left (Sensors). As we know from Section 6.1, we have a RGB camera and a LiDAR equipped. Both sensors collect environmental data. The camera and LiDAR data are topics the system within the ARV is subscribed to. How exactly the data is processed is currently a black box system (marked with a *?*). We do know that our runtime monitors and the AEB will run on this system on the ARV, as the comment in Figure 6.2 describes.

After the black box system, we have the outputs the system publishes. We think that the wheel speed, the AEB signal, the monitor result and the violations report may be relevant outputs. The wheel speed, the AEB signal, and the monitor result will also be topics.

The wheel speed likely controls the actual wheel speed of the ARV. This makes emergency braking and driving possible. The AEB signal is the decision of the AEB whether to stop or to drive. The monitor result is the evaluation of the perception data by the monitor. The violation report will be a file with details about the violations of the SRs. Further details on the actual implementation are provided in Chapter 7.

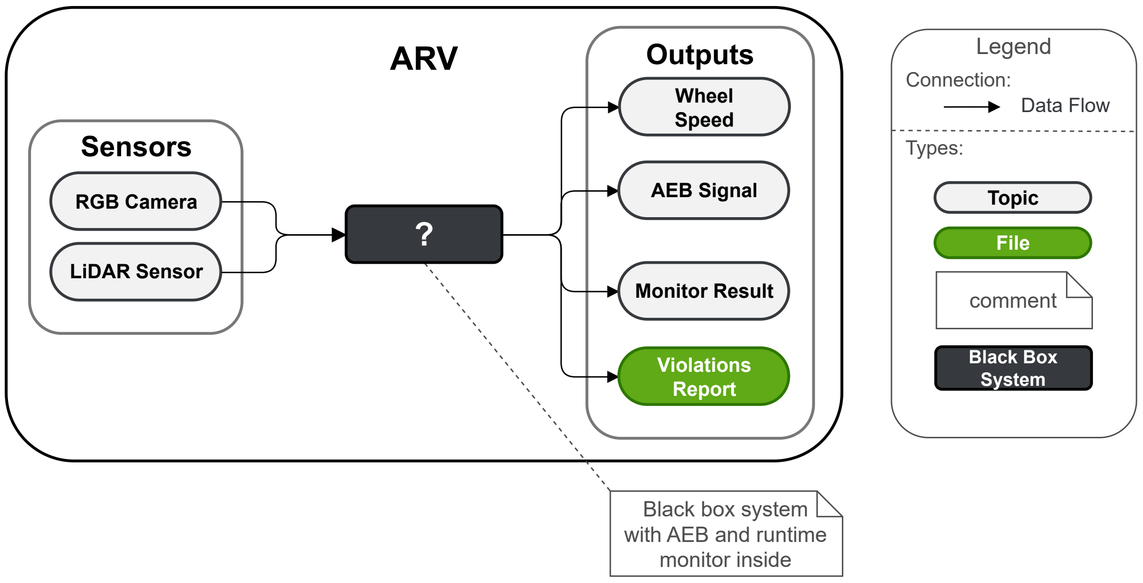


Figure 6.2: Prototype data flow with a black box system

6.3 Selection Procedure

For choosing the tool for our prototype, we evaluate relevant aspects from the form by Falcone et al. [FKRT21] considering the available hardware and use case. We evaluate the aspects and values shown in Table 6.1.

The table is similar to the abbreviations table (see Table 5.4). We exclude columns, where the actual entry is not relevant for our use case. We include the reasoning for the relevant aspects and their needed values in the following:

Specification We need verdicts for properties to check. Thus, we need at least *v* as value in explicit output information. We need some notion of time for our SRs. Therefore, we want neither *none* nor *?* as values in explicit physical time. Our SRs also require future operators. Thus, we need at least *f* (current and future) for the explicit modality.

Monitor We need specified open-source tooling.

Deployment We need the monitor to not be inline with the system to monitor (e.g., outline with separate ROS node). Thus, we need at least *outline* as value in stage online. We need a central deployment architecture (deployed as one centralized ROS node). Therefore, we need *c* as value in architecture. We would rather not use direct instrumentation but use pub/sub with ROS topics. Therefore, we need *none* as value in instrumentation.

Reaction We need specification outputs (e.g., verdicts of properties). Thus, we need at least *so* as value in passive.

Trace We need the monitor to use a precise trace model (assumes perfect precision of given trace data). Therefore, we need *p* as value in precision. We need the monitor to support retrieving trace information endlessly. Therefore, we need *i* as a value in the model.

ADAS application We need property violation checking. Therefore, we need *pv* as value in focus. ROS compatibility is ideal.

Column	**Values**
Specification	
explicit output information	at least *v*
explicit physical time	not *none*, not *?*
explicit modality	at least *f*
Monitor	
tool of the decision procedure	specified and open-source
Deployment	
stage online	at least *outline*
architecture	*c*
instrumentation	*none*
Reaction	
passive	at least *so*
Trace	
precision	*p*
model	*i*
ADAS application	
focus	at least *pv*
ideally, there is also a use case with ROS or there exists a library to make RM tool compatible with ROS	

Table 6.1: Tool requirements, missing columns can have any value

ID/ Filter criterion	**2**	**4**	**7**	**9**	**12**	**14**	**15**	**16**	**21**	**23**	**24**	**27**	**37**	**38**	**39**	**40**	**sum**
Specification																	
explicit output information: at least v	○	○	○	○	○	○	○		○	○	○	○	○	○	○	○	15
explicit physical time: not none, not ?				○		○	○	○	○			○		○			7
explicit modality: at least f	○	○					○		○			○	○		○		7
Monitor																	
tool: specified & ○pen source						○	○	○	○				○		○		6
Deployment																	
stage online: at least outline				○		○	○				○	○	○			○	7
architecture: c	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	16
instrumentation: none	○	○			○	○	○	○	○	○	○	○	○	○	○	○	14
Reaction																	
passive: at least so	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	16
Trace																	
precision: p	○	○		○	○	○	○	○	○		○	○	○	○	○	○	14
model: i	○	○		○	○	○	○	○	○		○	○	○	○	○	○	14
ADAS application																	
focus: at least pv	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	16
sum	8	8	4	8	7	10	11	8	10	5	8	10	10	8	9	8	

2:[MČSP22], **4**:[Han22], **7**:[Che21], **9**:[DTCL21], **12**:[GZWR20],**14**: [SUPR20], **15**:[ZBK20], **16**:[ALZ+20], **21**:[WKLS18], **23**:[RCL+18], **24**:[MHR16], **27**:[KCDK15], **37**:[BDH+21], **38**:[OKS19], **39**:[DADF18], **40**:[SL17]

Table 6.2: Filter criteria derived from our requirements applied to all retrieved RM techniques

After evaluation of the relevant aspects, we filter the entries from Table 5.5 and Table 5.6 based on the aspects from Table 6.1. We include the code in which we do the filtering in Listing A.2 in Appendix A. Also, in Table 6.2 we visualize each filtering step by showing the papers that fulfill each individual filtering criterion specified. After evaluation of the requirements and filtering out tools, we have one possible tool left: *rtamt*. The corresponding paper ID is ID 15 by Zapridou et al. [ZBK20]. Ideally, we want a tool that is compatible with ROS. The *rtamt* tool by itself is written in "*Python*" [VD09]. Thus, *rtamt* can be made compatible with ROS using the "*rospy*" [Ope17] library. However, this would require additional development effort. But, as Ničković and Yamaguchi[NY20] state, there also exists *rtamt4ros*. The *rtamt4ros* library is directly compatible with ROS. Using the *rtamt4ros* tool, we create our monitor on the robot. We describe our implementation in the next chapter.

7 Prototype Implementation

In the previous chapter, we evaluated the tool (*rtamt4ros*) to implement our monitor (see Chapter 6). In this chapter, we describe how we implement the prototype. In detail, we answer the following research questions, which are first defined in Chapter 4:

RQ3 How can the chosen RM technique be practically implemented in our prototype?

RQ3.1 What are the key considerations and challenges involved in the implementation process?

We provide insights into our implementation in Section 7.1. Additionally, we verify our monitor in Section 7.2 and provide details on our experiment in Section 7.3. To answer RQ3.1, we provide information on our challenges and key considerations in Section 7.4. Then, to answer RQ3, we implement the prototype with possible tools and provide the architecture of our solution in the next section.

7.1 Architecture

After evaluating which tools are usable on our hardware, we implement a prototype with the tools. We provide the architecture of the prototype in Figure 7.1. Some aspects of this figure are similar to Figure 6.2, so we do not explain them again. We only focus on aspects that were not mentioned yet or that have changed. We added implementation details as comments in Figure 7.1.

The sensors publish data to the ARV. Previously, the system inside the ARV was a black box system (see *?* in Figure 6.2). Now, we have the knowledge that we have an object detection ROS node (OD node) as well as a front distance measurement ROS node (FDM node) provided on the robot. The OD node works with *darknet_ros* with YOLOv3 which was trained on the COCO dataset. The FDM node is from the *simple_follower* ROS node, which is provided on the robot [Rob23a]. The OD node subscribes the RGB camera ROS topic. In the following, we just write the term topic to refer to a ROS topic. When the OD node receives data, it then detects objects in the camera images. The OD node publishes detected object data to a topic that the object tracking ROS node (OT node) subscribes to.

We implement the OT node based on *sort-deepsort-yolov3-ROS*. The OT node gives IDs to detected objects and tries to track equal objects within multiple frames. Ideally, equal objects should have the same ID. The OT node publishes data to a topic. This data and the camera topic data are subscribed by the AEB ADAS ROS node (AA node).

The AA node, which we implement from scratch based on the specifications in the previous chapter (see Chapter 6), publishes data on two topics: the wheel speed and the AEB signal. In our implementation, the wheel speed is either zero, if the ARV does an emergency brake, or a

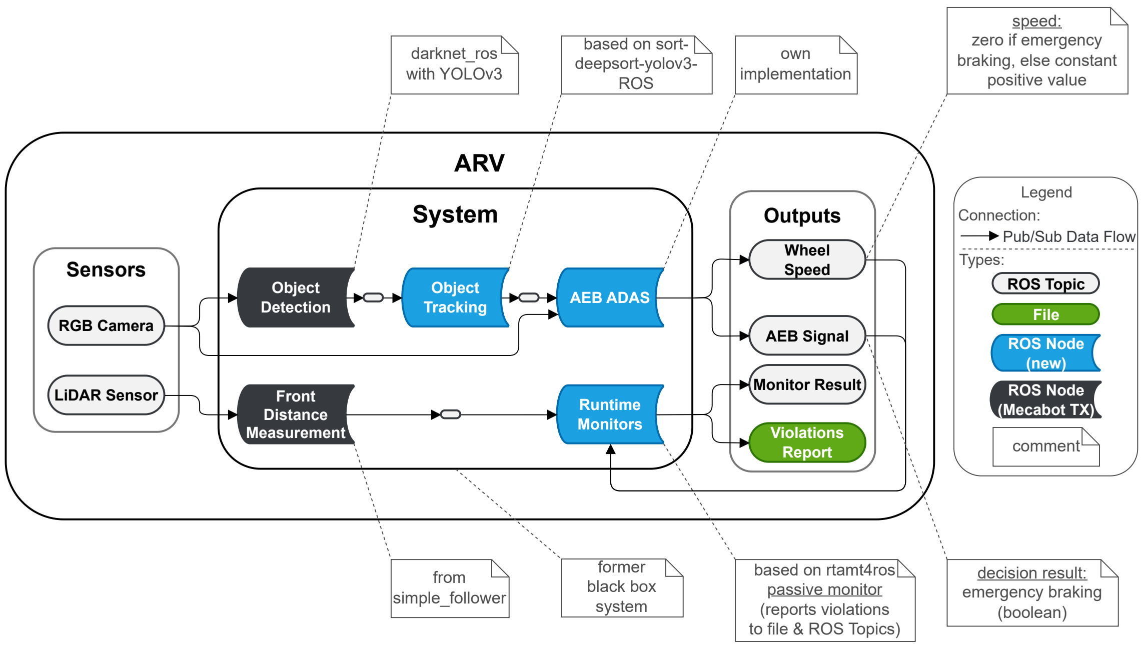


Figure 7.1: Architecture diagram of the prototype

constant positive value. The AEB signal contains the decision result. It is a boolean value which indicates whether the ARV should do an emergency brake (true) or whether the ARV should not do an emergency brake (false).

The provided FDM node subscribes the LiDAR sensor topic. When data is published, the FDM node receives the data and calculates the distance to the nearest object. The FDM node publishes this data to a topic.

The runtime monitors ROS node (RM node) subscribes to the FDM result topic. In addition, the runtime monitors node also subscribes to the wheel speed topic and the AEB signal topic. Here, the idea is that the runtime monitors can use the given wheel speed and distance to the nearest object in front to check the provided decisions by the AEB ADAS (i.e., whether the ARV should be in safe driving mode or in emergency braking mode). After checking the AEB results, the RM node publishes the monitoring result and writes to the violations report file. Note that this report does not only include the actual violations. Instead, we report all monitor outputs together with the input values and state whether this represents a violation or not. The RM node publishes the monitoring result to the monitor result topic. The RM node is based on our selected tool (*rtamt4ros*). It consists of four monitors (one for each SR from Chapter 6). They are passive monitors as they do not actively interfere with our system, but only report violations to a file and publish their results via ROS topics. In particular, the monitors cannot overwrite the driving decisions taken by the AEB ADAS. The violations represent violations of our previously defined SRs in Chapter 6. We simplify the architecture by leaving out how the *Mecabot TX* processes the outputs of the system. The robot subscribes to the wheel speed topic. The robot starts and stops driving according to the speed. For simplicity, we also omit the names of the topics between nodes in the system. The next sections further explain the blue ROS nodes.

7.1.1 Object Tracking

The object tracking ROS node subscribes to the */darknet_ros/bounding_boxes* topic provided by the object detection ROS node. This topic provides the bounding boxes of detected objects together with their COCO class labels (e.g., person or car). Whitelisted object classes restrict the tracker as it can only track whitelisted object classes. In our case, the whitelist contains the following COCO classes: person, car, bicycle, motorbike, bus, train, truck, boat, and stop sign. For each detected object, the tracker is updated with input (*bounding_box*, *prediction_probability*, *class_id*). On a given input, the tracker then checks if this object is already contained in the previous frame. This is done by comparing the new bounding box and COCO label with the bounding boxes and COCO labels of all objects in the last input image. If the tracker thinks that a given object is identical with an object in the last input image, the current object is assigned the same ID as the corresponding object in the last input image. Otherwise, the tracker selects a completely new ID for the given object. Note that a detected object can be assigned a new ID either if it is actually new (i.e., not contained in the previous frame) or because the current bounding box of the object differs too much from its previous bounding box. As visualized by Figure 7.1, the object tracking ROS node publishes to a topic (*/sort_track*) which is then consumed by the AEB ADAS ROS node.

7.1.2 AEB ADAS

In the AEB ADAS ROS node, we implement the logic of our AEB ADAS. As visualized in Figure 7.1, the AEB ADAS ROS node subscribes to the RGB camera topic and the */sort_track* topic provided by the object tracking ROS node. The camera images have width 640 px, height 480 px, and the origin is located in the upper left corner of the input image. The object tracking ROS node enables us to introduce cooldowns for detected objects. As an example, if an object is detected in a frame but not in the next frame, it does not directly get removed but lives on for a little while longer. The reason, why we want to do this is that we have identified that the used object detector (*darknet_ros* [Bje18]) produces very unsteady predictions. As an example, an object is detected in frame i, but not in frame i+1, but then again detected in frame i+2. This would cause large fluctuations in our system. By using the cooldowns for objects, we smoothen this object detection step. The consequence of this is that objects can live longer than they would actually live. But this is not a problem in our case, since this only means that the ARV performs an emergency braking maneuver in front of a detected object a little while longer than needed. On the other hand, if we would not use the object tracker, the detections of objects would be highly fluctuating leading to our AEB ADAS to constantly switch between safe driving mode and emergency braking mode, which is not desired by us. Therefore, the use of the object tracker introduces additional stability and safety for our system.

The danger zone definition is in the AEB ADAS ROS node (see danger zone from Figure 6.1). The following values are relevant for the size of the danger zone (in pixels): *danger_zone_min_x*: 140, *danger_zone_max_x*: 500, *danger_zone_min_y*: 140, *danger_zone_max_y*: 480. The figure also contains objects that overlap with the danger zone, but the objects are not fully within the zone. The minimum width that the bounding box has to be within the danger zone is 20 pixels. The minimum height that the bounding box has to be within the danger zone is 25 pixels. The safe driving speed is also defined here (0.1 m/s). The AEB ADAS ROS node is responsible for starting and stopping the ARV based on its evaluated signal. The AEB ADAS node regularly publishes the new wheel speed value and the evaluated signal (either safe driving or emergency braking) through ROS publications.

7.1.3 Runtime Monitors

Like the previous nodes, the runtime monitors have an input/output configuration and the monitor configuration. In the input/output configuration, there are the input and output ROS topics defined. As visualized in Figure 7.1, the node subscribes to three topics: the FDM topic, the ARV wheel speed topic, and the AEB decision topic. For simplicity, only one topic visualizes the monitor result in Figure 7.1. We simplify the architecture by combining the five output topics that could be used into one monitoring result in the architecture diagram in Figure 7.1. There is one topic for each SR and one combined topic (*monitor_output_all_topic*). As the monitors are passive, the monitor result is not actually used in our prototype. The last output (*output_logs_folder*) is for the violations report.

Basic property definitions are in the monitor configuration part. The *sampling_period* is the time period when a signal by the AEB system is expected (every 0.1 s). The *sampling_tolerance* is relevant for SR1. The value adds timely tolerance to the production of the AEB signal. In this case, we have the 0.5 as value. These values specify that the elapsed time between two input data samples must be in [0.5**sampling_period*,1.5**sampling_period*] s ([50,150] ms). For example, if one sample arrives 40 ms after the previous sample or 160 ms after the previous sample, this both counts as a violation. The robustness violation threshold set to the value zero defines that if the robustness (*rob*) is smaller than zero, there is a violation. The *min_distance_to_nearest_object* is a float that describes the minimal distance to the nearest object (safety distance). In the example, it is 1 m.

There is one formula for each SR from Chapter 6. As described in Table 5.6 at ID 15, *rtamt* requires to use STL as a formalism for writing specifications. Based on these STL specifications, *rtamt* generates the monitors by itself. Note that we use a dummy formula for SR1 (see formula 7.1 below), as we only care about the sample timing violations count outputted by the STL monitor. In the following, the STL specifications for our four SRs are listed. In example, formula 7.1 represents SR1, formula 7.2 represents SR2, formula 7.3 represents SR3, and formula 7.4 represents SR4:

$$\text{out} = (1 >= 1) \tag{7.1}$$

$$\begin{aligned}\text{out} &= (\text{nearestObjDist} < \text{safetyDistance}) \\ &\rightarrow (\mathit{eventually}[0{:}0.5](\text{aebIsEmergencyBraking} > 0))\end{aligned} \tag{7.2}$$

$$\begin{aligned}\text{out} &= (\text{aebIsEmergencyBraking} > 0) \\ &\rightarrow (\text{nearestObjDist} < \text{safetyDistance})\end{aligned} \tag{7.3}$$

$$\begin{aligned}\text{out} &= (\text{aebIsEmergencyBraking} > 0) \\ &\rightarrow ((\text{aebIsEmergencyBraking} > 0)\ \mathit{until}[0,2]\ (\text{speed} == 0))\end{aligned} \tag{7.4}$$

The formula 7.2 for SR2 describes that if the nearest object distance is smaller than the safety distance, then the AEB must respond with emergency braking signal within 0.5 s after violating the safety distance. The formula 7.3 for SR2 describes that if the AEB outputs an emergency braking signal, then the safety distance must be violated currently. Lastly, the formula 7.4 for SR4 describes that if the AEB outputs an emergency braking signal, then it must continue to do so until the ARV is stationary (speed is zero). For that, we assume that the emergency braking maneuver takes at most two seconds. For visualization, parts of the violations report are included in Listing 7.1. Note that this report does not only include the actual violations. Instead, we report all monitor outputs together with the input values and state whether this represents a violation or not. We describe the columns of the file further:

SR Provides information on which SR the row is about. Values can be the following: SR1, SR2, SR3, and SR4.

STL Contains the STL formula for the monitored property.

Logical Time The same logical time represents an integer counter that is incremented each time the monitors are evaluated. As for each logical time step, we report the outputs for all four SRs, the same logical time value can be found in four subsequent rows (e.g., lines two to five in Listing 7.1). The logical time connects the entries.

Sample Timestamp This is the timestamp the value (i.e., the AEB signal) is from.

Nearest Obj Dist It contains the distance to the nearest object as determined by the FDM ROS node using the LiDAR sensor.

Speed It contains the speed of the ARV. This can be either 0.1 m/s or 0 m/s.

Safety Dist In this example, the safety distance is always 1.0 m.

AEB Emergency Braking The values can be either -1.0 (false) or 1.0 (true). If the value is false, the ARV is not braking. If the value is true, the ARV is braking.

Robustness It is a value between [-1,1], that indicates the extents of the violation. A one indicates no violation. Everything below zero is a violation. The smaller the value is, the more the formula is violated. The robustness value is calculated by *rtamt*.

Violation True indicates a violation of the SR, false indicates no violation.

```
1  SR;STL;Logical Time;Sample Timestamp;Nearest Obj Dist;Speed;Safety Dist;AEB Emergency Braking;Robustness;
   Violation
2  SR1;1 >= 1;0;1700412683999655961;1.3669999837875366;0.1;1.0;-1.0;0;False
3  SR2;out = (nearestObjDist < safetyDistance) -> (eventually[0:0.5](aebIsEmergencyBraking > 0))
   ;0;1700412683999655961;1.3669999837875366;0.1;1.0;-1.0;nan;False
4  SR3;out = (aebIsEmergencyBraking > 0) -> (nearestObjDist < safetyDistance)
   ;0;1700412683999655961;1.3669999837875366;0.1;1.0;-1.0;1.0;False
5  SR4;out = (aebIsEmergencyBraking > 0) -> ((aebIsEmergencyBraking > 0) until[0,2] (speed == 0))
   ;0;1700412683999655961;1.3669999837875366;0.1;1.0;-1.0;inf;False
6  SR1;1 >= 1;1;1700412687024082899;1.3450000286102295;0.1;1.0;-1.0;-1;True
7  ...
```

Listing 7.1: Snippet from a sample violations report file

A description of the verification of the four monitors of the RM node is in the next section.

7.2 Verification of the Monitors

In this chapter, there are explanations on how the different monitors are verified. As already explained, there is one monitor for each SR from Section 6.2.

SR 1 For this verification, appropriate settings have to be determined. Therefore, all components run at the same time, and we check how frequently the monitor node gets an AEB result ROS message. The sampling period of SR1 monitor is set accordingly (approximately 10 Hz. Thus, the sampling period is 100 ms). We test the monitor with this setting and a tolerance of 50%. Accepted durations between two AEB samples are 50 ms to 150 ms. Any duration outside this interval should be detected as violation by the *rtamt* monitor. In our test run, we see that *rtamt* indeed detects these cases as violations and does not report false negatives.

We visualize the next test cases (SR2 to SR4) using Table 7.1. On the left most column, we have the SR that is being tested by the test case. Then in mocked condition, we have the variables that we mock for the test and what condition needs to be fulfilled by mocking their values. In AEB emergency braking, there are the necessary AEB signal values and changes in the value for the tests. Then we have the expected outcome of the test and the actual outcome of the test as two columns. If the expected outcome result and actual outcome result match, then the monitor passes the test successfully.

SR	Mocked condition	AEB emergency braking	Expected outcome	Actual outcome	Passed?
SR2	nearest object dist < safety distance	1.0	violation = false	violation = false	yes
	nearest object dist < safety distance	-1.0	violation = true	violation = true	yes
	nearest object dist >= safety distance	-1.0	violation = false	violation = false	yes
SR3	nearest object dist < safety distance	1.0	violation = false	violation = false	yes
	nearest object dist >= safety distance	-1.0	violation = false	violation = false	yes
	nearest object dist >= safety distance	1.0	violation = true	violation = true	yes
SR4	speed > 0	-1.0	violation = false	violation = false	yes
	speed > 0	1.0 to -1.0 before two seconds	violation = true	violation = true	yes
	speed > 0	1.0 to -1.0 after more than two seconds	violation = false	violation = false	yes

Table 7.1: Test cases and results of SR2, SR3, and SR4

SR 2 To simulate all relevant cases, there is the need of simulating incoming ROS subscription values. Firstly, set the nearest object dist and the safety distance such that the nearest object dist is smaller than the safety distance for the test. Then check if the monitor reports a violation when the AEB does not respond with a braking Signal. Otherwise, the monitor should not report a violation. Using a test run, we confirm that both cases are correctly handled by the monitor. Secondly, use fixed nearest obj dist and safety distance such that the nearest object dist is at least the safety distance. Check that the monitor does not report a violation. Using a test run, we confirm that the monitor handles the case correctly by not reporting a violation.

SR 3 To simulate all relevant cases, there is the need of simulating incoming ROS subscription values. Firstly, set the nearest object dist and the safety distance such that the nearest object dist is smaller than the safety distance. Then check that the monitor reports no violation even when the AEB emits a braking signal. Using a test run, we see that the monitor handles this case correctly. Secondly, set the nearest object dist and the safety distance fixed such that the nearest object dist is at least the safety distance. Then check that the monitor reports no violation when the AEB does not emit a braking Signal. The monitor should report a violation when the AEB does emit a braking Signal. Using a test run, we see that the monitor handles both cases correctly.

SR 4 To simulate all relevant cases, there is the need of simulating incoming ROS subscription values. For the speed of the robot, we set it to a fixed speed greater than zero for a sufficiently large number of time units (greater than two seconds, which is the time bound for the until operator) after the AEB emits a braking signal. After the time units, the speed is again set to fixed speed zero. We then test all relevant cases as follows: Firstly, check that the monitor never reports a violation when the AEB does not emit a braking signal. Using a test run, we can confirm that the monitor correctly handles this case. Secondly, place a person in front of the camera causing AEB to emit a braking signal. Then move the person out of the image quickly (before the logical time units pass) and check that the monitor reports a violation. The violation is caused due to the emergency braking signal switching to false (-1.0) again, although the braking maneuver is not successfully completed yet (as the speed is still greater than zero) and the two seconds time bound for the until operator has not passed yet. Intuitively, this means that the ARV stops the emergency braking maneuver before it halts completely. Using a test run, we confirm the monitor handles this case correctly. Thirdly, place a person in front of the camera, causing the AEB to emit a braking signal. Then move the person out of the image only after more than two seconds. Check that the monitor reports no violation. Using a test run, we confirm the monitor handles this case correctly.

This completes the verification of the monitors' implementation. This is needed for a correct evaluation of the usefulness of the implemented monitors for our use case. For the following experiment, we can be sure that the monitor is not buggy, causing it to produce wrong verdicts (robustness values) for given input values. However, input values could still be out of sync or delayed, causing the whole approach to not be useful in our setting. This needs to be investigated now in the experiment with a test track.

7.3 Experiment

As the implementation of the prototype was done using something similar to a test bench for cars, just in smaller size for the ARV, we did an experiment to test the implementation while the ARV was able to actually drive. A simple test track was set up for the ARV which consisted of a straight path. We then performed one experiment with different obstacles on the track. For the experiment we used a dynamic obstacle at the side of the track and the others were placed directly on the track. The aim of the AEB system was to let the ARV perform an emergency brake for all obstacles that were on the track (immediate danger) and no emergency brake for all obstacles beside the track (no immediate danger). After the experimental run, we observed the output file produced by the RMs that contained violations for the different SRs that were identified. We then analyzed the validity of the reported violations, which allowed us to give an insight into the effectiveness of the applied RMs for improving the safety of the AEB ADAS system.

7.3.1 Setup

The implementations of the monitors were tested with an experiment. For the experiment, the ARV was connected to a laptop via remote access. The necessary programs on the ARV could be started without connecting the ARV to a monitor itself by using remote access. This made the ARV movable as it was not restricted by cable lengths. The system and its nodes were started in the following order:

1. Started ROS on the *Mecabot TX*
2. Started *darknes_ros* with YOLOv3 object detector
3. Started *sort_track* SORT object tracker
4. Started LiDAR object distance tracker
5. Started AEB perception-based ADAS system
6. Started runtime safety monitoring system

The initial setup of the experiment is visualized in Figure 7.2 as starting position. The straight path consisted of sixteen 60 cm big square tiles. The ARVs length and width are rounded, the exact sizes are in Section 6.1. The ARV was placed on the first tile in the bottom-right corner, as visualized in the figure. All obstacles were printed out images of obstacles in A4 format that we attached to objects to make them face the ARV by themselves. There were two obstacles, which were initial obstacles, as they were placed during the experiment setup (initially). The image of the first obstacle is Figure 7.3a. The image of the second obstacle is Figure 7.3b. The first obstacle (green rectangle with number one) was positioned with a distance of three tiles from the ARV. The second obstacle was placed three tiles from the first obstacle. As previously described, there were three more obstacles placed during the experiment, which we will explain in the next section. Multiple cameras were used to record the experiment. We placed one camera behind the ARV and the other camera was carried by the person placing and removing obstacles during the experiment. By using cameras, stopping positions of the ARV and positions of the obstacles could be evaluated. Additionally, we recorded the screen of the laptop during the experiment to record the camera view and violations of the ARV.

7.3.2 Execution and Results

When the experiment started, the ARV started driving at a speed of 0.1 m/s. This value was constant as the speed was either zero or the constant value we choose, in this case it was 0.1 m/s. The safety distance was also a constant with a value of 1 m. The driving direction was indicated by the arrows in Figure 7.2. While the ARV drove, it processed the environmental information.

For the first obstacle, the ARV did two stops. The first stop was approximately 1.5 tiles away from the first obstacle. The second stop was approximately 1.16 tiles away from the first obstacle, as visualized in Figure 7.2. After the first stop, the ARV started driving again. At the second stop, there was no movement anymore. We then removed the first obstacle from the track. Now we investigated how the ARV reacted to the second obstacle. The ARV did two stops again (third and fourth stop), similar to the first two stops. When the ARV finally stopped at the fourth stop, we placed the third obstacle on the track. It was placed three tiles from the second obstacle. The

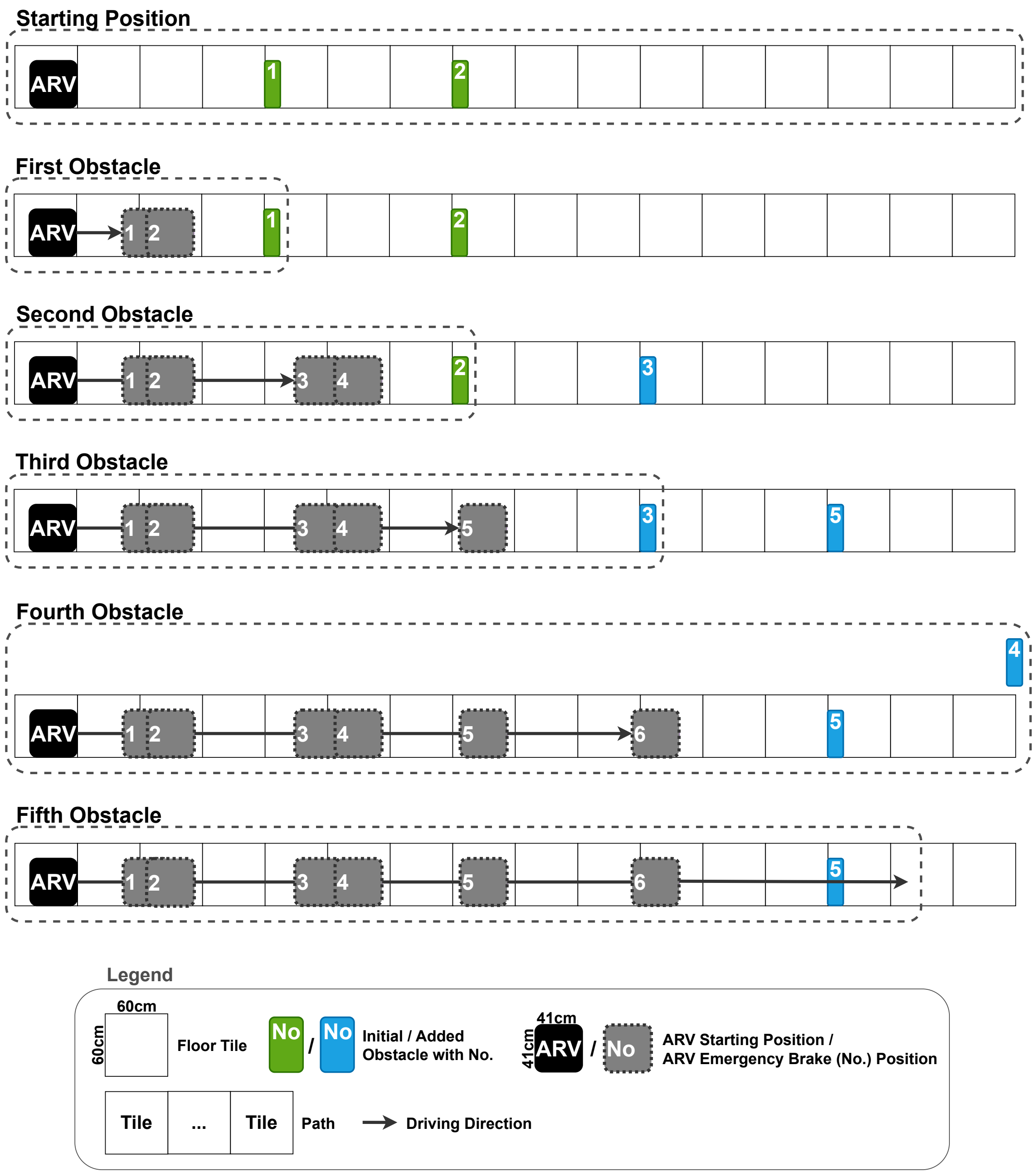
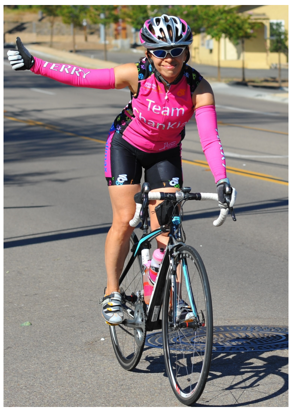


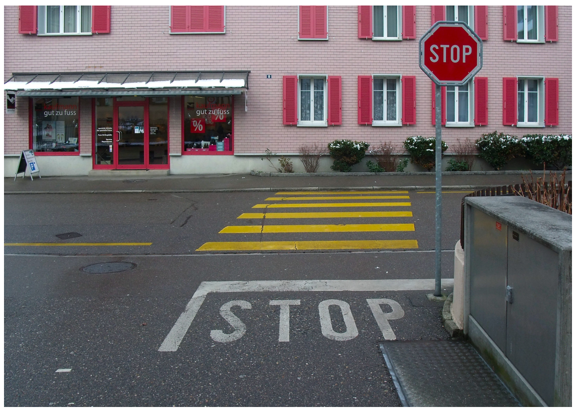
Figure 7.2: Experiment sequence of events

(a) Image 1 [ami10] showing a stop sign

(b) Image 2 [mal07] showing multiple persons on a crosswalk



(c) Image 3 [Hun12] showing a person on a bicycle, cropped



(d) Image 4 [Kec13] showing a crossing with a stop sign

Figure 7.3: Images used as obstacles for experiment

image of the second obstacle is Figure 7.3c. After placing the third obstacle, we removed the second obstacle. Removal and addition of the obstacles was done by a person walking into the frame and grabbing the obstacle to remove it. For adding the obstacles, the person was also in the frame. After the removal of the obstacle, the person walked out of the frame and the ARV started driving again. At the third obstacle, the ARV did its fifth stop, approximately 2.16 tiles away. When the ARV stopped, the person placed the fifth obstacle three tiles away from the third obstacle. The image for the fifth obstacle is Figure 7.3d. For the third obstacle, there was only one stop (the fifth). The fourth obstacle was a random person walking into our test track. The distance to the ARV was not accurately visualized for this obstacle. The ARV did a stop approximately 2.5 tiles away from the fifth obstacle (the sixth stop). When the fourth obstacle walked off the track, the ARV started driving again. For the fifth obstacle, the ARV never stopped. It drove against and over the obstacle. We then stopped the ARV and the experiment was finished. The monitors created the violations report file during the execution.

7.3.3 Evaluation

When observing the ARV emergency brake positions, we looked at the distance to the nearest objects and the screen recordings of the object detector and AEB ADAS.

ARV Emergency Brake 1. Position At this position, the object detector and the AEB ADAS detected the stop sign and the ARV halted approximately 96 cm away from the sign (similar to Figure A.1a and Figure A.1b but it should be further away). Then, the object detector and the AEB abruptly did not recognize the stop sign anymore. The cooldown by the AEB ADAS was not enough to keep the ARV at a stop long enough.

ARV Emergency Brake 2. Position The ARV drove further until it recognized the sign again (see Figure A.1a and Figure A.1b). The stop was within the safety distance at approximately 78 cm from the obstacle. The first halt should have been the only one for the first obstacle. Even though the ARV stopped to recognize the obstacle and drove again, it did not collide with the obstacle. In Figure A.1d, there is the screenshot of a cooldown of the stop sign.

ARV Emergency Brake 3. Position At the third stop, the object detector recognized a bicycle (see Figure A.2a). The AEB ADAS recognized the bicycle within the danger zone a few frames later (see Figure A.3b). Thus, it stopped approximately 1.13m from the third obstacle. The bicycle was, in fact, nonexistent. The floor had round holes in it, which were recognized as bicycle. Then, the bicycle was not recognized anymore and the ARV drove again.

ARV Emergency Brake 4. Position At the fourth stop, the ARV halted for the pedestrians approximately 79 cm away from the obstacle. The object detector initially mismatched the persons with another COCO object class (see Figure A.3a), this may have been caused by two pedestrians standing near to each other, and it looked like an object with four legs instead of two objects with two legs each. The AEB ADAS recognized the persons. Thus, the ARV then halted because it correctly recognized the persons as such (see Figure A.3c and Figure A.3d).

ARV Emergency Brake 5. Position At the fifth position, the ARV stopped approximately 1.42 m away from the obstacle. The object detector recognized a boat and a person as objects. The person was correctly recognized. The boat, on the other hand, was not correctly recognized as the object detector recognized it on the floor, but there was no boat (see Figure A.4a and Figure A.4b).

ARV Emergency Brake 6. Position The AEB ADAS recognized a person within the danger zone (see Figure A.5b). Thus, it stopped driving. The fifth obstacle was approximately 1.41m away.

There were no more stops after the sixth stop, as the object detector could not recognize the stop sign in the image. The object detector did recognize the whole obstacle as a TV monitor, but this was not a COCO object class that the AEB ADAS had whitelisted, so it did not stop for it. That was also why we knew with certainty that the sixth stop was caused by the fourth obstacle. In Appendix A, there are more images included, e.g., regarding the cooldowns of objects (see Figure A.4d) or examples where the object detector recognized objects that were not whitelisted by the AEB ADAS. Therefore, they were not recognized by the AEB ADAS (see Figure A.5c and Figure A.5d).

The violations report contained 4305 items (rows). For the purpose of this thesis, we filtered the relevant aspects.

SR1 There were 573 SR1 violation cases in total. These violations occurred regularly because the monitor ROS node received new data samples either too fast or too slow after each other (not within [50,150] ms compared to the previous sample). The occurrences were investigated in more detail. We identified that the smallest amount of time between two SR1 samples in our test run was 18 ms, and the longest amount of time between two SR1 samples was 562 ms. Thus, these values represent the extreme cases for the time elapsed between two samples. Furthermore, in 39% of the cases a sample arrived too early (less than 50 ms elapsed since the last sample), and in the other 61% of the cases, a sample arrived too late (more than 150 ms elapsed since the last sample).

This suggested that the current expected sampling rate (10 Hz) should be lowered a little, since in the majority of cases the samples arrived too late. This adaptation could lead to less SR1 violations being thrown. Generally, we had a large discrepancy in the elapsed time between two samples ([18,562] ms) which would not be desired in a real production system. This could be due to unsuitable outgoing and incoming queue sizes at the ROS publishers' and ROS subscribers' queues. For more information on the queue sizes, visit the ROS documentation [Ope23].

SR2 There were 84 SR2 violation cases in total which could be divided into three unique occurrence series.

Occurrence Series 1/3 This was the situation where the ARV stopped in front of the stop sign (the first obstacle and the first brake) but then continued driving again at logical time unit 140 because the stop sign was not recognized anymore. The safety distance was still violated at this point (distance to the nearest object in front was less than 1 m) so the monitor of SR2 started tracking if in the next 0.5 s the AEB would emit an emergency braking signal again in response to the violated safety distance. However, this signal did not arrive (this is visualized in the screen recording, where the ARV did not recognize the stop sign anymore and continued driving). Thus, the SR2 monitor reported a violation from logical time unit 145 (approximately 0.5 s after logical time unit 140) until the object in front was detected again at logical time unit 151 by the object detector. Therefore, it was also detected by the AEB ADAS.

A possible solution to improve the AEB ADAS is to increase the cooldown in the AEB ADAS node to better handle cases where the object detector suddenly did not recognize objects anymore, but then did again.

Occurrence Series 2/3 This was the situation where the ARV stopped in front of a wrongly detected bicycle on the floor (the third emergency braking position) but then continued driving and did not recognize the pedestrians on the crosswalk (the second obstacle) in front soon enough. At logical time unit 350, the safety distance was first violated at this point (distance to nearest object in front was less than 1 m) so the monitor of SR2 started tracking if in the next 0.5 s the AEB would emit an emergency braking signal again in response to the violated safety distance. However, this signal did not arrive (in the screen recording: the ARV first did not recognize the pedestrians on the crosswalk but mistook them for other objects (i.e., cows) and only detected actual persons at a much later point in time). Thus, the SR2 monitor reported a violation from logical time unit 355 (approximately 0.5 s after logical time unit 350) until the object in front was detected correctly at logical time unit 361 by the object detector. Therefore, it was also correctly detected by the AEB ADAS.

A possible solution to improve the AEB ADAS is to use a more sophisticated object detector that does not mistake two persons side by side on a crosswalk as cows.

Occurrence Series 3/3 At logical time unit 993 the safety distance was violated (LiDAR distance to nearest object in front was less than 1 m) so the monitor for SR2 started tracking if in the next 0.5 s the AEB would emit an emergency braking signal in response to the violated safety distance. However, this signal did not arrive (in the screen recording: where the ARV did not recognize the stop sign at the crossing and just drove over it). Thus, the SR2 monitor reported a violation from logical time unit 998 (approximately 0.5 s after logical time unit 993) until the object in front was completely run over (LiDAR distance to next object in front was again at least 1 m).

A possible solution to improve AEB ADAS is to adapt the AEB danger zone because as soon as the small stop sign was detected it was outside the danger zone, causing the AEB not to stop in front of it.

SR3 There were 232 SR3 violation cases in total which could be divided into six unique occurrence series. This violation describes when the ARV braked too early, or when the ARV braked even though the nearest object distance was at least the safety distance. In general, this can mean that the AEB ADAS was not working perfectly, as it started the emergency braking maneuver too early. For context, this is a problem, as a stop too far away from, e.g., a pedestrian crosswalk or a stop sign is not wanted behavior. Here should be improvements done in the future, e.g., by adapting the size of the AEB danger zone or the minimum overlap size of objects with the AEB danger zone.

Occurrence Series 1/6 At logical time unit 118 the AEB ADAS outputted a correct emergency braking signal but the current distance to the nearest object in front was still a little over the safety distance (1 m). In fact, this case did not represent an actual problem of the AEB ADAS.

Occurrence Series 2/6 and 4/6 At logical time units 203/529, after the object in front (in front of which the ARV successfully stopped) was removed. Due to the cooldown configured in the AEB ADAS, the previous objects (i.e., persons/ stop sign) were still detected for some time although the distance to the nearest object in front was now greater again, e.g., greater than the safety distance (1 m). Therefore, the monitor of SR3 reported a violation because the AEB ADAS still emitted the emergency braking signal, although the safety distance was not violated anymore. After the cooldown was over at logical time unit 216/542, the emergency braking signals stopped. Thus, no more SR3 violations were reported.

This problem came from the configuration of the cooldown inside the AEB ADAS. The number of SR3 violations caused by this could be lowered by lowering the cooldown, but this could result in a higher number of SR2 violations (see above). There is a trade-off between high cooldowns leading to more SR3 violations and low cooldowns leading to a higher number of SR2 violations.

Occurrence Series 3/6 At logical time unit 317, the object detector wrongly detected a bicycle in the tile on the ground. Thus, the AEB ADAS emitted an emergency braking signal. However, the safety distance was not violated at this point, since the current distance to the nearest object in front was larger than 1 m. The monitor of SR3 reported a violation here until logical time unit 332, where the bicycle was not detected anymore.

Occurrence Series 5/6 At logical time unit 636 the object detector wrongly detected a boat in the tile on the ground. Thus, the AEB ADAS emitted an emergency braking signal. However, the safety distance was not violated at this point, since the current distance to the nearest object in front was larger than 1 m. The monitor of SR3 reported a violation here. After wrongly detecting the boat, the system correctly identified the person on a bicycle in front, but still, the safety distance was not violated so the monitor of SR3 continued reporting violations. Even after the object in front was removed, the violations continued for a little while again due to the cooldown of the AEB ADAS until logical time unit 789 where the object in front was not detected anymore.

Occurrence Series 6/6 At logical time unit 916 the object detector detected a person entering the room at a long distance. Since the detected person was within the AEB danger zone with large overlap, the AEB ADAS started an emergency braking maneuver. However, the safety distance was not violated at this point, since the distance to the nearest object in front was larger than 1 m. The monitor of SR3 reported a violation here until logical time unit 954 where (after the cooldown) the person was not detected anymore and the AEB ADAS continued driving.

SR4 There were three SR4 violation cases in total. Until logical time 215, the ARV did emergency braking. Then the obstacle in front was removed, leading to the AEB to detect that there was no obstacle any longer. Thus, the AEB switched to safe driving mode. This means that the AEB published a wheel speed of 0.1 m/s and a new AEB result with an emergency braking value of -1.0 (false). The problem was that the monitor subscribed to the wheel speed and AEB result topics independently and in this case, the monitor already received the new wheel speed (0.1 m/s) but still an old AEB result (with an emergency braking value of 1.0 (true)). On receiving the old AEB result, the monitor was evaluated again, meaning that the two seconds period for checking SR4 was started (because emergency braking exceeds zero). The problem was that in the next frame, we retrieved the updated AEB result (with an emergency braking value of -1.0 (false)) and the wheel speed stayed 0.1 m/s because the ARV was in safe driving mode. Thus, after the two seconds period, the SR4 monitor reported a violation because the monitor thought that at logical time 215 a new emergency braking maneuver was started which was stopped before the wheels stopped (zero speed). But this was false because there never was a new emergency brake maneuver, an old value was retrieved for the AEB result. To solve this, the input values should be better synchronized in the future, e.g., we should send the current wheel speed and safety distance inside the AEB node and make sure that the times are synchronized. To summarize: the problem here did not originate from a wrongly implemented SR4 specification (monitor) or from a wrong AEB ADAS implementation, but from an unsynchronized use of multiple subscriptions for input data to update the monitors with.

In this section, we described our experiment, together with its execution, results, and evaluation. In the next section, we will outline our main challenges and key considerations of this thesis.

7.4 Challenges and Key Considerations

In this chapter, we provide information on the challenges and key considerations regarding our implementation process of the prototype. They range from language-specific challenges to hardware restrictions and software considerations. We also provide insights into our implementation and explain our choices for the final implementation.

7.4.1 Challenges Working with the Robot

A large fraction of these challenges could be summarized as "working with the robot". Nobody involved in this thesis worked with the *Mecabot TX* before. Thus, we had to learn everything from scratch and from examples how to work with the robot. But there were also restrictions due to the hardware provided by the *Mecabot TX*. In the following, we describe these challenges.

Foreign Language Most of the code samples on the *Mecabot TX* only contained comments in a foreign language, which we do not speak or read. Therefore, it was hard for us to get started with the implementation.

Deprecated Tutorials Initially, we wanted to use a newer version of ROS, namely "ROS 2 Galactic" [Ope23]. But this version required the *Ubuntu* 20.04 operating system to be installed on the robot [Ope23]. However, the pre-installed *Ubuntu* version on the robot was *Ubuntu* 18.04 and setting up the robot was extremely time-consuming. This was because we, firstly, have never worked with the robot and secondly, the online tutorials were either deprecated or for another robot family [Rob23c]("Rosbot" [Rob23c]). We only knew that the tutorials were deprecated because we contacted the Roboworks [Rob23b] support. Thus, we did not update the robot from *Ubuntu* 18.04 to *Ubuntu* 20.04. Generally, we had to work with deprecated tutorials and samples with comments we did not understand, which complicated the entire implementation process.

Lack of ADAS Solutions First, existing ADAS solutions to use with our environment were challenging to find. The retrieved solutions then were either developed for specific other hardware (e.g., another robot), or were incompatible with our hardware. Even if a solution was compatible with e.g., ROS, our hardware restrictions did not allow us to use this solution. For example, the state-of-the-art AD toolkit "Autoware" [The23c] was compatible with ROS but had higher system requirements than we could offer (see [The23a] and [The23b]). Therefore, it would not have run on the *Mecabot TX*. Generally, this restricted us in the tools we could use on the robot. All in all, we did not find any existing solution that could be used in our environment out of the box.

Own ADAS Solution As we mentioned, we could not use an existing AEB solution out of the box. However, there existed foundational models and systems, e.g., the YOLO object detector with direct ROS integration. Therefore, we used the YOLOv3 implementation for ROS provided by the *darknet_ros* repository to implement our own AEB ADAS system.

Further Hardware Restrictions The available hardware restrictions for the *Mecabot TX* also caused challenges for the YOLO object detection. Our goal was to use a newer version of the YOLO object detector. These newer versions were compatible with the YOLO implementation for ROS provided by the *darknet_ros* repository. However, after setting up newer versions of the YOLO object detector on the *Mecabot TX*, we identified that the performance was too bad for using it in a real-time setting. In example, with newer versions of YOLO, we only retrieved one frame per second from the object detector, compared to approximately 15 frames per second using YOLOv3. Therefore, we were restricted to use the older version of YOLO, namely YOLOv3, for implementing our AEB ADAS.

7.4.2 Key Considerations for our Implementation

Now we describe our key considerations that were made for implementing our own AEB ADAS system on the ROS-based *Mecabot TX*. First, we explain the considerations behind the implementation of our AEB ADAS.

Camera-based AEB We decided to implement a camera-based AEB system. The reason for this was that we wanted to detect concrete objects and categorize them based on their object class. This allowed us to only perform an emergency brake for certain object classes.

Speed For simplicity, we used a constant positive speed of 0.1 m/s for the safe driving mode of our AEB ADAS and switched the speed directly to zero when performing an emergency brake.

Whitelisted Object Classes We only want our AEB ADAS to perform an emergency brake whenever an object of a certain class is detected in front of our robot. Therefore, we used a whitelist for filtering all COCO object classes to only keep those where our AEB should cause an emergency brake. For example, we did not put any food object classes (e.g., donut or banana) on this whitelist because driving over such an object does not represent a safety threat.

Object Detection Cooldown We noticed that the YOLOv3 object detector we used for implementing our AEB ADAS produced very unsteady results. For example, the object detector detected an object in a frame, but then failed to do so in the next frame, although the object was still present. Later, the object was detected again. This would cause our AEB ADAS to first perform an emergency brake, but then switch to the safe driving mode again, before performing an emergency brake again. We did not want this unsteady behavior for our system.

Our solution for this problem was to introduce a cooldown to all detected objects. In example, a detected object lives on for a little while longer even if it is not detected anymore by the YOLOv3 object detector. The duration of this time was given by the cooldown that was used. As soon as the YOLO object detector detected an object again, the cooldown is again resetted.

However, the use of such cooldowns required that we could identify objects as the same over the course of multiple frames. Otherwise, we could not associate a detected object with a cooldown and decrease or reset the cooldown value for a specific object. Therefore, we additionally implemented an object tracker to associate each object with a unique ID.

For the object tracker, we found an existing repository called *sort-deepsort-yolov3-ROS*. This repository contained implementations of both the SORT object tracker, and its extension called DeepSORT. We tried to use the newer DeepSORT variant of the object tracker. However, this required a specific version for a *Python* dependency that was incompatible with our software setup on the *Mecabot TX*. Therefore, we decided to use the simpler SORT tracker.

The use of the tracker also allows monitoring of further properties in the future. For example, Balakrishnan et al. [BDH+21] monitored the consistency of detected objects and the smoothness of their bounding box trajectories across subsequent image frames. This would not be possible without the use of an object tracker. Therefore, such properties can easily be added to our RM component in the future without the need for large adaptations of the source code.

AEB Danger Zone We decided to introduce a danger zone to our AEB ADAS which represents a sub area in our input image frames. With this danger zone, we only considered detected objects in the view of the *Mecabot TX* to be a safety threat, if the detected bounding boxes of the objects have an overlap with this sub area. This allowed our AEB ADAS to only cause an emergency brake for objects that are close to the robot and in front of the robot, rather than at the side. Further, we specified that the overlap of each object's bounding box with the danger zone must have at least a given width and height. This was added to further prevent our AEB to produce an emergency brake for objects that are very far away (i.e., having a very small bounding box).

Next, we describe our considerations behind the implementation of our runtime monitors.

LiDAR Front Distance Measurement We decided to use an existing system on the *Mecabot TX* from the *simple_follower* module for determining the distance to the nearest object in front of the robot (in meters) using the LiDAR sensor attached to the robot. This enabled us to provide our runtime monitors with additional information that could be used to check the decisions of our AEB ADAS independent of the camera inputs. Therefore, we had an additional input mechanism to check the safe operation of our AEB ADAS.

Safety Distance We decided to use a safety distance of one meter for the implementation of our runtime monitors. This value was chosen specifically for our fixed safe driving speed of 0.1 m/s. In a realistic environment where the driving speed is not constant, this safety distance would need to be calculated dynamically at runtime, e.g., based on the current speed of the vehicle. For simplicity, we used this fixed safety distance in our prototype.

Safety Requirements The considerations behind our four SRs are that we wanted to monitor different safety-related aspects of our AEB ADAS. First, we were interested in the timeliness of the inputs for our AEB ADAS. This could be checked using the runtime monitor of SR1. Next, we wanted to detect false negatives and false positives for the braking signals produced by our AEB ADAS. These were covered by SR2 and SR3. Last, we wanted to detect whenever the AEB ADAS stops an emergency braking maneuver, although it is not completed yet, therefore we introduced SR4. This list of SRs is not complete, as additional aspects for monitoring can always be added. Thus, our intention was to provide a proof of concept that covered different aspects to monitor. This showcases the flexibility of our implemented RM approach.

Sampling Period For monitoring the timeliness of the inputs in SR1, we needed to specify a certain *sampling_period* in *rtamt*. To choose the value for the *sampling_period*, we performed a test run of our complete system and checked how often new AEB decisions are produced. We identified the output rate to be approximately 10 Hz, therefore we decided to use a *sampling_period* of 0.1 s. However, this value was chosen based on this one test run only and was not derived systematically. Therefore, in a realistic environment, choosing the value for the *sampling_period* is more difficult which must be considered for ensuring the correct operation of the associated runtime monitor.

Passive Monitors For simplicity, we decided to use passive runtime monitors. In example, instead of actively interfering with our system, our runtime monitors only report violations to an output file. Thus, for making use of our runtime monitors, one must inspect this output file, either manually or automatically. This can then help to test and debug the ADAS and to improve its safety in new software iterations. We decided to use passive monitors because we first wanted to investigate the usefulness and safety of our deployed runtime monitors. In the future, we could consider replacing these monitors with active monitors that can overwrite the decisions of the AEB ADAS to improve the overall safety of the system.

Using the rtamt library Based on our tool selection, we decided to use the *rtamt* library for implementing the runtime monitors. During the course of the implementation, we noticed that this library was really intuitive. Therefore, it was easy to use, even for beginners. The available *rtamt4ros* package allowed us to easily integrate *rtamt* in our ROS-based system. Also, specifying the SRs as STL formulas was straight-forward, also due to the detailed documentation provided by Ničković and Yamaguchi [NY20].

The only challenge for us was specifying a *.stl* file to provide as input to *rtamt4ros*. This was because in these specification files, each input variable for the STL formula had to be annotated with one ROS topic that provided these values. The *rtamt4ros* library would then set up subscribers for each input variable automatically. However, in our prototype, multiple input variables sometimes needed to be derived from the same ROS message. Therefore, we could not use these specification files. Instead, we implemented our own *Python* script based on the available scripts provided by *rtamt4ros*. In this script, we manually set up the ROS subscriptions for the input variables and handled the data management ourselves.

One more challenge for us was using different datatypes for these input variables. In example, we wanted to use a boolean variable for the AEB ADAS decision result (the emergency braking signal). However, we faced problems with the imports of these datatypes. Therefore, we converted all input variables to floating point numbers and used the default float datatype for each variable. For the boolean variable, we replaced the True value with the number 1 and the False value with the number -1.

Also, we added logic for handling the outputted robustness values produced by the *rtamt* monitors. In example, a specification was considered violated under the current input values, if the robustness value produced by the *rtamt* monitor was below zero.

Analyzing the Violations Report As mentioned above, we considered each negative robustness as a violation of a specification under the current input values. However, the concrete robustness values were still useful, since we also outputted them to the violations report produced by our runtime monitors ROS node. There, the robustness values indicated how strongly the specifications were violated without having to look into each individual input value in detail. This allowed us to easily scan the violations and allowed us to find the most severe violations.

To further assist in analyzing the violations report, we decided to add a logical time in addition to the timestamps of the input samples that are provided as input to the *rtamt* monitors. This logical time was an integer counter that was incremented each time the runtime monitors were evaluated with new input data. Using this logical time, we could easily associate specific violations with the exact moment in time they were caused in our screen recordings. This allowed us to analyze the correctness of the reported violations and to determine their root causes.

All in all, manually analyzing the violations report still was cumbersome to do. Therefore, one idea is to increase the automation of this step in the future. As a concrete example, we could build a dashboard with a user interface that automatically visualizes the violations associated with the relevant sequences in the screen recording, e.g., based on the logical time. This would remove the need for manual association. In general, understanding the root causes of the violations was crucial for testing and debugging the system with the ultimate goal of improving its overall safety.

Verification of our Runtime Monitors To make sure that our implemented runtime monitors operated correctly, we used multiple test cases for verification of each monitor. This allowed us to ensure that the monitors acted as required as long as they were provided with correct input values. This allowed us to ensure that wrongly reported violations were not caused by a faulty implementation of a monitor, but rather by a wrong (e.g., unsynchronized) combination of input values for the variables in its STL specification.

In this chapter, we provided detailed information on the implementation of our prototype. In example, we described our final architecture, the verification process of the runtime monitors, our experiment, and our key considerations and challenges in the implementation process. In the next chapter, we will discuss the answers to our research questions by summarizing our results.

8 Discussion

In this chapter, we answer our research questions in Section 8.1 by summarizing our previous results, and describe our threats to validity in Section 8.2.

8.1 Discussion

Here, we summarize our process and answers for each research question, as we already answered the research questions in the previous chapters.

RQ1 What are the current state-of-the-art RM techniques in the automotive domain, especially for ADAS?

In Chapter 5, we described the whole RR process. We used our search query to collect current state-of-the-art RM techniques in the automotive domain. Here, we focused on ADAS in the query. By trying multiple possible queries, we discovered that some keywords lead to too many false positives. These keywords are: “RV”, “ARV” and “automotive”.

We collected multiple results and filtered them by quality and contents. Then we extracted data from the remaining 16 papers by reading them. We classified the data using a taxonomy by Falcone et al. [FKRT21]. We visualized the results in two tables (see Table 5.5 and Table 5.6). With all of that, we collected and categorized the 16 current state-of-the-art RM techniques in the automotive domain, especially for ADAS.

Additionally, we did some analysis on the papers, discovering that the highest similarities from our sample (71%) both represent cases in which one paper referenced the other paper. The lowest similarity (16%) was retrieved for two papers which did not have a reference between each other.

RQ2 Which RM technique is optimal in the context of our prototype?

RQ2.1 What were the key factors that had to be considered during selection?

For answering RQ2, we firstly have to answer RQ2.1. We first needed to define the hardware requirements of the provided robot (*Mecabot TX*), as this restricted us in our decision. Then we worked on defining the context of our prototype by describing a use case. In our use case, we wanted to do AEB with the ARV. We explained our idea with the AEB danger zone and visualized this idea. We also described four SRs that were relevant for the runtime monitors. We described how a possible data flow could look like with the knowledge we had during that time on the robot and its provided examples and features. We thought about what useful outputs would be. With all these key factors that we had to consider, we then chose a technique for the prototype.

For answering RQ2, we provided Table 6.1 with the important aspects from the classifications. We oriented on the structure and aspects of the classification. Relevant aspects were, e.g., that the explicit output information contain verdicts for properties to check or that the tool should be open-source. Overall, the relevant aspects were about, e.g., the output information, the physical time, the modality, the reaction and more. The optimal tool that we evaluated with these criteria was *rtamt*. In general, we see a great potential in using the classification results of our retrieved RM techniques to select an optimal technique for the implementation.

RQ3 How can the chosen RM technique be practically implemented in our prototype?

RQ3.1 What are the key considerations and challenges involved in the implementation process?

For answering RQ3, we first answer RQ3.1. The greatest considerations and challenges involved in the implementation process were: the foreign language in code samples on the robot, deprecated tutorials for the robot and the hardware restrictions.

Working with the robot contained a lot of trial and error with samples one did not understand and tutorials that were deprecated. The hardware was challenging as it restricted us in the toolkits we could use but also in the speed of progress as the robot shut itself down if too much, e.g., tabs in a browser were opened. There were additional challenges, but they were less challenging as there were always alternative solutions for them.

To answer RQ3, we visualized and explained our architecture in the context of ROS and its topics and nodes (see Figure 7.1). To the existing nodes on the robot, we added the object tracking ROS node, the AEB ADAS ROS node, and the runtime monitors ROS node. We then explained the nodes. We had four runtime monitors in the corresponding node. We had one monitor for each SR. We added the STL formulas in the descriptions. We used future operators in our solution (*eventually* and *until*). We used *rtamt4ros* for our solution, as it was even more optimized for ROS than *rtamt* itself. For the generation of the monitors, only the STL formulas were needed. The generation of the monitors itself was handled by *rtamt* as well as calculating the robustness value. We included a snippet from a sample violations report file. We verified the monitors by simulating some incoming ROS topic values and evaluating whether the monitors delivered the expected outcome. We verified all four monitors to ensure that the monitors were working correctly for the experiment.

The experiment itself showed multiple anomalies in the system's behavior. The ARV did more emergency braking maneuvers than we expected and collided with an obstacle. In the evaluation of the experiment, we discovered causes of the unexpected behaviors. One cause was that the object detector suddenly did not recognize the objects anymore (SR2 violation). This would cause safety issues if it was in a real-world setting. In our experiment, the ARV started driving again, even though it should not have done that. By doing that, the distance to obstacles became even less than at the first stop. We tried to fix that with a cooldown. With the cooldown, the bounding boxes for the detected objects remained longer in the image and the ARV braked longer. As the cooldown was still not high enough, it would need to be increased. On the other hand, some violations (SR3) were caused because the cooldown was, in fact, too high. It appeared that there is a trade-off between SR2 and SR3 violations. We observed that the object detector wrongly classified some objects. The wrong classification could cause that objects, e.g., pedestrians were not recognized as such and the

AEB ADAS did not stop for them, even though it should have stopped. A more sophisticated object detector could be a solution for that. The size of the danger zone could also be adapted, as it was too small and missed one obstacle. Thus, it collided with it.

The sampling period of the monitor is when the monitor expects a sample. We expanded the period by allowing samples every [50,150] ms. We discovered that the sampling period of the monitor (10 Hz) was too high because 39% of samples arrived too early and 61% arrived too late. The elapsed time between two samples was always within [18,562] ms. The goal is to balance this as much as possible and reduce the outliers upwards.

The last aspect that we discovered was that the inputs of the monitor (AEB signal, wheel speed, and front distance measurement) were unsynchronized which caused violations of SR4.

Generally, we found multiple safety issues with the runtime monitors and discovered possible optimizations that could be investigated in the future to increase the safety of the system.

8.2 Threats to Validity

In this section, we describe the threats to validity that we see in the approaches we chose. Further, we explain our efforts to avoid known risks regarding those. We start with the threats to validity regarding our RR. We continue with the decision-making of the tools we chose, and lastly, we describe the threats to validity regarding the implementation of the prototype.

8.2.1 Rapid Review

By conducting a RR instead of a systematic review, we generally have a higher risk of missing relevant papers. The approach of RR is more rigorous, as the duration of RRs is shorter than the duration of a systematic review.

We tried to minimize the risk of missing relevant papers by including four scientific databases as sources and by doing snowballing. While doing snowballing, we chose strict criteria for including the papers which may have lead to missing relevant foundational papers. We chose this approach to only find relevant studies that were, e.g., not published in the chosen scientific databases. We did three iterations of snowballing to collect all papers satisfying our criteria.

The inclusion and exclusion criteria were more restrictive than in systematic reviews as we limited results regarding publication year, language and later also regarding quality and focus of the paper as described in Chapter 5.

In our RR, only one person was responsible for the review, so the results may be biased. We tried to reduce biases by keeping close communication between the reviewer and the supervisors of the bachelor's thesis regarding relevant decision-making during the review, e.g., when choosing the search query or the inclusion and exclusion criteria. Despite the close communication within our team, there is the possibility that we missed papers because of our search query. We tried to minimize the risk of an unsuitable query by trying multiple queries. The quality appraisal of the papers was conducted by the single reviewer with consultation of the supervisors if there were uncertainties.

Generally, we tried to design the RR process as reproducible as possible, by making detailed process descriptions in Figure 5.1 and Figure 5.2. We included interim findings of the review and tried to document the reasons for exclusion of the initial set of base papers in Table 5.1. The synthesis procedure is also a threat to validity. The outputs of the data synthesis were, among others, Table 5.5 and Table 5.6. We left entries with a question mark, if we were unsure how to categorize the data. But there is still the possibility of wrongly categorized entries, as only one person was responsible for it.

8.2.2 Prototype Technique Selection

As well as in the RR process, there is the possibility that we may have misunderstood information from the taxonomy [FKRT21] or from the final papers. Thus, this may have influenced our tool selection. Empty entries in the classification (entries with a question mark in it) may have lead to a different tool selection decision, compared to when all values would have been filled correctly. Additionally, we only chose to include open-source tools into consideration for the monitoring component, which rapidly reduced the amount of possible tools from 16 to six. Another threat to validity is the translation of our hardware and tool requirements (use case) into classification values. Everything was done by one person. Thus, there is a possible bias.

8.2.3 Prototype Implementation

The main threat to validity of the prototype implementation is the lack of experience of working with the robot. Implementation decisions may have been different if a more experienced researcher worked with the robot. E.g., in the implementation itself, we tried to use a newer version of YOLO. We were unable to get results fast enough to match our real-time requirements. Possible causes for the bad performance of newer versions of YOLO lie in our implementation and integration. In the given time, we were not able to improve this to use some newer YOLO version. A similar case was with DeepSORT, as we were not able to implement this on the robot. The conducted experiment was minimal, with one test run and five obstacles.

In this section, we discussed the answers to our research questions and provided our threats to validity. Next, in Chapter 9, we summarize the work done in this thesis. Further, we describe the benefits of this thesis and give insights into the limitations and important lessons learned during the thesis. Finally, we propose future work that can be approached following our thesis.

9 Conclusion and Outlook

In this last section, we provide a conclusion of our work by summarizing its contributions, benefits, and limitations. In addition, we provide an overview of possible future work following our thesis.

9.1 Summary

We provided a classification of 16 papers found with a RR. We chose *rtamt* as tool for the implementation of the prototype. We implemented the prototype and described our architecture. We added three ROS nodes to the already provided nodes on the provided robot. The three nodes were: object tracker, AEB ADAS, and runtime monitors. We implemented four runtime monitors in the runtime monitors node, one for each safety requirement we defined earlier. For each safety requirement, we specified a corresponding STL formula. We verified the implementation of our monitors. In an experiment, we tested the implementation. During the evaluation of the experiment, we discovered that several optimizations could be done to improve the prototype. Additionally, we discovered a trade-off between two types of violations. In example, increasing or decreasing the cooldown lead to the one type of violations to become more frequent and the other type of violations to become less frequent. The object detector sometimes caused unwanted behavior that could be safety critical if this had been a real-world setting. We also identified outliers upwards, with samples being delivered to the monitors only every 562 ms. The goal is to balance the sample periods as much as possible and reduce the outliers upwards. Generally, we found multiple safety issues with the runtime monitors and discovered possible optimizations that could be investigated in the future to increase the safety of the system.

9.2 Benefits

As we discovered multiple safety issues and optimizations using RM, our results are beneficial to everyone who wants to build safe ADAS perception systems. RM was identified as a useful technique for gaining additional insights on certain specified properties (e.g., safety requirements). Therefore, it is useful for debugging and testing of ADAS systems.

9.3 Limitations

Although the used RR approach was very efficient, it introduced a significant risk of missing relevant papers because of its shortened duration and stricter inclusion and exclusion criteria compared to a systematic review. Also, a systematic review is usually not performed by a single person. We attempted to keep these risks at a minimum by using multiple scientific databases, trying out different queries, performing snowballing and ensuring close communication within our team. The selection of a suitable technique for our prototype introduces additional limitations due to the restriction to open-source tools only and wrong or missing labels caused by misunderstandings of the applied taxonomy by Falcone et al. [FKRT21]. Additionally, the correct implementation of our prototype is threatened by a general lack of experience with the *Mecabot TX* robot environment. In example, the limited available time did not allow us to upgrade our operating system and to use newer versions of certain tools (e.g., YOLO). Further, we only performed one minimal experiment to evaluate our implementation.

9.4 Lessons Learned

The main lesson we learned is how important good literature reviews are. As our whole work was depending on the results of the RR, we were: firstly blocked for some time as all the steps of the work (RR, tool selection, and implementation) had to be done mostly sequentially. Secondly, the classification of the papers was of extreme relevance for the steps after. Ideally, the classification is done by a team or by the authors themselves discussing their thoughts and comparing their results. This was not the case here, as only one person did the classification. In the experiment, it was difficult to do multiple tasks as one person. From setting the experiment up, to starting the robot and simultaneously filming multiple perspectives of the experiment. The quality of the video recordings was accordingly bad. Luckily, we still got useful information from the videos and screen recordings for the evaluation.

9.5 Future Work

We have several suggestions for future work. We separate our ideas into two parts: a RR part and an implementation part.

9.5.1 Rapid Review

There are several possibilities to improve the search query we used in the RR. In example, we would add “runtime safety monitor*” and “run-time safety monitor*” to the “RM” part of the query. We would also replace all occurrences of “monitoring” with “monitor*”. For the “ADAS” part of this thesis, we can imagine adding “automotive driv*”. To make the query more general, words like: “cyber physical system”, “cyber-physical-system”, “autonomous system”, and “automotive” can be added. In our case, this was not necessary, but this may be interesting for other research. Conducting a RR instead of a systematic review involves increasing the threats to validity, as we mention in Chapter 8. Therefore, conducting a systematic review can decrease the threats to validity of the results. Therefore, an additional idea is to contact the authors in the scope of a systematic review to validate our classification of the tools and also to fill the open classification values (the “?”) in our results. The similarities of our papers were calculated in a simple fashion. An alternative approach would be to also separate values and compare them. E.g, if we have one paper with value “x,y” and another paper with value “x”, then they do not have similarity with our approach as we currently require identical values. This could be more loose, allowing identical parts of the values to have an impact on the total similarity. Furthermore, in the tool classification results, we could add the programming language, that a tool was written in. This would allow to easily filter the tools for compatibility with a certain environment.

9.5.2 Prototype Implementation

Firstly, all the unexpected behaviors that occurred during the experiment could be further investigated. From changing and testing different sample periods to fixing the synchronization issues of the runtime monitor, and investigating how to choose the cooldown. Additionally, the newer YOLO versions could be tested. Besides that, we chose constant values for the speed and safety distance. This could be more complex and realistic in future work with, e.g., a safety distance formula. Additional experiments would further test the implementation. For instance, we could consider using different obstacles, and multiple rooms with different lighting conditions. We also imagine adding additional monitors to check other properties of the detected objects. For example, we could check the plausibility of the bounding box transitions or the consistency of detected COCO object classes, as shown by Balakrishnan et al. [BDH+21].

In conclusion, our extensive exploration and successful implementation of multiple runtime monitors for an exemplary AD use case showcase the effectiveness of RM for improving the safety of ADAS applications. Therefore, our work serves as a solid foundation for future research on the use of RM in the area of AD.

Bibliography

[AĐHM23] M. Albonico, M. Đorđević, E. Hamer, I. Malavolta. “Software engineering research on the Robot Operating System: A systematic mapping study”. In: *Journal of Systems and Software* 197 (2023). ISSN: 0164-1212. DOI: 10.1016/j.jss.2022.111574 (cit. on pp. 24, 25).

[ALZ+20] D. An, J. Liu, M. Zhang, X. Chen, M. Chen, H. Sun. “Uncertainty modeling and runtime verification for autonomous vehicles driving control: A machine learning-based approach”. In: *JOURNAL OF SYSTEMS AND SOFTWARE* 167 (Sept. 2020). ISSN: 0164-1212. DOI: 10.1016/j.jss.2020.110617 (cit. on pp. 38, 41, 42, 45, 46, 50, 56).

[AM18] M. A. Akbar, S. Mandala. “IoT on Heart Arrhythmia Real Time Monitoring”. In: *Indonesian Journal on Computing (Indo-JC)* 3 (2 Sept. 2018), pp. 1–10. DOI: 10.21108/indojc.2018.3.2.170 (cit. on p. 38).

[ami10] amitp. *Solar powered stop sign*. This work is licensed under the CC BY-SA 2.0 license. To view a copy of this license, visit https://creativecommons.org/licenses/by-sa/2.0/. 2010. URL: https://openverse.org/image/8ea98097-14f4-442a-98d2-a8850abff76d?q=stop%20sign (cit. on pp. 66, 95).

[AR18] H. Aspriyono, R. Riska. “APLIKASI REAL-TIME MONITORING KEHADIRAN KARYAWAN TERINTEGRASI DENGAN FINGERPRINT SYSTEM PADA UNIVERSITAS DEHASEN BENGKULU”. In: *ILKOM Jurnal Ilmiah* 10 (3 Dec. 2018), pp. 260–266. DOI: 10.33096/ilkom.v10i3.352.260-266 (cit. on p. 38).

[Bau22] A. Bauder. *Toter Winkel bei Lkw, Bus und Pkw: Das müssen Sie wissen*. Allgemeiner Deutscher Automobil-Club e.V. (ADAC). Aug. 2022. URL: https://www.adac.de/verkehr/verkehrssicherheit/gefahrensituation/toter-winkel/ (cit. on p. 17).

[BDH+21] A. Balakrishnan, J. Deshmukh, B. Hoxha, T. Yamaguchi, G. Fainekos. “PerceMon: Online Monitoring for Perception Systems”. In: *Runtime Verification*. Ed. by L. Feng, D. Fisman. Cham: Springer International Publishing, 2021, pp. 297–308. ISBN: 978-3-030-88494-9. DOI: 10.48550/arXiv.2108.08289 (cit. on pp. 40–42, 45, 46, 50, 56, 73, 83).

[BFFR18] E. Bartocci, Y. Falcone, A. Francalanza, G. Reger. “Introduction to Runtime Verification”. In: *Lectures on Runtime Verification: Introductory and Advanced Topics*. Ed. by E. Bartocci, Y. Falcone. Cham: Springer International Publishing, 2018, pp. 1–33. DOI: 10.1007/978-3-319-75632-5_1 (cit. on pp. 18, 19, 22–24).

[BGO+16] A. Bewley, Z. Ge, L. Ott, F. Ramos, B. Upcroft. “Simple online and realtime tracking”. In: *2016 IEEE International Conference on Image Processing (ICIP)*. 2016, pp. 3464–3468. DOI: 10.1109/ICIP.2016.7533003 (cit. on p. 27).

[Bje18] M. Bjelonic. *YOLO ROS: Real-Time Object Detection for ROS*. GitHub. 2016–2018. URL: https://github.com/leggedrobotics/darknet_ros (cit. on pp. 24, 27, 59).

[BY90] T. Benaya, A. Yehudai. "An application generator for a family of real-time monitor and control systems". In: *COMPEURO'90: Proceedings of the 1990 IEEE International Conference on Computer Systems and Software Engineering - Systems Engineering Aspects of Complex Computerized Systems*. IEEE Comput. Soc. Press, 1990, pp. 274–279. DOI: 10.1109/cmpeur.1990.113635 (cit. on p. 38).

[Cam23] Cambia. *The Lens - Free & Open Patent and Scholarly Search*. 2023. URL: https://www.lens.org/ (cit. on p. 34).

[Che21] C.-H. Cheng. "Provably-Robust Runtime Monitoring of Neuron Activation Patterns". In: IEEE, 2021, pp. 1310–1313. ISBN: 978-3-9819263-5-4. DOI: 10.23919/DATE51398.2021.9473957 (cit. on pp. 38, 41, 42, 45, 46, 50, 56).

[CL18] C. Colombo, M. Leucker. *Runtime Verification: 18th International Conference, RV 2018, Limassol, Cyprus, November 10–13, 2018, Proceedings*. Lecture Notes in Computer Science. Springer Verlag, Nov. 2018. ISBN: 978-303003768-0. DOI: 10.1007/978-3-030-03769-7 (cit. on pp. 29, 38).

[Cla23] Clarivate group. *Web of Science*. 2023. URL: https://www.webofscience.com/ (cit. on p. 34).

[CPS20] B. Cartaxo, G. Pinto, S. Soares. "Rapid Reviews in Software Engineering". In: *Contemporary Empirical Methods in Software Engineering*. Ed. by M. Felderer, G. H. Travassos. Cham: Springer International Publishing, 2020, pp. 357–384. ISBN: 978-3-030-32489-6. DOI: 10.1007/978-3-030-32489-6_13 (cit. on pp. 31, 33, 34).

[DADF18] A. Dokhanchi, H. B. Amor, J. V. Deshmukh, G. Fainekos. "Evaluating Perception Systems for Autonomous Vehicles Using Quality Temporal Logic". In: *Runtime Verification*. Ed. by C. Colombo, M. Leucker. Cham: Springer International Publishing, 2018, pp. 409–416. ISBN: 978-3-030-03769-7. DOI: 10.1007/978-3-030-03769-7_23 (cit. on pp. 40–42, 45, 46, 50, 56).

[DLP+87] R. D'Ippolito, K. Lee, C. Plinta, M. Rissman, R. Van Scoy. *Prototype real-time monitor: Requirements*. Tech. rep. Technical Report CMU/SEI-87-TR-36, Software Engineering Institute, Nov. 1987. URL: https://apps.dtic.mil/sti/citations/ADA188929 (cit. on p. 38).

[DN20] J. Deshmukh, D. Ničković. *Runtime Verification: 20th International Conference, RV 2020, Los Angeles, CA, USA, October 6–9, 2020, Proceedings*. Vol. 12399. Springer Nature, Oct. 2020. ISBN: 978-3-030-60508-7. DOI: 10.1007/978-3-030-60508-7 (cit. on pp. 29, 38).

[DTCL21] X. Du, A. Tiu, K. Cheng, Y. Liu. "Trace-Length Independent Runtime Monitoring of Quantitative Policies". In: *IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING* 18 (3 May 2021), pp. 1489–1510. ISSN: 1545-5971. DOI: 10.1109/TDSC.2019.2919693 (cit. on pp. 29, 38, 41, 42, 45, 46, 50, 56).

[Eco14] Economic Commission for Europe of the United Nations (UN/ECE). *Regulation No 131 of the Economic Commission for Europe of the United Nations (UN/ECE) — Uniform provisions concerning the approval of motor vehicles with regard to the Advanced Emergency Braking Systems (AEBS)*. July 2014. URL: http://data.europa.eu/eli/reg/2014/131/oj (cit. on p. 21).

[Els23] Elsevier B.V. *Scopus*. 2023. URL: https://www.scopus.com/ (cit. on p. 34).

[Eur21] European Commission. *Road safety thematic report – Advanced driver assistance systems*. European Road Safety Observatory. Brussels, European Commission, Directorate General for Transport., 2021. URL: https://road-safety.transport.ec.europa.eu/system/files/2022-04/Road_Safety_Thematic_Report_ADAS_2021.pdf (cit. on pp. 17, 19, 21, 22).

[FKRT18] Y. Falcone, S. Krstić, G. Reger, D. Traytel. "A Taxonomy for Classifying Runtime Verification Tools". In: *Runtime Verification*. Ed. by C. Colombo, M. Leucker. Cham: Springer International Publishing, 2018, pp. 241–262. ISBN: 978-3-030-03769-7. DOI: 10.1007/978-3-030-03769-7_14 (cit. on pp. 29, 44).

[FKRT21] Y. Falcone, S. Krstić, G. Reger, D. Traytel. "A taxonomy for classifying runtime verification tools". In: *International Journal on Software Tools for Technology Transfer* 23.2 (May 2021), pp. 255–284. ISSN: 1433-2787. DOI: 10.1007/s10009-021-00609-z (cit. on pp. 29–31, 40, 43, 44, 54, 77, 80, 82).

[GKS+22] D. Grundt, A. Köhne, I. Saxena, R. Stemmer, B. Westphal, E. Möhlmann. "Towards Runtime Monitoring of Complex System Requirements for Autonomous Driving Functions". In: vol. 371. 2022, pp. 53–61. DOI: 10.4204/EPTCS.371.4 (cit. on p. 38).

[Goo23] Google LLC. *Google Scholar*. 2023. URL: https://scholar.google.com/ (cit. on pp. 33, 39).

[GZWR20] J. Grieser, M. Zhang, T. Warnecke, A. Rausch. "Assuring the Safety of End-to-End Learning-Based Autonomous Driving through Runtime Monitoring". In: ed. by A. Trost, A. Zemva, A. Skavhaug. IEEE COMPUTER SOC, 2020, pp. 476–483. ISBN: 978-1-7281-9535-3. DOI: 10.1109/DSD51259.2020.00081 (cit. on pp. 38, 41, 42, 45, 46, 50, 56).

[Han22] H. Hanifadinna. "Pembuatan Sistem Real Time Monitoring Pengukur Oil Layer Pada Vertical Continuous Tank di Pabrik Kelapa Sawit Pekawai Kalimantan Barat". In: *JURNAL VOKASI TEKNOLOGI INDUSTRI (JVTI)* 4 (1 June 2022), pp. 1–10. URL: https://journal.ugm.ac.id/v3/JNTETI/article/view/2998 (cit. on pp. 38, 41, 42, 45, 46, 50, 56).

[Hil23] P. Hilton. *Process Milestones for Workflow Execution Status Visibility*. 2023. URL: https://www.signavio.com/post/process-milestones/ (cit. on pp. 33, 36, 37).

[Hun12] C. Hunkeler. *Woman Cyclist Gives Thumbs Up*. This work is licensed under the CC BY-SA 2.0 license. To view a copy of this license, visit https://creativecommons.org/licenses/by-sa/2.0/. 2012. URL: https://openverse.org/image/5aa66793-532c-47a1-9e59-61642c357824?q=cyclist (cit. on pp. 66, 96, 97).

[ily20] ilyasmg. *sort-deepsort-yolov3-ROS*. GitHub. 2019–2020. URL: https://github.com/ilyasmg/sort-deepsort-yolov3-ROS (cit. on p. 27).

[Ins23] Institute of Electrical and Electronics Engineers (IEEE). *IEEE Xplore*. 2023. URL: https://ieeexplore.ieee.org/ (cit. on p. 34).

[KA21] W. Kontar, S. Ahn. *Real-time Monitoring of Autonomous Vehicle's Time Gap Variations: A Bayesian Framework*. Jan. 2021. DOI: 10.48550/arXiv.2102.00375 (cit. on p. 38).

[KCDK15] A. Kane, O. Chowdhury, A. Datta, P. Koopman. “A case study on runtime monitoring of an autonomous research vehicle (ARV) system”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 9333 (2015), pp. 102–117. DOI: 10.1007/978-3-319-23820-3_7 (cit. on pp. 18, 38, 41, 42, 45, 46, 50, 56).

[Kec13] Kecko. *Swiss Crosswalk*. This work is licensed under the CC BY-SA 2.0 license. To view a copy of this license, visit https://creativecommons.org/licenses/by-sa/2.0/. 2013. URL: https://openverse.org/image/c6ca8537-b60c-4178-a5eb-22a6ae9ad129?q=crosswalk (cit. on pp. 66, 97–99).

[mal07] malouette. *Crosswalk Conversations*. This work is licensed under the CC BY-SA 2.0 license. To view a copy of this license, visit https://creativecommons.org/licenses/by-sa/2.0/. 2007. URL: https://openverse.org/image/717e7e97-e6d0-48c1-b2de-9f46bfb27327?q=crosswalk (cit. on pp. 66, 95, 96).

[MČSP22] A. Mehmed, A. Čaušević, W. Steiner, S. Punnekkat. “Early Concept Evaluation of a Runtime Monitoring Approach for Safe Automated Driving”. In: *2022 IEEE Zooming Innovation in Consumer Technologies Conference (ZINC)*. 2022, pp. 53–58. DOI: 10.1109/ZINC55034.2022.9840649 (cit. on pp. 38, 41, 42, 45, 46, 50, 56).

[MHR16] M. Mauritz, F. Howar, A. Rausch. “Assuring the safety of advanced driver assistance systems through a combination of simulation and runtime monitoring”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 9953 LNCS (2016), pp. 672–687. DOI: 10.1007/978-3-319-47169-3_52 (cit. on pp. 38, 41, 42, 45, 46, 50, 56).

[MMGP19] E. Marti, M. A. de Miguel, F. Garcia, J. Perez. “A Review of Sensor Technologies for Perception in Automated Driving”. In: *IEEE Intelligent Transportation Systems Magazine* 11.4 (2019), pp. 94–108. DOI: 10.1109/MITS.2019.2907630 (cit. on pp. 17, 52).

[MPPP14] A. de Matos Pedro, D. Pereira, L. M. Pinho, J. S. Pinto. “Towards a Runtime Verification Framework for the Ada Programming Language”. In: *Reliable Software Technologies – Ada-Europe 2014*. Ed. by L. George, T. Vardanega. Cham: Springer International Publishing, 2014, pp. 58–73. ISBN: 978-3-319-08311-7. DOI: 10.1007/978-3-319-08311-7_6 (cit. on p. 38).

[MPPP15] A. de Matos Pedro, D. Pereira, L. M. Pinho, J. S. Pinto. *Formal Contracts for Runtime Verification Support in the Ada Programming Language*. Mar. 2015. URL: http://hdl.handle.net/10400.22/6802 (cit. on p. 38).

[MRS14] M. Mauritz, A. Rausch, I. Schaefer. “Dependable ADAS by combining design time testing and runtime monitoring”. In: 2014, pp. 28–37. URL: https://www.researchgate.net/publication/278684216_Dependable_ADAS_by_Combining_Design_Time_Testing_and_Runtime_Monitoring (cit. on p. 38).

[NY20] D. Ničković, T. Yamaguchi. “RTAMT: Online Robustness Monitors from STL”. In: *Automated Technology for Verification and Analysis*. Ed. by D. V. Hung, O. Sokolsky. Cham: Springer International Publishing, 2020, pp. 564–571. ISBN: 978-3-030-59152-6. DOI: 10.1007/978-3-030-59152-6_34 (cit. on pp. 23, 24, 26, 56, 74).

[Obj23] Object Management Group, Inc. *BPMN Specification - Business Process Model and Notation*. 2023. URL: https://www.bpmn.org/ (cit. on p. 33).

[OKS19] M. H. Osman, S. Kugele, S. Shafaei. “Run-Time Safety Monitoring Framework for AI-Based Systems: Automated Driving Cases”. In: *2019 26th Asia-Pacific Software Engineering Conference (APSEC)*. 2019, pp. 442–449. DOI: 10.1109/APSEC48747.2019.00066 (cit. on pp. 40–42, 45, 46, 50, 56).

[Ope17] Open Robotics. *rospy*. This work is licensed under the CC BY 3.0 license. To view a copy of this license, visit https://creativecommons.org/licenses/by/3.0/. 2017. URL: https://wiki.ros.org/rospy (cit. on p. 56).

[Ope22] Open Robotics. *ROS/Concepts*. This work is licensed under the CC BY 3.0 license. To view a copy of this license, visit https://creativecommons.org/licenses/by/3.0/. 2022. URL: https://wiki.ros.org/ROS/Concepts (cit. on p. 25).

[Ope23] Open Robotics. *ROS 2 Documentation: Galactic - Installation*. 2023. URL: https://docs.ros.org/en/galactic/Installation.html (cit. on pp. 68, 71).

[Pam15] J. Pamungkas. “Desain Real-Time Monitoring Berbasis Wireless Sensor Network Upaya Mitigasi Bencana Erupsi Gunungapi”. In: *Jurnal Nasional Teknik Elektro dan Teknologi Informasi (JNTETI)* 4 (3 Dec. 2015), pp. 177–181. DOI: 10.22146/jnteti.v4i3.160 (cit. on p. 38).

[PHCR20] F. Pop, B. Herrera, C. Cassella, M. Rinaldi. “Enabling Real-Time Monitoring of Intrabody Networks Through the Acoustic Discovery Architecture”. In: *IEEE transactions on ultrasonics, ferroelectrics, and frequency control* 67 (11 June 2020), pp. 2336–2344. DOI: 10.1109/tuffc.2020.3002973 (cit. on p. 38).

[QSA20] W. Qi, H. Su, A. Aliverti. “A Smartphone-Based Adaptive Recognition and Real-Time Monitoring System for Human Activities”. In: *IEEE Transactions on Human-Machine Systems* 50 (5 2020), pp. 414–423. DOI: 10.1109/thms.2020.2984181 (cit. on p. 38).

[RCG22] R. Roriz, J. Cabral, T. Gomes. “Automotive LiDAR Technology: A Survey”. In: *IEEE Transactions on Intelligent Transportation Systems* 23.7 (July 2022), pp. 6282–6297. DOI: 10.1109/TITS.2021.3086804 (cit. on p. 21).

[RCL+18] J. Rufino, A. Casimiro, A. Lopes, F. Singhoff, S. Rubini, V.-A. Nicolas, M. Lallali, M. Dridi, J. Boukhobza, L. Allache. “NORTH - Non-intrusive observation and Runtime Verification of cyber-physical systems”. In: *Ada User Journal* 39 (4 2018), pp. 278–281. URL: https://www.di.fc.ul.pt/~casim/papers/auj18-NORTH/auj18-NORTH.pdf (cit. on pp. 38, 41, 42, 45, 46, 50, 56).

[RDGF16] J. Redmon, S. Divvala, R. Girshick, A. Farhadi. “You Only Look Once: Unified, Real-Time Object Detection”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016, pp. 779–788. DOI: 10.1109/CVPR.2016.91 (cit. on p. 26).

[RF18] J. Redmon, A. Farhadi. “YOLOv3: An Incremental Improvement”. In: *arXiv* (2018). DOI: 10.48550/arXiv.1804.02767 (cit. on p. 24).

[Rob23a] RobotShop Europe. *Mecabot TX*. 2023. URL: https://eu.robotshop.com/products/mecabot-tx (cit. on pp. 51, 57).

[Rob23b] Roboworks. *Roboworks: Robot Educational Programable Mobile Robots*. 2023. URL: https://www.roboworks.net/ (cit. on p. 71).

[Rob23c] Roboworks. *Wheelbots*. 2023. URL: https://www.roboworks.net/store/wheelbots (cit. on p. 71).

[RS11] V. V. Rubanov, E. A. Shatokhin. "Runtime Verification of Linux Kernel Modules Based on Call Interception". In: Mar. 2011, pp. 180–189. DOI: 10.1109/ICST.2011.20 (cit. on p. 38).

[SAE21a] SAE International. *SAE J3016^{TM} LEVELS OF DRIVING AUTOMATION^{TM}*. Pdf found at: https://www.sae.org/standards/content/j3016_202104/ under "Related Items", named: "SAE J3016 Visual Chart". 2021. URL: https://www.sae.org/binaries/content/assets/cm/content/blog/sae-j3016-visual-chart_5.3.21.pdf (cit. on pp. 17, 19, 20).

[SAE21b] SAE On-Road Automated Driving (ORAD) Committee. *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*. SAE International, Apr. 2021. DOI: 10.4271/J3016_202104 (cit. on pp. 17, 19, 20).

[Sco87a] R. V. Scoy. *Prototype Real-Time Monitor: Ada Code*. Nov. 1987. URL: https://apps.dtic.mil/sti/tr/pdf/ADA191095.pdf (cit. on p. 38).

[Sco87b] R. V. Scoy. *Prototype Real-Time Monitor. Executive Summary*. Nov. 1987. URL: https://apps.dtic.mil/sti/citations/ADA188928 (cit. on p. 38).

[SL17] M. Seo, R. Lysecky. "Hierarchical Non-intrusive In-situ Requirements Monitoring for Embedded Systems". In: *Runtime Verification*. Ed. by S. Lahiri, G. Reger. Cham: Springer International Publishing, 2017, pp. 259–276. ISBN: 978-3-319-67531-2. DOI: 10.1007/978-3-319-67531-2_16 (cit. on pp. 40–42, 45, 46, 50, 56).

[SM93] S. Sankar, M. Mandal. "Concurrent runtime monitoring of formally specified programs". In: *Computer* 26 (3 1993), pp. 32–41. DOI: 10.1109/2.204684 (cit. on p. 38).

[Spr23a] Springer Nature. *Lecture Notes in Computer Science*. 2023. URL: https://www.springer.com/gp/computer-science/lncs (cit. on p. 29).

[Spr23b] Springer Nature Switzerland AG. Part of Springer Nature. *International Conference on Runtime Verification*. 2023. URL: https://link.springer.com/conference/rv (cit. on p. 29).

[Sta23a] Statistisches Bundesamt (Destatis). *Accidents registered by the police: specification*. Nov. 2023. URL: https://www.destatis.de/EN/Themes/Society-Environment/Traffic-Accidents/Tables/accidents-casualties.html (cit. on p. 17).

[Sta23b] Statistisches Bundesamt (Destatis). *Driver-related causes of accidents involving personal injury*. 2023. URL: https://www.destatis.de/EN/Themes/Society-Environment/Traffic-Accidents/Tables/driver-mistakes.html (cit. on p. 17).

[Sta23c] Statistisches Bundesamt (Destatis). *General causes of accidents*. Nov. 2023. URL: https://www.destatis.de/EN/Themes/Society-Environment/Traffic-Accidents/Tables/causes-accidents-personal-injury2.html (cit. on p. 17).

[SUPR20] S. Shankar, V. R. Ujwal, S. Pinisetty, P. Roop. "Formal runtime monitoring approaches for autonomous vehicles". In: vol. 2785. 2020, pp. 89–94. URL: https://ceur-ws.org/Vol-2785/paper15.pdf (cit. on pp. 38, 41, 42, 45, 46, 50, 56).

[SW10] "TF–IDF". In: *Encyclopedia of Machine Learning*. Ed. by C. Sammut, G. I. Webb. Boston, MA: Springer US, 2010, pp. 986–987. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_832 (cit. on pp. 39, 41).

[TBCM21] V. Tripathi, L. Ballotta, L. Carlone, E. Modiano. “WiOpt - Computation and Communication Co-Design for Real-Time Monitoring and Control in Multi-Agent Systems”. In: IEEE, Oct. 2021, pp. 65–72. DOI: 10.23919/wiopt52861.2021.9589966 (cit. on p. 38).

[The23a] The Autoware Foundation. *Autoware Documentation - Installation*. GitHub. 2023. URL: https://autowarefoundation.github.io/autoware-documentation/main/installation/ (cit. on p. 71).

[The23b] The Autoware Foundation. *Autoware Documentation - Source installation*. GitHub. 2023. URL: https://autowarefoundation.github.io/autoware-documentation/main/installation/autoware/source-installation/ (cit. on p. 71).

[The23c] The Autoware Foundation. *Autoware - the world’s leading open-source software project for autonomous driving*. GitHub. 2021–2023. URL: https://github.com/autowarefoundation/autoware (cit. on p. 71).

[TNM+19] H.-D. Tran, L. V. Nguyen, P. Musau, W. Xiang, T. T. Johnson. “Real-time verification for distributed cyber-physical systems”. In: *arXiv preprint arXiv:1909.09087* (Sept. 2019). DOI: 10.48550/arxiv.1909.09087 (cit. on p. 38).

[VD09] G. Van Rossum, F. L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN: 1441412697 (cit. on p. 56).

[WB18] N. Wojke, A. Bewley. “Deep Cosine Metric Learning for Person Re-identification”. In: *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2018, pp. 748–756. DOI: 10.1109/WACV.2018.00087 (cit. on p. 27).

[WBP17] N. Wojke, A. Bewley, D. Paulus. “Simple Online and Realtime Tracking with a Deep Association Metric”. In: *2017 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2017, pp. 3645–3649. DOI: 10.1109/ICIP.2017.8296962 (cit. on p. 27).

[WKLS18] K. Watanabe, E. Kang, C.-W. Lin, S. Shiraishi. “INVITED: Runtime Monitoring for Safety of Intelligent Vehicles”. In: *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. June 2018, pp. 1–6. DOI: 10.1109/DAC.2018.8465912 (cit. on pp. 38, 41, 42, 45, 46, 50, 56).

[Woh14] C. Wohlin. “Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering”. In: *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*. EASE ’14. London, England, United Kingdom: Association for Computing Machinery, 2014. ISBN: 9781450324762. DOI: 10.1145/2601248.2601268 (cit. on p. 39).

[XLX+06] L. Xuandong, W. Linzhang, Q. Xiaokang, L. Bin, Y. Jiesong, Z. Jianhua, Z. Guoliang. *Ada-Europe - Runtime verification of java programs for scenario-based specifications*. Springer Berlin Heidelberg, 2006, pp. 94–105. DOI: 10.1007/11767077_8 (cit. on p. 38).

[YCP22] S. Yu, W. Chen, H. V. Poor. “TSI-Aided Real-Time Monitoring of Brownian Motions: A Rate-Latency-Distortion Perspective”. In: IEEE, 2022, pp. 5342–5347. ISBN: 978-1-6654-3540-6. DOI: 10.1109/GLOBECOM48099.2022.10001271 (cit. on p. 38).

[YCP23] S. Yu, W. Chen, H. V. Poor. “Real-Time Monitoring With Timing Side Information”. In: *IEEE TRANSACTIONS ON COMMUNICATIONS* 71 (4 Apr. 2023), pp. 1953–1969. ISSN: 0090-6778. DOI: 10.1109/TCOMM.2023.3239514 (cit. on p. 38).

[Yuf19] A. Yufiyanto. *RANCANG BANGUN SISTEM REAL TIME MONITORING GAS BERBAHAYA PADA PETERNAKAN AYAM BROILER BERBASIS INTERNET OF THINGS DAN DATA LOGGER*. 2019. URL: https://lens.org/011-562-388-952-255 (cit. on p. 38).

[ZBK20] E. Zapridou, E. Bartocci, P. Katsaros. "Runtime Verification of Autonomous Driving Systems in CARLA". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 12399 LNCS (2020), pp. 172–183. DOI: 10.1007/978-3-030-60508-7_9 (cit. on pp. 19, 38, 41, 42, 45, 46, 50, 56).

All links were last followed on December 11, 2023.

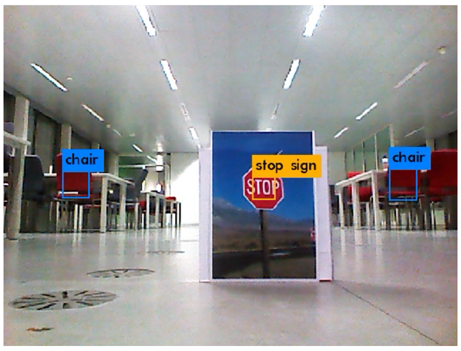
A Appendix

```python
# List of words to be ignored
ignore_words = ['doi', 'ieee', 'vol', 'proposed', 'work', 'figure', 'fig', 'authorized',
'licensed', 'use', 'restrictions', 'may/june', 'apply', 'xplore', 'utc', '12:08:36',
'12:04:40', 'downloaded', 'july', '21,2023', 'august', '25,2023', '10:42:32',
'stuttgart', 'universitaetsbibl', 'limited', 'international', 'conference', 'd.an',
'al./the', 'journal', 'elsevier', 'page', 'may', 'https', 'org', '10', '1007', '978',
'//doi.org/10.1109/tcomm.2023.3239514']

# List of punctuations to be ignored
punctuations = ['(',')',';',':','[',']',',','.','--','-','#','!','*','”','%']

# Get the stopwords list to be ignored; stopwords can be found here:
# https://github.com/nltk/nltk_data/blob/gh-pages/packages/corpora/stopwords.zip
stop_words = stopwords.words('english')

# Convert ignore words from user to lower case
ignore_words_lower = [x.lower() for x in ignore_words]

# Combine all the words to be ignored
all_ignored_words = punctuations + stop_words + ignore_words_lower

# Keyword fix, when letters could not be read correctly
fix_keywords = { 'verication': 'verification', 'trafc': 'traffic', 'trac': 'traffic' }

# Keywords that should not be singularized
do_not_singularize = { 'autonomous', 'data' }

```

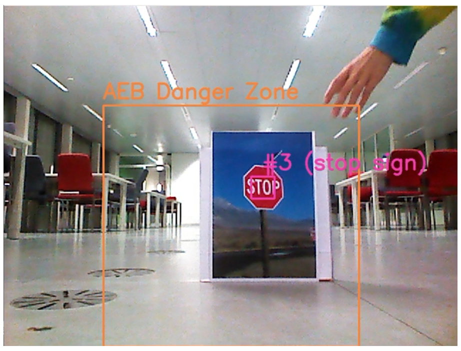
Listing A.1: Ignored words and punctuations in the source papers

```python
import pandas as pd

# read file and rename columns
# used spreadsheet file represents Table 5.5 + focus and communication columns from Table 5.6
df = pd.read_excel('/content/Taxonomy_Data_Extraction_Form_simplyfied.xlsx',
    names=["paper_id", "specification_organization", "specification_behavior_implicit",
    "specification_behavior_explicit_data", "specification_behavior_explicit_output_information",
    "specification_behavior_explicit_output_frequency", "specification_behavior_explicit_time_logical",
    "specification_behavior_explicit_time_physical", "specification_behavior_explicit_modality",
    "specification_behavior_explicit_paradigm", "monitor_decision_procedure_realisation",
    "monitor_decision_procedure_tool", "monitor_decision_procedure_properties",
    "monitor_generation", "monitor_execution", "deployment_stage_offline", "deployment_stage_online",
    "deployment_architecture", "deployment_instrumentation", "interference", "reaction_active",
    "reaction_passive", "trace_information", "trace_sampling", "trace_evaluation", "trace_precision",
    "trace_model", "ADAS_application_focus", "ADAS_application_communication" ])

# fill in missing information on whether the tool is open source or not
df["monitor_decision_procedure_tool_open_source"] = df["monitor_decision_procedure_tool"]
    .apply(lambda t: t in ["easy-rte", "rtamt", "UPPAAL-SMC", "Breach", "PerceMon",
    "Persephone based on S-TaLiRo"])

# define the filter criteria
rm_technique_filter = ' and '.join([
    "monitor_decision_procedure_tool_open_source == True",
    "specification_behavior_explicit_output_information.str.contains('v')",
    "specification_behavior_explicit_time_physical not in ['none', '?']",
    "specification_behavior_explicit_modality.str.contains('f')",
    "deployment_stage_online.str.contains('outline')",
    "deployment_architecture == 'c'",
    "deployment_instrumentation == 'none'",
    "reaction_passive.str.contains('so')",
    "trace_precision == 'p'",
    "trace_model == 'i'",
    "ADAS_application_focus.str.contains('pv')"
])

# apply the filter criteria to the data
df_filtered = df.query(rm_technique_filter)
df_filtered
```

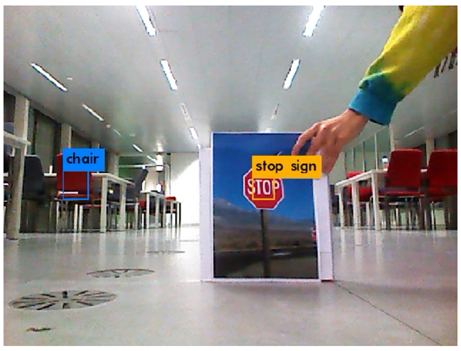
Listing A.2: Tool selection code



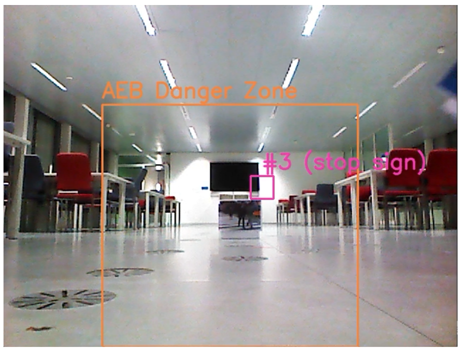
(a) The object detector recognizes stop sign and chairs



(b) The AEB ADAS detects stop sign (ID #3) in the AEB danger zone (larger rectangle)

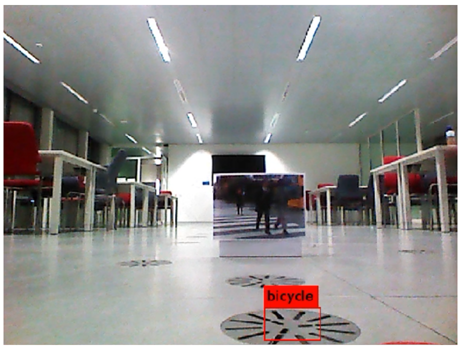


(c) The object detector recognizes a chair and the stop sign while AEB ADAS is cooling down the stop sign

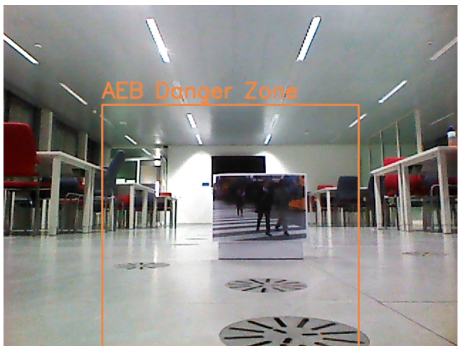


(d) The AEB ADAS recognizes the stop sign a few moments after it is removed (in the distance is the second obstacle [mal07])

Figure A.1: Screenshots of object detector (left) and AEB ADAS (right) camera images during experiment - related to the first obstacle (obstacle image [ami10])

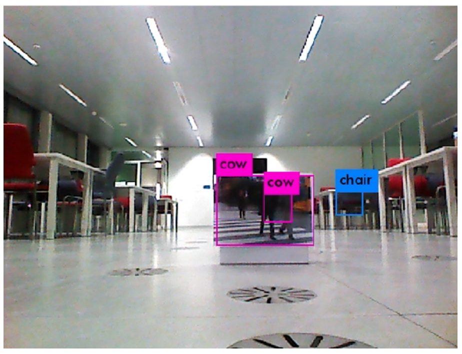


(a) The object detector recognizes bicycle in the floor

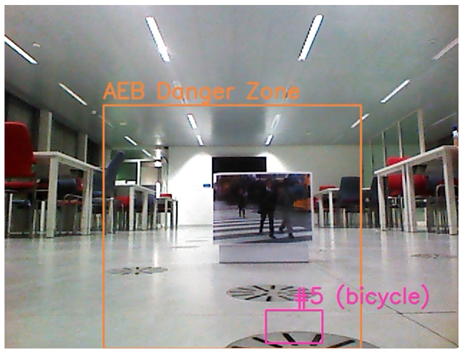


(b) The AEB ADAS recognizes no objects in the frame contrary to Figure A.2a

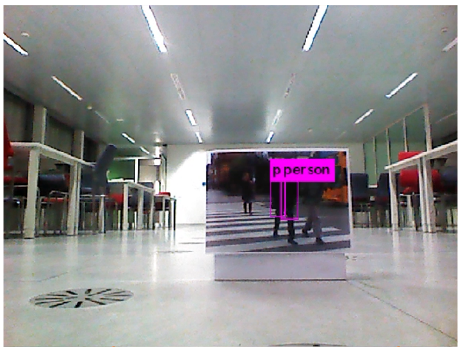
Figure A.2: Screenshots of object detector (left) and AEB ADAS (right) camera images during experiment - related to the second obstacle (obstacle image [mal07]) - Part 1



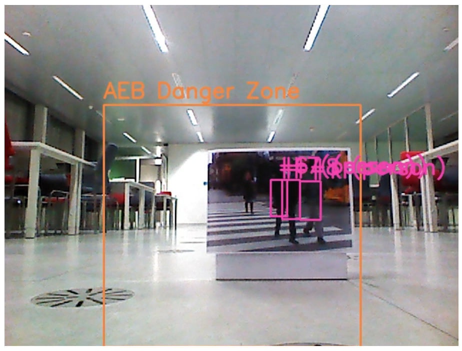
(a) The object detector recognizes the obstacle as cows as the obstacle is still far away. It also recognizes a chair



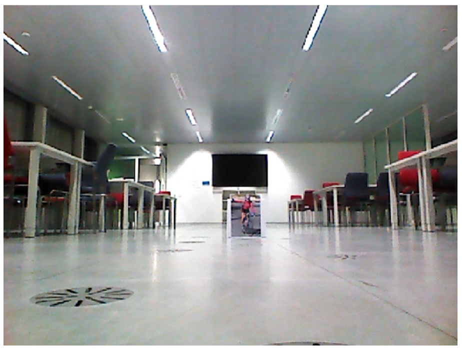
(b) The AEB ADAS processes the bicycle from Figure A.2a as object (ID #5) in the AEB danger zone (larger rectangle)



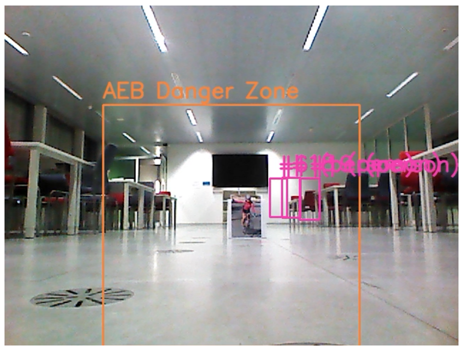
(c) The object detector recognizes the persons in the obstacle contrary to Figure A.3a



(d) The AEB ADAS recognizes the obstacle (mutiple persons with multiple IDs) in the AEB danger zone (larger rectangle)

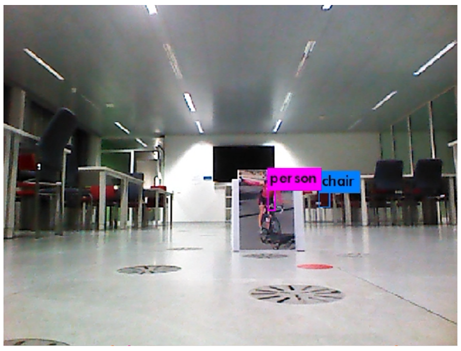


(e) After removal of the second obstacle, the object detecter recognizes no objects (in the distance is the third obstacle [Hun12], cropped)

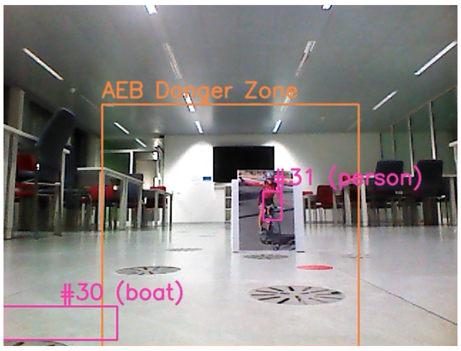


(f) The AEB ADAS recognizes the persons a few moments after they are removed (in the distance is the third obstacle [Hun12], cropped)

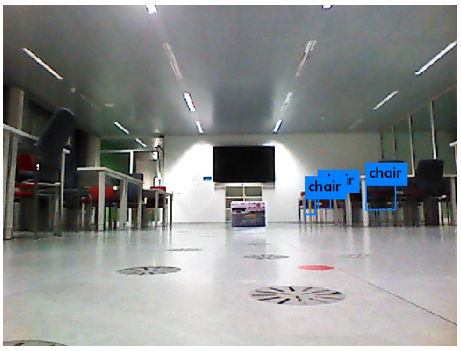
Figure A.3: Screenshots of the object detector (left) and AEB ADAS (right) camera images during experiment - related to the second obstacle (obstacle image [mal07]) - Part 2



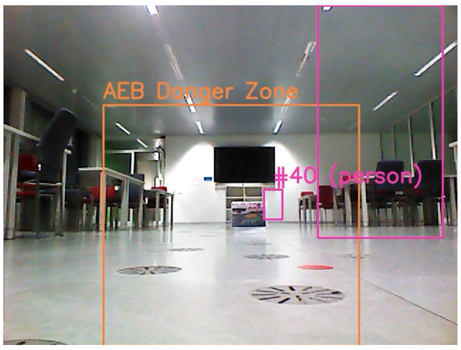
(a) The object detector recognizes a person and a chair



(b) Contrary to Figure A.4a, the AEB ADAS also recognizes a boat (ID #30) besides the person (ID #31)

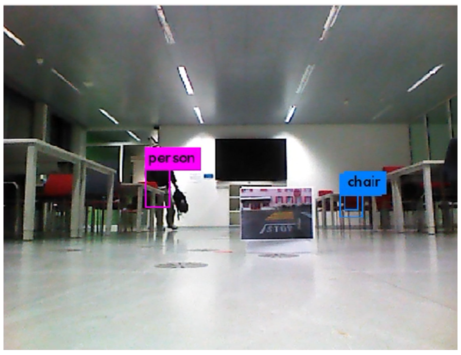


(c) The object detector recognizes multiple chairs (in the distance is the fifth obstacle [Kec13])

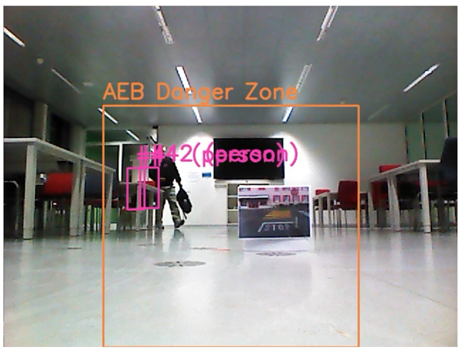


(d) Contrary to Figure A.4c, the AEB ADAS still recognizes the person, but with a different ID (ID #40) and the bounding box of the person that removed the obstacle (in the distance is the fifth obstacle [Kec13])

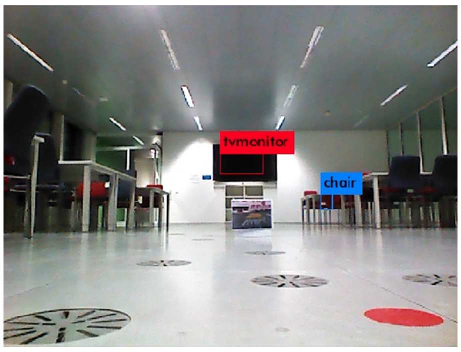
Figure A.4: Screenshots of object detector (left) and AEB ADAS (right) camera images during experiment - related to the third obstacle (obstacle image [Hun12], cropped)



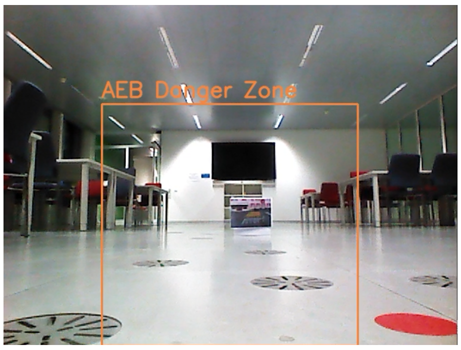
(a) The object detector recognizes person an chairs



(b) The AEB ADAS recognizes the person multipe times as there are multiple bounding boxes and the person is moving (e.g., ID #42)

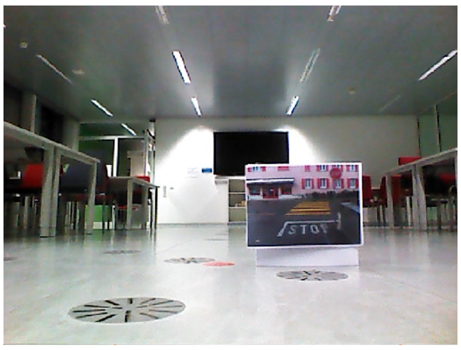


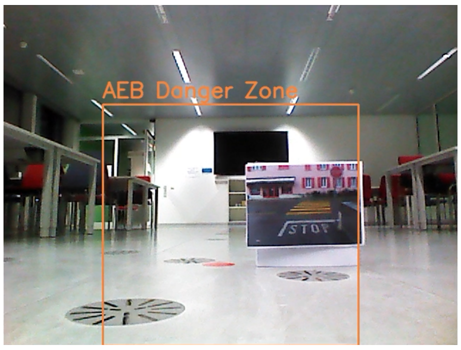
(c) The object detector recognizes the tv monitor and a chair



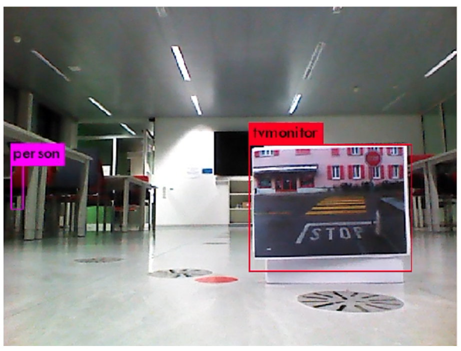
(d) TheAEB ADAS does not recognize the objects from Figure A.5c, as they are not whitelisted

Figure A.5: Screenshots of object detector (left) and AEB ADAS (right) camera images during experiment - related to the fourth obstacle (obstacle - random moving person, unrecognizable and anonymous, there is also the fifth obstacle [Kec13])

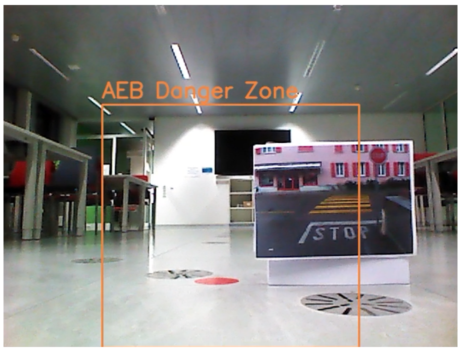

(a) The object detector does not recognize the obstacle



(b) As in Figure A.6a, the AEB ADAS does not recognize the obstacle



(c) The object detector recognizes a person and then the obstacle as tv monitor



(d) The obstacle is not recognized as tv monitors are not whitelisted and the stop sign is outside the danger zone (rectangle)

Figure A.6: Screenshots of object detector (left) and AEB ADAS (right) camera images during experiment - related to the fifth obstacle (obstacle image [Kec13])

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

place, date, signature