METHODS TO ENSURE SOFTWARE SAFETY FOR SAFETY-CRITICAL AUTONOMOUS SYSTEMS: A SYSTEMATIC LITERATURE REVIEW

Author

Bikash Shrestha

Supervisor

Prof. Marina Walden

A thesis submitted for the degree of

Master's in Computer Engineering

December 2021

Abstract

Many companies are working on autonomous systems, such as autonomous vehicles and autonomous vessels, as the solution to reduce human errors that can result in huge losses. Over the past decade, various machine learning techniques were implemented for controlling autonomous systems. Due to the safety-critical nature of autonomous systems, it is important to ensure software safety along with the safety of the physical system of an autonomous system. In this paper, a systematic literature review method is used to analyse the various aspects of software safety in an autonomous system. Specifically, this paper assesses the existing techniques, such as software development guidelines, system design and architecture, machine learning techniques, and formal verification methods, to ensure software safety in autonomous systems and summarizes the challenges referred to in the literature concerning software safety of autonomous systems. The results presented in this paper are relevant to the researchers seeking better methods to ensure software safety in autonomous systems.

Keywords: Systematic literature review; Autonomous system; Reinforcement learning; Q-learning; Software safety; Autonomous vehicles; Machine learning; Software architecture; Software design; System architecture.

Acknowledgement

I would like to express my sincere gratitude to my supervisor, Professor Marina Walden, for her incredible support and assistance throughout the writing of this thesis. Her expertise and insightful feedback were indispensable to formulate the research methodology and brought my work to a higher level.

In addition, I would like to thank my parents, brother, sister, and friends for their continuous support and happy distraction to keep my mind soulful outside my research.

Table of Contents

1. Introduction

An autonomous system is referred to as an automated system that will operate itself in response to external conditions and without any human intervention. The requirements for a system to be autonomous are sensing the environment and keeping the record of the state, perceiving, and understanding the data sources, determining the response to the environment, and acting only when it is safe to do so. The application of autonomous systems was limited to industrial settings for many years but in recent years, autonomous systems have gradually become involved in people's everyday life as a means of transportation or entertainment. Approximately 94% of road accidents are caused by human error [17]. While designing a vehicle in an automotive domain, the development of dependable motion control systems has a key role. Before the last two decades, motion control systems such as steering and braking systems were dependent on mechanical components and operated independently of one another. However, in recent decades, motion control systems are developed using electronics and software such as power steering, crash mitigation braking, and adaptive cruise control. Under some circumstances, such advanced features operate independently of drivers, and multiple motion control systems may activate simultaneously based on the actions taken by other features.

Many kinds of autonomous controls were used in various modes of transportation for many decades. The first autopilot in an aircraft was developed nearly a century ago. Similarly, autopilots for ships were developed after those for aircrafts. Since then, the level of automation in various modes of transportation has been developed dramatically and changed the man-machine relationship. Driverless trains were also developed nearly half a century ago. In 1905, a global association of engineers and related technical experts in the aerospace, automotive, and commercial-vehicle industries was inaugurated which is known as the Society of Automotive Engineers (SAE) International. Depending on human assistance, it has defined a taxonomy with detailed definitions for six levels of automation for autonomous vehicles (AV) as shown in Table 1.

Level	Definition
0	No Driving Automation
1	Driver Assistance
2	Partial Driving Automation
3	Conditional Driving Automation
4	High Driving Automation
5	Fully Driving Automation

Table 1. SAE levels of automation [53]

At level 0, a driver is fully responsible for controlling the vehicle, whereas vehicle systems are limited to providing warnings and momentary assistance. Similarly, at level 1, the driver cooperates with the vehicle system to increase driving performance. At level 2, the vehicle system is more responsible for controlling the vehicle, although the driver should be in an alert position to intervene whenever needed. The attention of the driver is less strictly required in level 3 than in level 2. The vehicles included in level 3 have limited capability to control and perceive through sensors. The vehicles that are classified into level 4 are considered highly automated vehicles. These vehicles are operated in predefined modes and can perform all tasks including safety-critical ones, without any need for human assistance, if only the required conditions are met. the level 5 vehicles are fully driven by the vehicle system in all conditions and without any human intervention [16].

Along with the gradual development of autonomous systems, advanced technology such as enhanced sensor systems, high computational power, software design paradigms, artificial intelligence (AI), and machine learning (ML) were developed that helped to make the autonomous systems more significant [46]. All of these are related to the software directly or indirectly. Autonomous systems are also known as cyber-physical systems because they integrate sensing, computation, control, and networking into physical objects, connecting them to the internet and to each other autonomous systems are gradually integrating with people's lives, it has become critical to research and develop effective quality assurance methods to prevent loss as early as possible. Hence, it is very important to ensure the software safety of autonomous systems.

In general, data-driven software development processes, complying with safety standards (ISO 26262 and IEC 61508), and applying automation to design, verification, and validation are considered as building blocks to develop safe

software for autonomous systems. A secure development process of software is essential to keep the external interference out. This can be achieved by following good software developing practices, such as doing continuous testing to remove the security vulnerabilities, analysing the risks and hazards, and keeping the control over build/release environment to prevent an external attack. Also, using various tools to design and validate the software may help to increase software safety. In addition, it is important to follow safety standards like ISO 26262 and IEC 61508 in autonomous systems as they were developed by experts and ensure the overall safety of autonomous systems.

In this paper, unmanned aerial vehicles (UAV and drones), autonomous vehicles, and vessels are overall termed autonomous systems. Both partial and fully autonomous systems are included in this work. The methods to ensure software safety in autonomous systems and their importance are the main motivation of the study. The results are presented in this paper as a Systematic Literature Review (SLR) of the existing methodologies and tools used to support the study.

The rest of the paper is organized as follows: Chapter 2 describes the research methodology which includes research questions and other contexts of study along with a general classification of the papers resulting from the literature search. Chapter 3 presents the results of the literature review on existing methodologies. Chapter 4 analyses and discusses more specific SLR results. Chapter 5 concludes the theme of the paper with general remarks.

2. Research Methodology

This survey adopts the guidelines for SLR in software engineering from *Kitchenham and Charters, 2007* and *Kitchenham et al., 2010*. Following the guidelines, this paper is presented as a foundation for the researchers and practitioners seeking hints related to the study.

2.1. Research Questions

The aim of this SLR is to analyse the methods to ensure software safety in autonomous systems. To meet the proposed objectives of the study, the research questions are defined as follows.

RQ1: Are there any guidelines to ensure software safety in autonomous vehicles? What kind of guidelines are available?

The main aim of RQ1 is to find out if there are any guidelines and standards available to create safe software for the autonomous system. If so, the main concern will be whether they are developed by the experts or a relevant organization and how they address the solution to the topic of study.

RQ2: What is the role of system design and architecture in autonomous systems?

This question is created to answer the queries related to the software development process and software architecture in an autonomous system. An inappropriate design and architecture may lead the system to fail. Also, there might be various software components to be considered during the development for its robustness. Hence, RQ2 is important.

RQ3: Evaluate available ML approaches for software safety of autonomous systems.

Various modelling techniques, such as machine learning and artificial intelligence, are implemented in various forms on autonomous systems. The goal here is to evaluate the most effective techniques used for the software safety of autonomous systems.

RQ4: Are there any frameworks or tools available to ensure software safety of autonomous systems? If so, what kind of frameworks are available?

The core objective of this research question is to determine the framework tools available to produce a quality software and report them.

RQ5: How does verification and validation of software help in software safety?

Verification and validation (VV) of software helps to develop quality software by mitigating risks and chances of failure. VV also evaluates the weakness of software and prevents the build/release from unauthorized access. Thus, RQ5 is questioned to evaluate the existing VV methods.

RQ6: Are there any other approaches that improve a safety critical system?

The goal of this question is to research and evaluate the other approaches that help to improve software safety. An autonomous system has been a hot topic for the last few decades. Several methods and approaches have gradually been developed to enhance the autonomous system. Therefore, RQ6 is meant to determine the other approaches relevant to the study.

2.2. Search Process

A comprehensive search for primary studies was done online through some digital libraries: IEEE Xplore, Mendeley, SpringerLink, ACM Digital Library, Elsevier ScienceDirect, and Google Scholar. Only the studies written in English were considered. Google search was also used to gather some vital information. The search keywords are very important when doing an online search in digital libraries to retrieve high-quality results and coverage. For the search, the following search keywords were used: ‘software safety’, ‘standards’, ‘software architecture’, ‘Q learning’, ‘autonomous system’, and ‘reinforcement learning’. To retrieve the quality result related to the study and using the search keywords, the search strings were formulated as follows:

A. ‘Software safety’ AND ‘autonomous system'
B. ‘Software safety’ AND ‘standards’
C. ‘Software safety’ AND ‘software architecture’
D. ‘Q learning’ AND ‘autonomous system’
E. ‘Reinforcement learning’ AND ‘autonomous system’

Using the strings above, 96 papers were collected from digital databases (Step 1 in figure 1). After screening through the papers briefly based on title, abstract, and introduction, about 33 papers are found to be irrelevant to the study of interest.

2.3. Inclusion and Exclusion

Furthermore, inclusion and exclusion criteria were applied to refine the studies. As shown in figure 1 (Step 3), the criteria were applied to the chosen 63 papers for refinement, and the number of studies to be considered dropped to 51. The papers that describe the existing software safety approach in autonomous systems, methods to ensure software safety, and compliance with software

safety standards are defined as inclusion criteria. The study that describes the safety of autonomous systems in terms of hardware or physical body only, written in different languages other than English, incomplete or duplicated studies, tutorials, and editorials are defined as exclusion criteria.

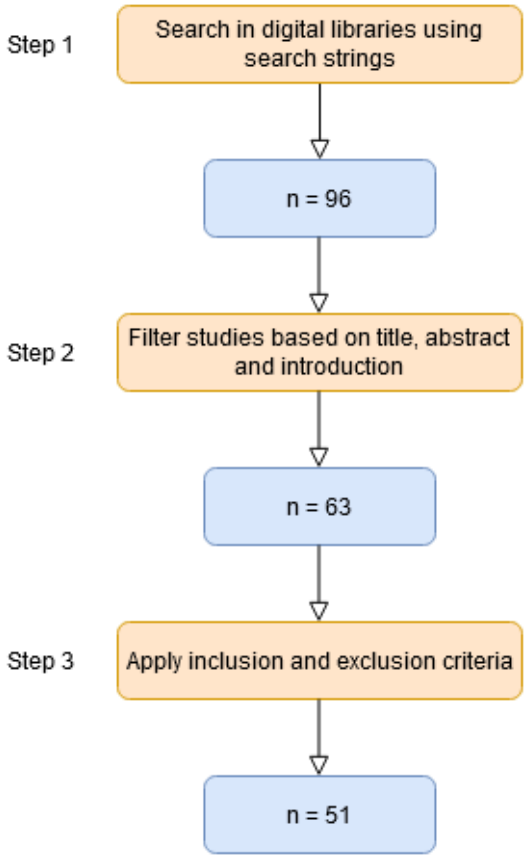

Figure 1. Overview of the selection of primary studies

2.4. Quality Assessment

The resourcefulness or confidence of the research papers is evaluated using the quality assessment (QA) questions. The questionaries are mainly aimed to evaluate the relevance and significance of the studies. The QA questions are listed below.

QA1. Is the research objective being properly elucidated?

QA2. Is the proposed method clearly defined?

QA3. Does the paper define findings and results properly?

QA4. Are the limitations of this study explicitly discussed?

Based on the QA questions, a quality score is allotted for each study. The grading of a study is done as follows.

- 1, if the study answered the question satisfactorily
- 0.5, if the study answered the question partly
- 0, if the study was not able to answer the question

The result of the QA is shown in Table 2.

Study	***Author***	***QA1***	***QA2***	***QA3***	***QA4***	***Total Score***
S1	Zuo et al. (2020)	1	1	1	0	3
S2	Okuyama et al. (2018)	1	1	1	0	3
S3	Lee et al. (2017)	1	1	0.5	0	2.5
S4	Beine (2010)	1	1	0	0	2
S5	Strauss and Sahin (2008)	1	1	1	0.5	3.5
S6	Wang et al. (2017)	1	0.5	0.5	0	2
S7	Mallozzi (2017)	1	1	0	0	2
S8	Kiran et al. (2021)	1	1	0.5	1	3.5
S9	Wang and Chan (2017)	1	1	0	0	2
S10	Vierhauser et al. (2018)	1	0.5	0	0.5	2
S11	Deshmukh et al. (2019)	1	1	1	0	3
S12	Pendleton et al. (2017)	1	1	1	0	3
S13	Sward (2005)	1	1	0	0	2
S14	Afzal (2018)	1	1	0	0	2
S15	Fulton and Platzer (2018)	1	1	1	0	3
S16	Rajabli et al. (2020)	1	1	1	1	4
S17	Culley et al. (2020)	1	0.5	0.5	0.5	2.5
S18	Furst (2019)	0.5	0.5	0.5	0	1.5
S19	McDermid et al. (2019)	1	0.5	0.5	1	3
S20	Hardin (2021)	1	1	0.5	0	2.5
S21	Alberri et al. (2018)	1	1	1	0	3
S22	Aniculaesei et al. (2018)	1	1	0	1	3
S23	Beri and Mishra (2019)	1	1	1	0	3
S24	Lyins and Zahra (2020)	1	1	0.5	0	2.5
S25	Wesel and Goodloe (2017)	1	1	0.5	1	3.5
S26	Zhang et al. (2020)	1	0.5	0.5	0	2
S27	Gambi et al. (2019)	1	1	0.5	0	2.5
S28	Fehlmann and Kranich (2017)	1	1	0.5	0	2.5

S29	Koopman and Wagner (2016)	1	0.5	0	1	2.5
S30	Tahir and Alexander (2020)	1	1	0.5	0	2.5
S31	Meltz and Guterman (2019)	1	1	0	0	2
S32	Jha et al. (2019)	1	1	1	0	3
S33	Natella et al. (2012)	1	1	0.5	0.5	3
S34	Kuutti et al. (2019)	1	1	0.5	0	2.5
S35	Debouk et al. (2011)	1	1	0.5	0	2.5
S36	Zhang and Li (2020)	1	1	0.5	0	2.5
S37	Kim et al. (2017)	1	1	0	0	2
S38	Bhat et al. (2018)	1	1	0	0	2
S39	Gustavsson (2016)	1	1	0	0	2
S40	MISRA (2001a)	1	1	0	0	2
S41	MISRA (2001b)	1	1	0	0	2
S42	MISRA (2001c)	1	1	0	0	2
S43	MISRA (2001d)	1	1	1	0	3
S44	Vuori (2011)	1	1	1	0.5	3.5
S45	Salay et al. (2017)	1	1	0.5	0	2.5
S46	Liu et al. (2020)	1	0.5	0.5	1	3
S47	Ebert and Weyrich (2019)	0.5	1	0	0.5	2
S48	Lera et al. (2016)	1	1	0.5	0	2.5
S49	Liu et al. (2018)	1	1	0.5	0	2.5
S50	Madan et al. (2016)	1	0.5	0.5	0	2
S51	Whalen and Heimdahl (1999)	1	1	0.5	0	2.5

Table 2. Quality Assessment of the studies

Table 2 shows that most of the studies have scored 2 or more in terms of quality analysis. Only one has scored less than 2 ([S18]) while study [S16] has scored 4 out of 4 in context to its quality.

2.5. Data Extraction

This stage aims to extract the appropriate information from the selected papers accurately and without bias. The extracted information should follow the study quality criteria and should also be able to answer the research questions defined in the protocol of this SLR. From each of the studies, bibliographic information on studies was extracted along with various information such as methods to develop safety-critical software, underlying software safety standards for autonomous systems, and tools available for ensuring software safety.

Publication Source	***Type***	***Number***
The Motor Industry Software Reliability Association (MISRA)	Book	4
National Aeronautics and Space Administration (NASA)	Technical Memorandum	1
IEEE	Journal	9
Elsevier	Journal	1
Agile Development of Safety-Critical Software	Journal	1
Society of Automotive Engineers (SAE)	Journal	2
Journal of Defense Modeling and Simulation: Applications, Methodology, Technology	Journal	1
CSEE Journal of Power and Energy Systems	Journal	1
Multidisciplinary Digital Publishing Institute (MDPI)	Journal	1
SAE International Journal of Transportation Safety	Journal	1
International Conference on Intelligent Autonomous Systems (ICoIAS)	Conference	1
Internation Conference on Ubiquitous Robots and Ambient Intelligence (URAI)	Conference	1
International Conference on System of Systems Engineering	Conference	1
Global Conference on Signal and Information Processing (GlobalSIP)	Conference	1
International Conference on Software Engineering Companion (ICSE-C)	Conference	4
International Conference on Intelligent Transporation Systems	Conference	1
International Conference on Computer-Aided Design (ICCAD)	Conference	1
IEEE Networking, Sensing and Control	Conference	1
European Software Engineering Conference and Symposium on the Foundation of Software (ESEC7FSE)	Conference	1
AAAI Conference on Artificial Intelligence	Conference	1
International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP)	Conference	1
International Systems Conference (SysCon)	Conference	1
International Conference on Vehicular Electronics and Safety (ICVES)	Conference	1
International Conference on Trends in Electronics and Informatics (ICOEI)	Conference	1
Security and Privacy Workshops (SPW)	Conference	1
International Workshop on Software Measurement (IWSM)	Conference	1
International Conference on Software Testing, Verification and Validation	Conference	1
International conference on Artificial Intelligence Testing (AITest)	Conference	1
International Conference on Intelligent Data Engineering and Automated Learning (IDEAL)	Conference	1

International System Safety Society Conference	Conference	1
International Conference on Dependable Systems and Networks	Conference	1
XVII Workshop en Agentes Fisicos	Conference	1
Workshop on Cyber-Physical Systems Security and Privacy (CPS-SPC)	Conference	1
International Conference on Automated Software Engineering	Conference	1
International Workshop on Software Engineering for AI in Autonomous Systems	Workshop	1
CEUR Workshop	Workshop	1

Table 3. Distribution of studies according to the publication venue

Table 3 shows the number of publications found for each venue: book, technical memorandum, journals, conferences, and workshops. The study collection consists of one book, one technical memorandum, 17 journals, 27 conference papers and two workshop articles with the earliest article from year 1999 and the latest from 2021.

2.6. Data Synthesis

This section includes the further evaluation of the selected studies. The selected studies are categorized according to the topics addressed by the studies. Table 4 shows the categorization of studies based on their essence. It is worth mentioning that there are some overlaps in the topic coverage of those classification. However, the main purpose of classification is to provide general overview on the papers and the categories will be described later in Section 3.

Topics	***Studies***
Guidelines	[S4], [S40], [S41], [S42], [S43], [S45]
System architecture and design	[S10], [S12], [S17], [S18], [S21], [S22], [S23], [S35], [S38], [S44], [S48], [S49], [S50]
Machine Learning	[S1], [S2], [S3], [S5], [S6], [S7], [S8], [S9], [S11], [S24], [S26], [S34]
Safety assurance frameworks	[S19], [S51]
Formal Methods	[S13], [S14], [S15], [S16], [S20], [S27], [S28], [S30], [S31], [S32], [S33], [S36], [S37], [S39], [S47]
Challenges	[S25], [S29], [S46]

Table 4. Categorization of studies

In table 4, the studies related to various machine learning approaches such as reinforcement learning, deep neural network, Q-learning are classified into the category “Machine Learning” while “Guidelines” includes the studies related to safety standards of vehicles and general ideas to keep the software safe from external threats. Similarly, the studies that describe the formal methods to create safe software and methods to verify and validate the quality of software are categorized under “Formal Methods”.

3. Methods to ensure software safety

In this chapter, various ways to ensure the software safety of an autonomous system are overviewed. The guidelines and standards developed by various organizations are taken as a reference for developing safe autonomous systems. Similarly, autonomous system design, machine learning techniques, safety assurance frameworks, and formal methods illustrated by various authors are assessed as a systematic literature review with the purpose of gathering relevant information that might be vital for the subject-related researchers.

Several papers describe various approaches and techniques to make an autonomous system safe. The essence of the papers is concluded under several headings in the following sections.

3.1 Safety standards and guidelines to ensure safety

Many guidelines and safety standards have been developed for the safety of autonomous vehicles concerning the development of safety-critical software and functional safety of the autonomous systems. The functionality safety standard ISO 26262 and automotive industrial standard IEC 61508 are discussed in this section along with model-based reference workflow as guidelines that help to follow safety standard ISO 26262. In addition, a consortium was established in the early 1990s, in response to the UK Safety-Critical Systems Research Programme to focus on coding security standards which are known as The Motor Industry Software Reliability Association (MISRA). At present, a set of rules developed by the association acts as guidelines in the automotive industry. Some of the relevant reports produced by MISRA are covered in this chapter.

3.1.1 ISO and IEC standards

The study done by Beine [S4] illustrates the model-based software development to shorten the software development cycles while ensuring efficient implementation and conforming to relevant safety standards such as IEC 61508 and ISO 26262. IEC 61508 is a generic industry standard that helped to derive other industry-specific standards for process industry (IEC 61511), nuclear power plants (IEC 61513) and machinery (IEC 61513). Similarly, ISO 26262 is a widely followed standard for the development of safety-critical E/E (Electrical/Electronic) systems and requirements applicable to the automotive industry to safeguard the systems from failures [4]. ISO 26262 mainly sets a rule for functional safety of the road vehicles using the Hazard Analysis and Risk Assessment (HARA) method to identify and remove the hazards in the system. The potential hazards can be known or unknown. The known behaviour/ hazards can be removed by doing requirements-based testing, whereas unknown potential hazards can be addressed with scenario-based testing. Scenario-based testing refers to converting driving scenarios into test cases to reduce the risk while validating the behaviour of an autonomous system. ISO 26262 are divided into 10 parts. The 6th part of the standard (ISO 26262-6) defines the V-model and is focused on the safety-critical software development cycle which recommends the use of designs and coding guidelines for modelling as well as programming languages [45].

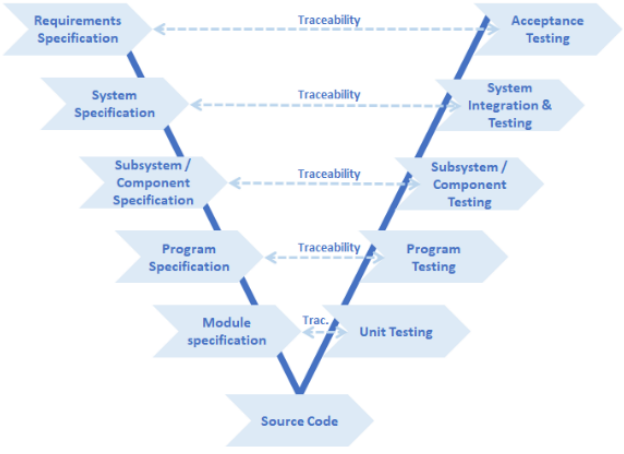

Figure 2: V-model in ISO 26262-6 [16]

In the V-model, as shown in Figure 2, the left side represents the waterfall model while the right side represents the incremental verification and validation process applied to the waterfall model. ISO 26262 also defines the risk classification scheme for a subsystem in an automotive system which is known as Automotive Safety Integrity Level (ASIL). The scheme represents the degree of accuracy required, such as testing techniques and documentation types to reduce the risk of the subsystem. The highest risk is represented by ASIL D, whereas ASIL A represents the lowest risk [45].

In contrast to non-safety critical software development and code-based development, all the additional requirements including specific requirements in the relevant safety standards must be met when developing safety-critical software. In such conditions, a reference workflow can provide the guidance to follow the safety requirements of ISO 26262 or IEC 61508 standard in developing software up to and including ASIL D or SIL 3 respectively. Model-based development and automatic code generation have played a significant role in the software development processes in recent years. These methods have been deployed successfully in various automotive and aerospace industries. These methods have also improved the software development life cycle by reducing the development times, improving the quality of the product due to more precision, and early verification and validation by means of simulation. The best practices and experience from the real-world projects are considered in terms of software safety and a reference workflow for the development of safety-critical software has been prepared for various established toolchains such as MATLAB, Simulink, Stateflow, and TargetLink. Such model-based development includes automatic code generation and model-based testing methods [4].

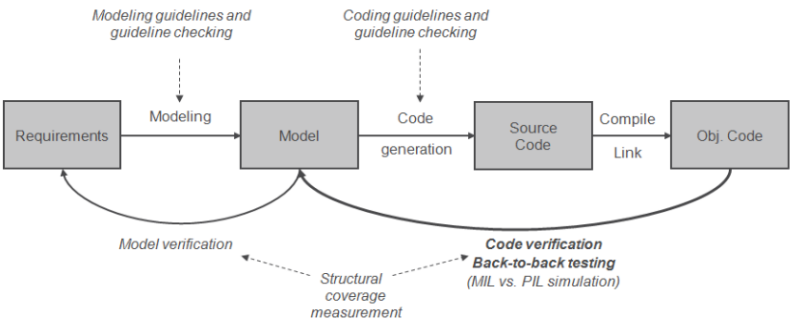

Figure 3: An overview of model-based reference workflow [4]

In Figure 3, an overview of model-based reference workflow is shown which consists of general elements of the processes. The figure shows that the requirements are designed and implemented in an executable model, which then is translated into code using the code generation method. The stage from requirements to model requires model verification which is done by model simulation and requirement-based testing. Similarly, the code generated from the model is verified by back-to-back testing, directly comparing the intended functional behaviour of the model and code. The key element of this workflow is the verification of automatically generated code against the model [4].

3.1.2 MISRA guidelines

The MISRA consortium was formed with a mission to aid the automotive industry in the creation and application of a safe and reliable software in a vehicle. The consortium provides detailed guidelines for software lifecycle to ensure software safety which includes project planning, integrity, requirements specification, design, programming, testing, and product support. Furthermore, the consortium focuses on several factors in software development such as human factors, quality assurance, documentation, and team management to enhance the software quality [40].

In the study [S41], the MISRA consortium has reported the role of software in the design of control systems in terms of theory, design, and practical considerations. The study defines the control theory and its role in the development of safe and reliable software. In fact, control theory is a mathematical description of the behaviour of dynamic systems that varies with time and is typically applied to feedback systems, where a proportion of output is fed as input to the system. The report has also emphasized automatic code generation after a system is designed and modelled. However, the code generated can be optimized or adjusted further based on the requirements [41].

In the study [S42], MISRA has documented the software metrics and attributes which help to measure the quality of the software. Software process metrics help to improve the development process by providing information about its performance. Such information or data assists in highlighting the areas of inefficiency or error-prone areas of the process and the ultimate quality of the

software being produced. The basic metrics suggested by the study are the estimated duration of each task, actual effort spent on tasks and the number of defects detected. The study also suggests that software reliability can be achieved by improving the design strategies, such as defensive programming, diversity, or redundancy of the software. Defensive programming allows the software to take specific actions in case some failure occurs. Likewise, the diversity of the software removes some risks that are common causes for failures. Similarly, software redundancy allows another version of the software to continue the operation if a version of the software becomes corrupt. In addition, it is also reported that the testability of the system is essential. A software system and its software component should be designed in such a way that the functions can be fully tested to validate their requirements [42].

Moreover, the study [S42] has also defined a maturity index as a metric to specify the degree of functional change incorporated into a version of a software system. In fact, this metric is useful to quantify the readiness of software and is calculated as below.

$$Maturity\ index = (Mt - (Fa + Fc + Fd))/Mt$$

where, Mt = number of software functions in current delivery

Fa = number of additional modules

Fc = number of changed modules

Fd = number of deleted modules

The formula above indicates that the maturity applies to all existing software components used in the software system and to any software tools used in the specification, development, integration, test, and maintenance of the system.

The study [S43] is also a report of the MISRA consortium regarding the verification and validation of the software subsystems. The report suggests that verification and validation activities should be carried out as soon as a component is ready to prevent unnecessary costs. However, the depth and preciseness of the verification and validation activities will depend on the ASIL of the system. The report provides the guidelines on software validation and

verification and the personnel responsible for the tasks. Besides these, the report also focuses on the importance of software requirements analysis prior to the software architecture design. The objective of software requirement analysis is to ensure that the system acts as intended and is verifiable after design. It is also essential for a designed system to be compatible with the targeted hardware and conformance to the standards [43].

3.2 System architecture and design

A huge number of autonomous systems have been made and demonstrated since the late 20th century. The autonomous systems are developed as unmanned aerial vehicles, unmanned ground vehicles, or unmanned water vehicles. Such systems are developed for delivering goods, performing surveillance, conducting search and rescue, and supporting hobbyist activities. Enhanced sensor systems, high computational power devices, and machine learning techniques have been a part of the software design paradigm of autonomous systems. A failure in such systems or a cyber threat could lead to fatal accidents or significant damages to the environment. Hence, system and software safety in autonomous systems along with the cost-effective solution has been a hot topic in the recent years. Many organizations and research teams have invested a tremendous amount of time and budget in the design and system architecture of autonomous systems. This chapter covers some of the research papers relevant to the system design and architecture of autonomous systems.

3.2.1 System design strategies

In the study [S10], Vierhauser et al. (2018) have demonstrated an approach for interlocking safety cases of unmanned aerial vehicles in the airspace of urban environments for safe use of unmanned aerial systems. The purpose of the research was to define the safety mechanism of autonomous systems at both infrastructure and the application level. Thus, the authors have divided the safety assurance cases (SAC) into two parts that can be effectively interlocked: Infrastructure-level SAC (iSAC) and unmanned aerial system-level SAC (uSAC) for unmanned aerial systems associated with it [10]. Each of the SAC is defined with interlock points where potentially untrusted uSAC can interface with trusted and assured iSAC to provide static or runtime checkable constraints. When each unmanned aerial system approaches the controlled airspace, runtime monitoring

of unmanned aerial vehicles is created dynamically. The iSAC is created with specific assumptions on SAC that help in interlocking with unmanned aerial systems and for this, interlock points are established between unmanned aerial systems' specific assumptions in iSAC and claims that uSAC meets the assumptions. The assumptions are categorized into three groups: @entry, @monitor, and @usac. The first assumption denotes the monitorable property of unmanned aerial systems once it enters space. The @monitor denotes the property of unmanned aerial systems that are monitored continuously whilst in controlled space. Similarly, @usac denotes the assumptions that are not covered fully under monitorable properties. Such an approach has been proposed by the authors for assuring safe use of unmanned aerial systems in urban environments, but dynamic analysis of @usac arguments is still left open for discussion by authors [10].

An autonomous vehicle is developed using an array of enhanced sensors and a localization algorithm is applied to create a link between the sensors [18]. An object recognition algorithm is used in an autonomous vehicle to identify the objects within its boundaries. Similarly, a path-planning algorithm is applied to follow a track and a mapping algorithm is applied to record the data [17]. Hence, the core competencies of an autonomous vehicle software system can be classified into three groups: perception, planning, and control.

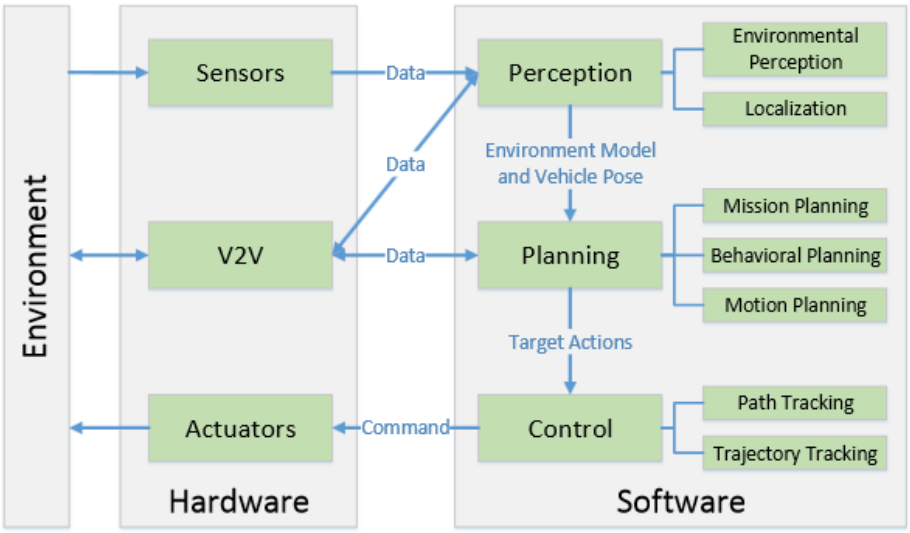

Figure 4: A typical autonomous vehicle system overview, highlighting core competencies [12]

Perception is defined as an ability of an autonomous system to gather relevant information from the environment. It is further divided into environmental perception and localization, as shown in Figure 4. Environmental perception is referred to as perceiving the contextual environment while localization is referred to as the capability to determine its position with respect to the environment. Planning is a process of making decisions to achieve goals. Similarly, control is defined as the ability of autonomous systems to execute the plans as directed by higher level processes [12].

In the study [S35], Debouk et al. (2011) have proposed software design strategies considering factors relevant to the software safety of autonomous systems. Such safety factors could be potential unpredicted behaviour, availability of autonomous systems, and driver attentiveness. The authors have designed a fail-safe / fail-silent control unit to monitor and analyse the safety factors and hence, ensure fault handling. The fail-safe / fail-silent control unit acts as a building block to construct a fail-operational unit implementing the needed autonomous driving functionality. A fail-safe / fail-silent control unit itself is a hardware component. However, it contributes to the software approach for detecting faults in the system, as shown in Figure 5.

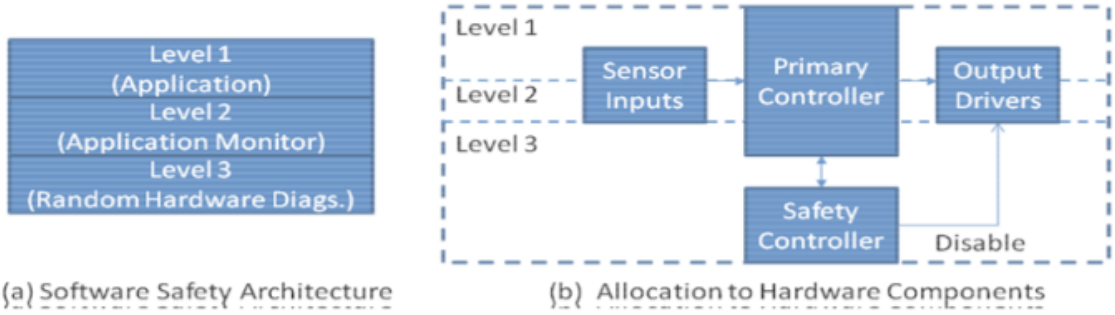

Figure 5: Software Strategy for Fail-safe/ Fail-silent Control Unit [35]

As shown in Figure 5, level 1 and level 2 run on the primary control unit. Level 1 defines the control application that provides the primary functionality of the system. Level 2 consists of diagnostic software that monitors the unexpected behaviour of the level 1 application. Level 3 software is dedicated to detecting

random hardware failures of both control units by performing diagnostic cross-checks between the two control units [35].

In a similar way, the study [S21] introduces a generic hierarchal Robotic Operating System (ROS)-based architecture for autonomous systems that is capable of exchanging data among inter-connected multi-robotic heterogeneous systems. The introduced architecture consists of three layers, as shown in Figure 6.

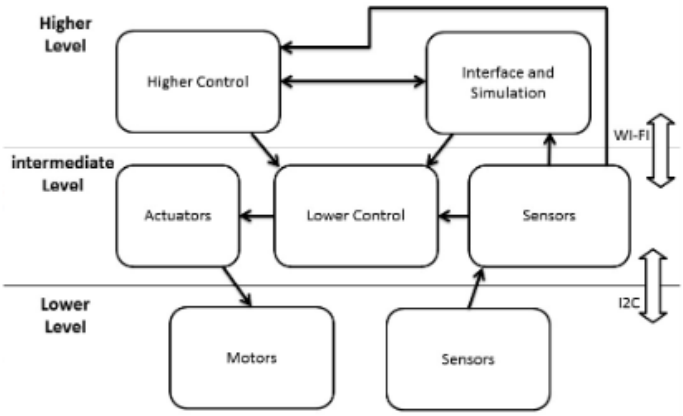

Figure 6: Block diagram describing software architecture and packages [21]

In Figure 6, the lower-level controller gathers data and forwards it to a higher-level controller for complex computations. The intermediate-level acts as a medium to transfer data from lower-level to higher-level. The authors assure that the proposed architecture was validated in the form of experiments on an autonomous mobile robot, autonomous vehicle, and autonomous quadcopter, and the results of the experiment have proven to be acceptable for more complex autonomous system applications too [21].

3.2.2 Software development process

In addition to all these, a software development cycle has a vital role in developing safety-critical software. It is important for a product to adapt to the changing requirements during the process and yet deliver the best value to customers and developers earlier. Hence, the agile software development process was developed and Vuori [S44] has analysed the agile principles in his

paper “Agile Development of Safety-Critical Software” and provides guidance on how an organization can switch to the agile environment from traditional software development lifecycle models. Prior to defining an agile process, he has also analysed the requirements for a safety-critical software development process as follows [44].

- Knowledge of risk: During the process, risks and hazards should be analysed and shared with the whole team for better understanding. The process also requires a thorough analysis of the software’s actual usage.
- Quality: Expertise and professionalism is highly required to develop high-quality software. All abstraction levels of the safety-critical software should maintain a high level of quality assurance.
- Control: A safety-critical software development process needs good teamwork and a better understanding of tasks and objectives. The process might be complex and team collaboration can make it simple and clear.
- Analysis: In a safety-critical software development process, safety assessment should be carried out and documented accordingly. The changes in a system should also be documented so that they can be retracted if needed.
- Time and resources: A safety-critical software development process will succeed only if there is enough time and resources to carry out the tasks properly. All the development practices should follow the safety standards and should be verified and validated as soon as implemented or changes occurs.
- Auditability: The process must be auditable during and after execution.

An agile software development process consists of principles, project models, software development lifecycle and software engineering techniques, and practices. The principle is described as values and policies that stakeholders behold. The project model is described as a project planning and execution where software is designed and developed using a specific software development lifecycle and using various software engineering techniques and practices. In the agile development lifecycle, each phase is done iteratively which

helps to deliver a better product. The workflow of an agile software development process generally consists of five steps: story, coding, running program, evaluation, and design iteration [44].

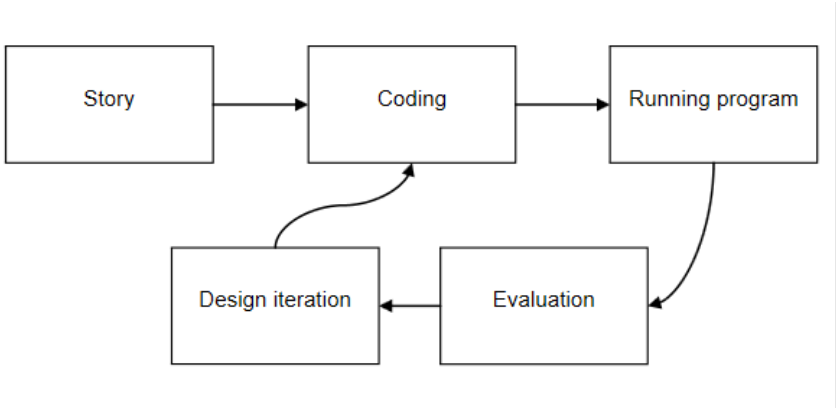

Figure 7: A generic agile software development process [44]

In Figure 7, the story represents the product requirements needed to design a product. Such requirements can be functional and safety requirements as shown in Figure 8.

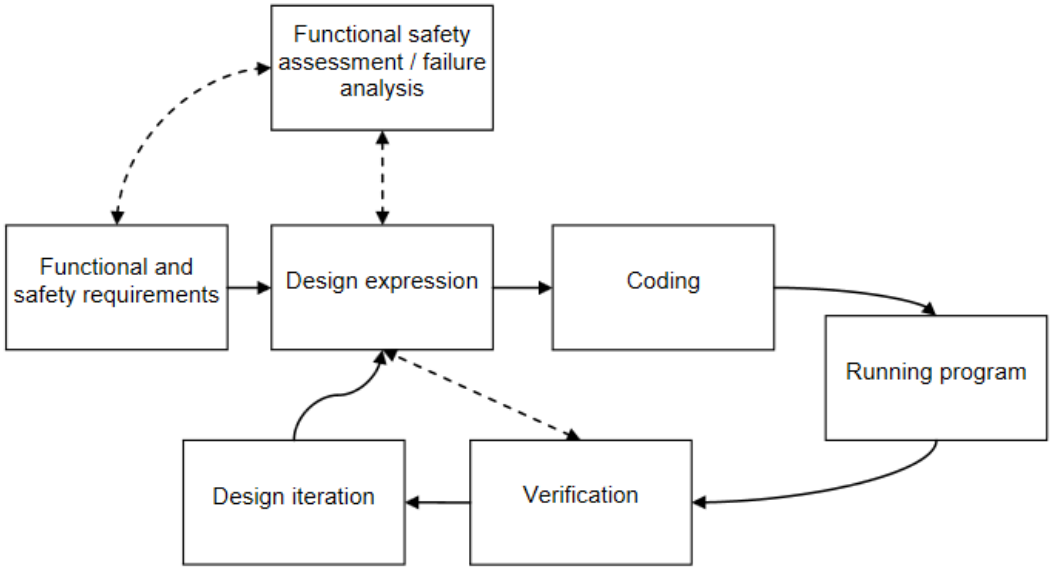

Figure 8: A safety-critical software development lifecycle using agile method [44]

A safety-critical software should be able to handle the unpredicted behaviour of the system during its operation. Thus, it is important to do failure analysis and other functional safety assessments during the safety-critical software development. Once the requirements and assessment are documented, the programming of the product can be started. Each implementation of the system is then tested and evaluated iteratively. If needed, changes are sent as feedback to the design. This process is done repeatedly until a complete product is ready. This helps to save time and cost, and yet produce robust software by removing all possible errors [44].

3.2.3 Cyber security on connected autonomous systems

During the development of autonomous systems, developers might not be able to consider all the situations that the system will confront whilst in operation. So, the system must adapt to the changes in the environment to prevent failure. For such cases, Aniculaesei et al. (2018) have introduced the concept of dependability cages in the study [S22] where unpredicted situations are captured and fed into the iterative development process as new development artifacts [22]. Similarly, cyber security is another concern that may be skipped during the development process. The study [S50] states that cyber risks should be identified and analysed although it is not feasible to deal with all the threats. Therefore, the study emphasizes modelling the threats using proper risk analysis tools [50]. The threats related to any computer system are mainly classified into three vulnerabilities: availability (data interruption), confidentiality (data interception), and integrity (data modification). An attacker could try to make network resources of an autonomous system unavailable, which is also known as a denial-of-service attack. The attacker could also change the behaviour of the autonomous systems by intercepting and modifying the instructions as shown in Figure 9. Similarly, system and users' data could be captured by an unauthorized person or organization [48]. For this reason, untrusted information or input from untrusted sensors should be blocked from reaching the subsequent software component and language-level information flow control helps to ensure the safety-critical operations without explicit verifications and endorsement [49].

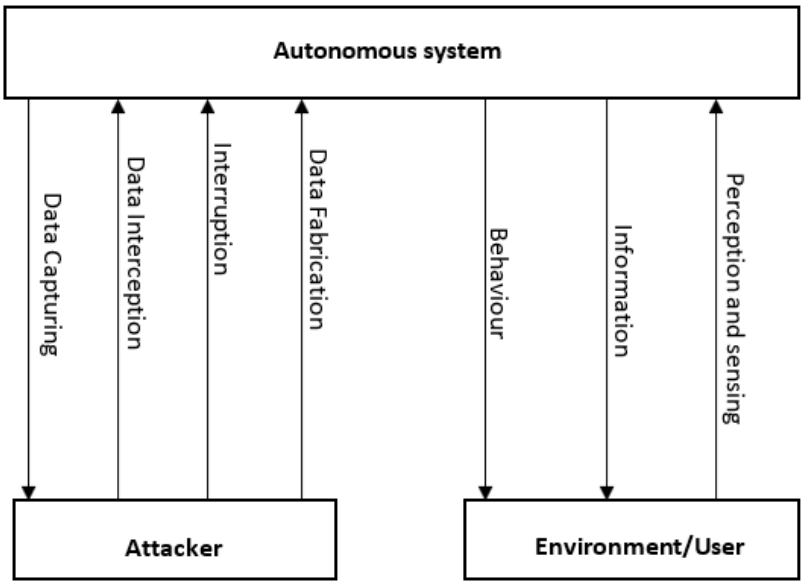

Figure 9: Conceptual model of the security attack

Furthermore, autonomous systems are designed to adapt to a dynamic environment. The systems must be able to execute appropriate actions depending on the external changes. Also, there could be a possibility to insert a new software component while the system is running through a connected network. Hence, there is a high risk of a malicious attacks on such cyber-physical systems which are considered safe and security-critical software systems. Thus, the study [S23] introduces a novel approach of using slack space of software components, for authentication of components to safely integrate with an autonomous system. This approach is designed to prevent the integration of tampered components into the system. According to the authors of the study [S23], traditional hash verification methods and graph-based state modelling approaches for the security of software components are not ample for large autonomous systems. The study has suggested storing a keyed hash of the component in its own slack space on the disk. The slack space is leftover space in the hard disk which is not used to store files. It is difficult for hackers or adversaries to attack slack space, as it requires administrative privileges. The authors have used a cryptographic hashing algorithm called HMAC to avoid hash collision attacks and such algorithms help to generate new hashes for the same file by changing the key. In addition, a randomized offset location technique is

used to store HMAC value in the slack space as an advanced feature in the presented approach [23].

Nowadays, autonomous vehicles can also communicate with each other or the infrastructure around them. This has been possible with the use of the latest communication technologies like dedicated short-range communications (DSRC). The autonomous vehicles with such capabilities are referred to as connected autonomous vehicles (CAVs). Such CAVs need to be protected from the potential external vulnerabilities which come from the environment that they operate in. So, all the functional behaviour of CAVs should be validated before on-road testing. The study [S38] describes the tools and methodologies for developing a robust autonomous system. The study has introduced SysWeaver and SysAnalyzer as vital tools in the development cycle of a CAV. SysWeaver is a model-based design, integration, and analysis framework for embedded real-time systems. It generates infrastructure and middleware code that binds all the software components together. Likewise, SysAnalyzer analyses the schedule of various components and ensures the timeliness of software components executed in a real-time computing environment. It can also assign and deploy software backups to ensure safety in presence of faults [38].

3.3. Machine learning methods

In autonomous systems, the systems must decide autonomously to react to a dynamic environment and such an environment is not precisely known during the development cycle. There might be various unforeseeable events that the system must deal with while in operation. The task to predict all the scenarios while in operation, is difficult during the development process. At the same time, for a safety-critical system, maintaining the safety requirements is also important along with system adaptations. Hence, many kinds of research have been done on various machine learning algorithms to employ them for the safety of autonomous systems and some of them are assessed in this chapter.

3.3.1 Reinforcement learning

The study [S1] presents a reinforcement learning approach with value function approximation and feature learning for autonomous decision-making of autonomous vehicles on highways. In this paper, the driving decision-making

process was modelled as a Markov decision process (MDP). The study [S1] also distinguishes the multi-objective approximate policy iteration (MO-API) approach from the multi-objective reinforcement learning (MORL) approach using data-driven feature representation for value and policy approximation to gain better learning efficiency of the autonomous system. MDP is considered as a standard for formalizing sequential decision-making problems using reinforcement learning (RL). MDP follows a Markov property where an agent depends only on the current state for the decision process, not on the full history of states and actions [26].

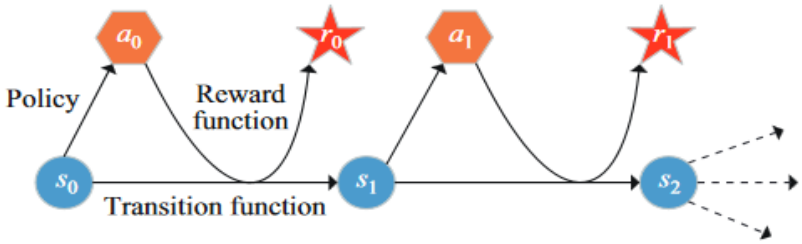

Figure 10. An overview of Markov decision process [26]

Figure 10 illustrates a Markov decision process where "a_0, a_1" represents agents, "s_0, s_1" represents states, and "r_0, r_1" represents where a_0 depends on s_0 to receive reward r_0. Similarly, a_1 depends only on s_1 to receive reward r_1, and so on.

In automation industry practice, there are usually many objectives that need to be optimized. It is difficult to optimize each of the objectives simultaneously. Thus, the MORL technique was developed to address the sequential decision-making problems with multiple objectives by learning from the experiences. However, due to the uncertainty and complexity of the dynamic environment, it is a challenging task for an autonomous system to find an optimal decision. Thus, an optimal policy for MDP is created after many iterations. This process is known as approximate policy iteration (API) [1].

Zuo et al. [S1] have also discussed other decision-making approaches such as fuzzy logic approach, probabilistic model approach, and supervised learning approach along with their limitation compared to the reinforcement learning approach. RL is a machine learning technique where an agent receives feedback

for every action that it does in an interactive environment by the trial-and-error method. In such a technique, correct actions are rewarded and over time, the agent learns to take the action with cumulative reward [1].

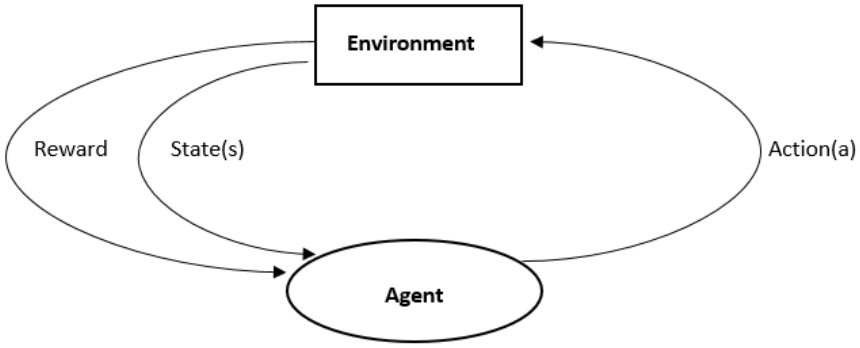

Figure 11: Reinforcement learning

As shown in Figure 11, a maximum reward is sought for an action (a) at a given state (s) and defined as action-value function Q (s, a). This function is known as Q-learning which is a variation of the reinforcement learning algorithm. The “Q” in Q-learning stands for quality and that defines the usefulness of a given action in gaining the maximum total reward. However, in real time, there can be many states and actions, and it is impossible to deal with them individually. Hence, an approximate Q-learning is applied using function approximators like a neural network, linear combinations of features, and decision tree [3], [5].

3.3.2 Deep Reinforcement learning

The traditional RL represents state-action-reward values in tabular form and such tabular representation can only cope with a simple environment where the number of states and actions are small. There might be a non-deterministic environment that is composed of huge numbers of states and actions. In real-world applications, it may lead to an issue due to memory or computational constraints. This problem in literature is referred to as “the curse of dimensionality”. Thus, to overcome such a situation, RL is combined with a deep neural network (DNN) to form deep reinforcement learning (DRL). The approximate value functions of traditional RL are represented as a parameterized functional form with a weight vector in DRL [2], [9], [26].

In the study [S6], the authors have proposed an approach for autonomous navigation of autonomous aerial vehicles in a large-scale complex unknown environment using the MDP concept and DRL algorithm. Using a simulation technique, the approach shows that the sensory information of the local environment and GPS signal helps to navigate the path from the starting position to the target position. The authors have suggested that such an approach for the autonomous vehicle does not need any path planning or map construction to follow [6].

In the study [S7], Mallozzi has proposed an approach for delegating part of the decision-making process on an autonomous system to reinforcement learning techniques while still maintaining the system's invariants. The main components of the proposed approach are shown in Figure 12.

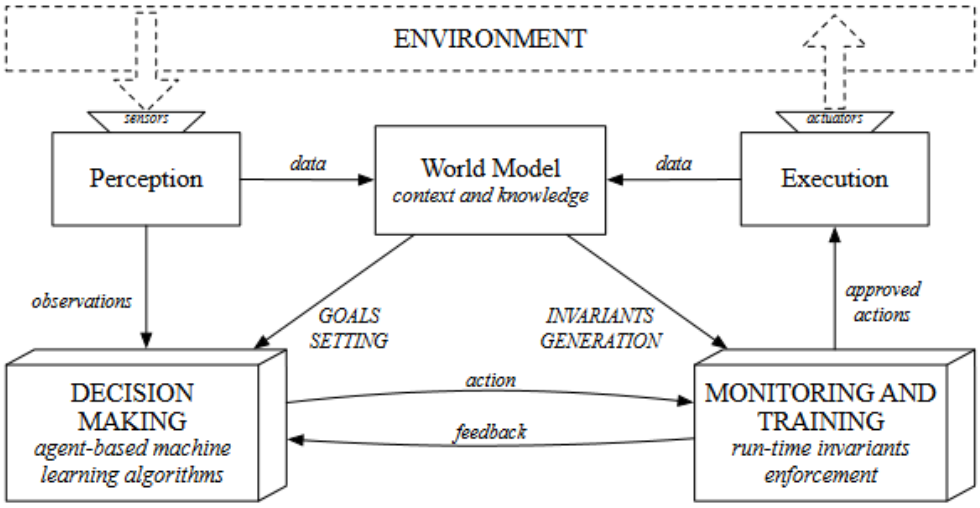

Figure 12: An overview of approach proposed by Mallozzi [S7]

Perception and execution are the components that interact with the environment through sensors and actuators. The sensors collect the raw data which are represented as perception. The action to the environment is defined as execution. The world model component in the proposed approach consists of information from both the external environment and the internal state of the system. It helps to set goals and maintain invariants of the system. The decision-making component acts based on the observations from perception and goals

set by the world model. The action generated by the decision-making component is not pre-set but learned from the feedback sent by the monitoring and training component. Likewise, the monitoring and training component monitors the actions taken by the decision-making component to preserve the system's invariants and trains the reinforcement learning algorithm by sending feedback for every action taken by the decision-making component. It also forwards appropriate action generated by the algorithm to execution [7].

Similarly, the study [S8] presents a survey on the DRL algorithm and its implementation on autonomous driving decision-making processes. The study also discusses the role of simulators in training agents, methods to validate and test along with real-world challenges. Various machine learning methods, such as value-based methods, policy-based methods, actor-critic methods, model-based and on/off-policy methods, are also overviewed in brief. In addition, the authors have focused on multi-agent reinforcement learning (MARL) techniques which have a high potential for high-level decision-making between groups of autonomous vehicles, as well as providing new opportunities for testing autonomous driving policies [8].

Deep reinforcement learning has been employed in many safety-critical systems like autonomous vehicles. According to Deshmukh et al. [S11], implementation of DRL itself does not ensure safe system behaviours. Hence, they have proposed a modified DRL approach to guarantee the system behaviours as on the safety requirements. A verification-in-the-loop RL mechanism to learn DNN controllers has been presented in the study [S11] using several examples.

3.3.3 Taint analysis

For an autonomous system, it is mandatory to have a performance guarantee, and to establish a performance guarantee, a formal verification must be done. Many software designers may lack the mathematical skillset to verify formally. Even if the software designers have the skill set, it is not enough for formal verification due to the uncertainty of the environment while in operation. Therefore, the authors of study [S24] have proposed automated monitoring and repairing of the autonomous robot software. A taint analysis and reinforcement learning (TARL) approach are proposed in the study. Taint analysis is an analysis

technique of a program that is implemented in compiler optimization and security analysis. The taint analysis starts with the storage location of a program for a potential attack which is known as a sink. A user input location is considered as a taint source. The taint analysis method is used by the authors to automatically extract the data flow sequence from the input topic to publish a topic, instrument the information to determine the software behaviour and ultimately repair the software in case of failure [24].

3.3.4 Use of software safety cages to train RL system

Another way to ensure the software safety of an autonomous system is to limit the control output based on the dynamic environment. The implementation of a software safety cage helps to improve the safety of an autonomous system by limiting outputs to a safe operational envelope. Such software safety cages do not interrupt the controllers' actions and degrade the overall performance of the systems. Instead, it removes all the unpredicted actions of the autonomous systems outside the safety cage and increases the confidence in the safety of autonomous vehicles. In addition, all interruptions by the software safety cages can be used for training deep reinforcement learning and generating robust learned policies [34]. Such safety cages for the safety and reliability of autonomous systems are proposed in the study [S34].

3.4. Safety assurance frameworks

Autonomous systems have a great deal of potential to offer to society. The number of autonomous systems has been increasing rapidly in recent years and extensive research is conducted to enhance the field further. However, an autonomous system might come across various challenges due to its dynamic behaviour. Many advanced technologies, such as machine learning and artificial intelligence, are applied to address the problems in autonomous systems but there is still much room for improvement. The studies [S19] and [S51] have outlined some challenges and methods to ensure the safety assurance of autonomous systems.

The study [S19] proposes a new framework for the safety assurance of autonomous systems in contrast to the challenges of safety assurance of autonomous systems. The study uses the machine learning technique to provide

the safety case for a dynamic system that updates dynamically according to the system behaviour. The term assurance in the study is defined as a system's confidence or capability to ensure safety. There can be various risks and hazards associated with an autonomous system. It is essential to identify hazards, safety risks associated and design the autonomous system to make it acceptably safe. Hence, McDermid et al. [S19] proposed a framework to provide a basis for assurance and regulation of autonomous systems that use machine learning or artificial intelligence in their development.

As shown in Figure 13, the framework has four main conceptually distinct components, although they are linked to each other.

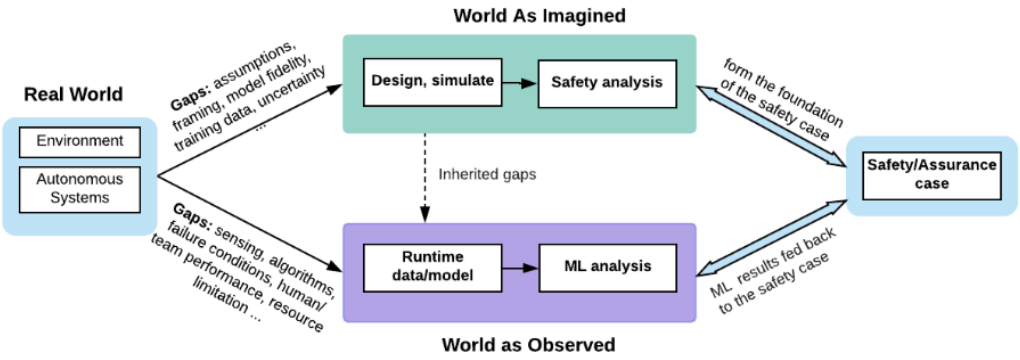

Figure 13: An overview of proposed safety assurance framework [S19]

The four main components are:

- Real world
- World as imagined
- World as observed
- Safety case

The "real world" component consists of autonomous systems in their operational environment. The "world as imagined" consists of designs and simulation models, and the results of safety analysis based on the models created. Similarly, the "world as observed" consists of operational data produced by the autonomous systems, such as images produced by the sensors, and the results of machine learning analysis of the data. Finally, a safety case is an initial reflection of the "world as imagined" which later is updated to the "world as

observed” and reduces the gap between the safety case and reality. The main challenge of the safety analysis is shown in Figure 13 as an inherited gap between “real world” and the “world as imagined”. The gap includes the assumptions about the system or environment, scoping the objects, preciseness, and accuracy of the models used, limitation of the training data, and the inability of the model to adapt to the environment due to “real world” complexity. Similarly, the “real world” differs from the “world as observed”. The gap includes sensing abilities, algorithmic limitations, failure conditions, limitations in human abilities, and resource limitations.

Autonomous systems that use machine learning or artificial intelligence gather a tremendous amount of data. These data are useful to understand the behaviour of the system and thus, to validate or refine the safety analyses in accordance with the system behaviour. In this way, the initial safety case referred to as the “world as imagined” would be updated to a dynamic safety case which is referred to as the “world as observed”. Therefore, the framework helps to improve the safety analysis based on the “world as observed" and to improve the data collection and machine learning analysis by providing feedback. Besides these, a safety case also needs to provide arguments and proof of training and testing data coverage in the operational design domain (ODD), from a safety perspective [19].

In the study [S51], Whalen and Heimdahl [S51] have described the role of automatic code generation in safety-critical systems. The study also outlines the requirements of such code generation to obtain a high level of confidence in the correctness of the translation process. The authors also describe a translator for a state-based modelling language called RSML (Root System Markup Language) which largely meets these requirements.

The possible reasons for flaws in safety-critical systems could be incorrect, incomplete, or ambiguous software system specifications. Formal specification languages can help to remove ambiguity to some extent. However, designing and developing a software product from formal specifications can be time-consuming and error-prone. Thus, the automatic code generation method was developed to provide consistency in behaviour of the production code in the

software development life cycle and be cost-effective. For safety-critical software, the code produced by an automatic code generator should have a high level of confidence and for this, the study has defined a set of requirements for creating a code generator as follows [51]:

- Requirement 1: The source and target languages must have formally well-defined syntax and semantics.
- Requirement 2: The translation between a specification expressed in a source language and a program expressed in a target language must be formal and proven to maintain the meaning of the specification.
- Requirement 3: The implementation of the translator must be verified to confirm that it correctly implements the translation.
- Requirement 4. The implementation of the translator must be rigorously tested and treated as high-assurance software.
- Requirement 5. The generated code must be well structured, well documented, and easily traceable to the original specification.

The generated code must be vivid, well documented, and retraceable to the original specifications so that a third-party tool for testing and inspections can verify the correctness of the code with respect to the formal model.

3.5. Formal methods

Many autonomous systems have been developed. However, the development of reliable software for autonomous systems continues to be challenged. The reliability of such safety-critical systems has been the foremost requirement of autonomous systems. Therefore, formal methods are used in recent safety-critical software developments to eliminate the ambiguousness and yet follow the required specifications. Such formal methods include testing, verification and validation of the software, and fault injection. The formal methods provide a high degree of confidence in safe system operation, if the verified model matches the real implementation of the system. The sixteen study papers that demonstrate the formal method techniques are included in this section.

The study [S13] highlights the importance of reliable software or high-integrity software in autonomous systems. The study has also emphasized on SPARK

programming language to develop high integrity software. According to Sward [S13], SPARK programming language helps to develop software with proof of correctness. In other words, the software developed using SPARK proves the implementation of requirements correctly. The author has used the formal method techniques offered by SPARK in his application to enumerate a flight plan for an autonomous system during reconnaissance [13]. Similarly, the study [S14] proposes techniques for quality assurance automation in automation systems. The proposed technique in the study [S14] generates test suites for the developed system and automates the fault localization along with the root cause of failures. In addition, the author has also proposed an evaluation of automated program repairing tools on autonomous systems [14].

In addition, the study [S15] presents an approach of assimilating formal verification results into safe reinforcement learning systems ensuring safety guarantees. In the paper, Fulton and Platzer [S15] have mentioned that the reinforcement learning method itself does not guarantee safe operations. Hence, they have introduced a technique called Justified Speculative Control (JSC) for combining formally verified models with reinforcement learning to transfer proofs of safety to learned policies.

The study [S16] is a systematic literature review paper on software verification and validation of safe autonomous cars. The paper, in general, defines safety certifications, and assesses the safe operation of autonomous systems including techniques to ensure software safety. Furthermore, the authors have also discussed the challenges of machine learning techniques in the development of safety-critical software. According to the paper, real-life test scenarios rarely produce robust software and thus simulation environments are considered as a viable solution to test the application under different conditions and environments. A tool called AsFAULT has been developed to automatically generate virtual tests for systematic testing of self-driving car software. The tool uses a search-based procedural content generation and is demonstrated by testing the lane-keeping functionality of an autonomous car software. The automatically generated virtual tests by AsFAULT trigger the behaviours of an autonomous system which, in return, helps to identify the safety-critical bugs and environmental conditions [27]. Likewise, the study [S16] also mentions a

simulation framework called MOBATSim that integrates fault injection and simulation together. The advantage of MOBATSim over other simulation tools is that it supports the fault injection method to verify whether the safety requirements are violated in the simulation [16].

3.5.1 Fault injection

The phenomenon of injecting faults into a system to assess its behaviour and to measure its efficiency of fault tolerance mechanism is known as fault injection. This technique is recommended by safety standards for the automotive society like ISO 26262. The technique of introducing faults in software is commonly knowns as software fault injection (SFI). In SFI, faults are injected into software by changing a small part of code and creating a different version of the program where each version has one injected software fault. This technique resembles the mutation testing technique but has different objectives. Mutation testing is done to identify an adequate test suite, while SFI is done to evaluate the system behaviour and validate fault-handling mechanisms at runtime. Mutation testing is mainly done during software development and SFI is a post-development event [33]. The study [S32] introduces a machine learning-based fault injection tool called DriveFI, which can detect the situations and faults that are most likely to impact the safety of autonomous vehicles.

3.5.2 Software testing and verification

An autonomous vehicle is an integration of various complex systems that encounters an unlimited number of unpredicted environments. Thus, it is essential to reduce all the possible hazards that might come across due to the dynamic environment. It is very challenging to verify and validate the system against all the possible hazards. However, researchers have been actively involved in verifying and validating autonomous systems. Various software testing methods like coverage-based testing, mutation testing, and functionality-based testing are used to test the developed software. In coverage-based testing, all the possible inputs are coverage criteria that are used for testing purposes. The possible inputs can be categorized into three coverage criteria: scenario coverage, situation coverage, and requirement coverage [30].

- Scenario coverage: Scenario coverage includes a set of linear evolution from one scene to another, such as lane change, following another vehicle, change in speed, and so on.
- Situation coverage: Situation coverage includes all the situations that can occur inside or outside the system and are tested with unexpected or expected situations.
- Requirement coverage: Such coverage includes a set of identified requirements that the system must meet.

Functionality-based testing of an autonomous vehicle comprises three classes: sensing, decision, and action functionalities. The sensor data received from the environment is considered as input to decision functionality and several decisions can lead to actions. In functionality-based testing, the scenarios can be broken down into various operational components which can be tested individually [47].

An autonomous vehicle or a cyber-physical system may encounter another unknown cyber-physical system and wants to connect to each other. The behaviours of the systems are barely predictable. Thus, autonomous real-time testing of the safety-critical systems is required to prevent any harmful action by the autonomous system. The study [S28] demonstrates an example of the Internet of Things (IoT) in relation to autonomous real-time testing. The authors have used combinatory the logic approach in the original test model to automatically extend it to a larger model based on the existing testing experiences. The combinatory logic approach is based on combinatory algebra and provides significant test coverage in cyber-physical systems [28].

Similarly, the study [S36] reviews the testing and verification techniques of neural network-based safety-critical software. The paper, in fact, is a review of approaches and tools used for testing and verification. According to the authors of the study [S36], formal verification of neural networks-based systems is usually presented as models and then model checkers, such as Boolean satisfiability (SAT) solvers, are applied to verify the safety property of the system. Formal verification approaches in software engineering are highly demanding, and also have scalability issues [36].

3.5.3 Simulation for verification

As discussed earlier, autonomous systems software can act in an unpredicted way depending on the environment. The complexity in autonomous systems is increasing while accommodating the users' various needs. Thus, it is more challenging to test the correctness of all features of the system in a real environment. Therefore, study [S37] suggests testing autonomous vehicle software in a virtual prototyping environment or in a simulator. In the paper, the authors have also presented an approach for automatic test case generation based on test criteria. The virtual prototyping of software can be tested at an early stage of development without integrating the software in a real vehicle. The test criteria should be defined based on environmental factors affecting the perception accuracy, the geometric roadway designs, and the behavioural aspects of the dynamic object in order to obtain a convincing result from virtual prototype testing [37].

Likewise, the study [S39] presents several verification activities during the development of fully autonomous heavy vehicles, including guidelines for verification of the decision-making process and their impact on the safety of the system. According to the author, formal verification methods guarantee some of the expected behaviour of the system, while testing and reviewing the system remain as dominant verification techniques due to the complexity of the system and cannot guarantee the correct implementation of the requirements in the system. Usually, the performance of testing is based on test coverage and does not cover the location of errors. In addition, an autonomous vehicle should be able to drive safely through all the situations that might come across and for this, a higher level of decision-making strategy is required. Due to the dynamic nature of the environment around autonomous vehicles, it is very challenging to define the expected behaviour or requirements of the system [39]. In addition, high-level data structures and complex algorithms are implemented in autonomous systems that support hardware/software co-design approaches. The study [S20] presents results of various experiments based on hardware/software co-designs which are formally verified and validated via simulation to determine safety and security factors for critical systems [20].

The simulation technique is used to verify the algorithms used in autonomous systems. However, detailed information about simulation techniques is not vivid in many cases. Therefore, using a test-case study of unmanned ground vehicles, the paper [S31] introduces a verification methodology based on statistical testing of autonomous vehicles' safety-related functionality in simulated scenarios. The study also serves as a discussion topic on the use of simulation tests for functional safety verification of autonomous vehicles. The methodology defined in the study is based on Statistical Scenarios testing using the Monte Carlo method in a simulated environment to reduce logistical complexity. The use of the Monte Carlo method helps to avoid statistical bias as a result of human presumption in the experiment [31].

4. Discussion

The systematic literature review performed in this paper has featured the methods to ensure software safety of critical autonomous systems. This paper has highlighted various aspects relevant to the safety of critical systems, such as system and architecture design of autonomous systems, machine learning techniques, safety standards and guidelines to ensure software safety, safety assurance frameworks, and formal methods. Many researchers have involved themselves in the field of autonomous systems and many of them have created a starting point on various topics related to the software safety of autonomous systems. Using machine learning and artificial intelligence, the field of autonomous vehicles has been revolutionized, but there is still much room for improvement. Besides these, the design and architecture of autonomous systems have advanced considerably. Six research questions have been defined to explore the methods to ensure software safety of autonomous systems:

- RQ1: Are there any guidelines to ensure software safety in autonomous vehicles? What kind of guidelines are available?
  The international functionality safety standard ISO 26262 and industrial standard IEC 61508 has been discussed along with the MISRA guidelines and model-based reference workflow to produce a safe and reliable software.

- RQ2: What is the role of system design and architecture in autonomous systems?

  Various system design strategies, software development processes and cyber security on connected autonomous systems were analysed to prevent the system failure and huge losses.
- RQ3: Evaluate available ML approaches for software safety of autonomous systems.

  Several machine learning techniques, such as RL, DRL, MORL, and MARL have been implemented on modern autonomous systems to adapt with the dynamic environment. In addition, use of taint analysis and safety cages to support the machine learning techniques have been evaluated.
- RQ4: Are there any frameworks or tools available to ensure software safety of autonomous systems? If so, what kind of frameworks are available?

  A safety assurance framework has been illustrated to ensure software safety of safety-critical systems. The importance of automatic code generation has also been emphasized in reducing risks and hazards associated to an autonomous system.
- RQ5: How does verification and validation of software help in software safety?

  A software is verified and validated against the required specification, and the required specifications are listed by the project team, only after proper research on system's use cases. This method ensures that the product satisfies the requirements. For a safety-critical system, safety is the main requirement. hence, the verification and validation approaches ensure the software safety.
- RQ6: Are there any other approaches that improve a safety critical system?

  The other approaches, such as fault injection and simulation techniques has helped to improve the quality of safety-critical systems.

Autonomous systems are facing new challenges because of dynamic nature of the environment. The technology used and standards developed do not seem to be good enough for new challenges. One of the challenges in an autonomous

system is the standardization of safety issues. Even a small change in an autonomous system may cause failure in the system. Thus, machine learning techniques may create instability in terms of the performance of the system. Also, the automotive functional safety standard of ISO 26262 was created before the emergence of artificial intelligence and hence, it can be concluded that ISO 26262 does not address the integration of artificial intelligence in autonomous vehicles. However, the machine learning models used in autonomous systems learn from a huge amount of data and store the model in a complex set of features combination. Therefore, it is infeasible to test autonomous systems thoroughly enough to ensure safety [29], [46].

A generic “V” software development model shown in Figure 2 has been considered as a development reference model in MISRA guidelines, which is also a basis for ISO 26262. However, fully autonomous systems face new challenges in mapping the technical aspects of the system to the V model. In addition, fully autonomous systems must be able to handle faults, malfunctions, and exceptional specified conditions. Hence, this has raised significant complexity and controllability challenges in fully autonomous systems [29].

An autonomous vehicle consists of various types of sensors, such as LiDARs, radars, GPS, and IR sensors. It is very important for a system to synchronize the data taken by several sensors so that they can react to external changes and adapt accordingly. The handling of sensor data taken in different frequencies and timestamp accuracy impacts an autonomous system’s performance. Another challenge is the data itself collected by the sensors. The data might not reflect the real scenario in case of disturbances. Also, the failure of the sensors might provide false data. Similarly, algorithm failure is another challenge, and such failure may occur due to extreme disturbances. All these failures might result in environmental catastrophes. Besides these, bad weather conditions, unforeseen road works, and emergency situations, such as road collapse, brake failure and tire blowout are possible threats to the safety of autonomous systems as manoeuvring the system is independent of human drivers [46].

With the rapid development of advanced computing technology, the existing cryptographic techniques cannot ensure data protection and secure connection

between the cyber-physically connected systems. Along with the development of new sensors, devices, infrastructures, and applications, the risk of attack in connected autonomous vehicles has grown too. Thus, top-notch security can be a topic for discussion in cyber-physical systems or connected autonomous systems. A connected autonomous system consists of powerful computing components including rich sensors. Thus, power management would be another challenge in such systems [46]. Formal methods are seen as a viable option to achieve completeness in the testing approach, but they do not fit well into complex systems, as they rely on sensors' accuracy and perception algorithms. However, implementation of safety cages, as well as monitor and actuator approaches, have been vital to offset the incompleteness in verification processes [16]. A monitor/actuator architecture is commonly used to handle a high-ASIL autonomy function.

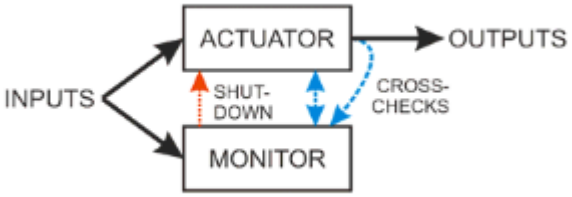

Figure 14: Monitor/actuator pair conceptual diagram [29]

As shown in Figure 14, in monitor/actuator architecture, the primary functions are performed by an actuator and the monitor validates the behaviour of the actuator. In a case where the actuator misbehaves, the monitor shuts down the system, resulting in a fail-silent system. In other words, the system shuts down if there is a fault. The use of two modules: monitor and actuator in such architecture are planned to prevent a misbehaving actuator from sending hazardous commands [29].

Furthermore, the verification of various aspects of a reinforcement learning algorithm can be divided into offline verification and online verification. The mapping of states to actions and action sequences can be verified offline by underlying the model of the system. However, in model-free algorithms, the model could be learned next to the value function. Since the model is in the form

of a Markov decision process (MDP), the requirements can be verified in some probabilistic temporal logic specifications. Using arbitrary data, the algorithms are verified theoretically and offline to ensure their behaviours. However, the attributes that cannot be verified offline are verified at runtime, although the runtime verification is not always feasible due to limitations in computation resources. At runtime verification, verifying the entire specification is not feasible. Thus, only the required aspects are verified at runtime [25].

5. Conclusion

This paper is a systematic literature review of methods to ensure software safety of autonomous systems. It includes relevant studies that were conducted since early 2000 until now. No specific technique exists that will build a robust and safe autonomous system alone, although there are various techniques that contribute towards the software safety of autonomous systems. Modern autonomous systems are designed to adapt the external changes in the environment. Many ways to verify and validate the model exist. However, the positive result from the verification of the system cannot guarantee the safety of the autonomous system. With the boom of new technologies and devices, new challenges are rising. The challenges regarding various aspects of autonomous systems are to be examined thoroughly to reduce the gap between simulation and the real environment.

This thesis has elaborated existing international functionality safety standard ISO 26262 and industrial standard IEC 61508 about the safety of autonomous systems. The guidelines to ensure software safety in the automotive domain are presented by MISRA reports. Additionally, various autonomous system designs and software architectures, as well as safety assurance frameworks that contribute towards the safety of the autonomous system, are observed. The application of reinforcement learning combined with deep neural networks in autonomous systems seems to be promising, because in using machine learning techniques, the system will learn to adapt to the dynamic environment with the help of data collected from various sensors. These can be verified using a diverse approach like formal verification and fault injection to some extent. Nevertheless, the usage of machine learning does not provide full confidence in

the system and is inadequate to ensure the safety of fully autonomous systems in all operating conditions. However, safety/dependability cages, as well as monitor/actuator approaches, can compensate for the incompleteness in verification processes.

Overall, the title of this thesis itself emphasizes the importance of software safety in autonomous systems. The topic “autonomous systems” has been a hot topic in recent years. There are still many aspects of autonomous systems to explore to build them with full confidence. This paper can serve as a starting point for the researchers and engineers who are working in safety-critical domains.

References

[1] Xu, X., Zuo, L., Li, X., Qian, L., Ren, J. and Z. Sun (2020) ‘A Reinforcement Learning Approach to Autonomous Decision Making of Intelligent Vehicles on Highways’, *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, Vol. 50, No. 10, pp. 3884-3897.

[2] Okuyama, T., Gonsalves, T. and Upadhay, J. (2018) ‘Autonomous Driving System based on Deep Q Learnig’, *2018 International Conference on Intelligent Autonomous Systems (ICoIAS)*, pp. 201-205.

[3] Lee, J., Kim, T. and Kim H. J. (2017) ‘Autonomous lane keeping based on approximate Q-learning', *2017 14th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, pp. 402-405.

[4] Beine, M. (2010) ‘A Model-Based Reference Workflow for the Development of Safety-Critical Software’, *Embedded Real Time Software and Systems*, Toulouse, France.

[5] Strauss, C. and Sahin, F. (2008) ‘Autonomous navigation based on a Q-learning algorithm for a robot in a real environment’, *2008 IEEE International Conference on System of Systems Engineering*, pp. 1-5.

[6] Wang, C., Wang, J., Zhang, X. and Zhang, X. (2017) ‘Autonomous navigation of UAV in large-scale unknown complex environment with deep reinforcement learning’, *2017 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pp. 858-862.

[7] Mallozzi, P. (2017) ‘Combining Machine-Learning with Invariants Assurance Techniques for Autonomous Systems’, *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pp. 485-486.

[8] Kiran, B. R., Sobh, I., Talpaert, V., Mannion, P., Sallab, A. A. A., Yogamani, S. and Perez, P. (2021) ‘Deep Reinforcement Learning for Autonomous Driving: A Survey’, *IEEE Transactions on Intelligent Transportation Systems,* pp. 1-18.

[9] Wang, P. and Chan, C. (2017) ‘Formulation of deep reinforcement learning architecture toward autonomous driving for on-ramp merge’, *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1-6.

[10] Vierhauser, M., Bayley, S., Wyngaard, J., Cheng, J., Xiong, W., Lutz, R., Huseman, J. and Cleland-Huang, J. (2018) ‘Interlocking Safety Cases for Unmanned Autonomous Systems in Urban Environments’, *2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)*, pp. 416-417.

[11] Deshmukh, J. V., Kapinski, J. P., Yamaguchi, T. and Prokhorov, D. (2019) ‘Learning Deep Neural Network Controllers for Dynamical Systems with Safety Guarantees: Invited Paper’, *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1-7.

[12] Pendleton, S.D., Andersen, H., Du, X., Shen, X., Meghjani, M., Eng, Y.H., Rus, D., Ang, M.H. (2017) ‘Perception, Planning, Control, and Coordination for Autonomous Vehicles’, *Machines 2017, 5, 6*.

[13] Sward, R. E. (2005) ‘Proving correctness of unmanned aerial vehicle cooperative software’, *Proceedings. 2005 IEEE Networking, Sensing and Control, 2005.,* pp. 767-771.

[14] Afzal, A. (2018) ‘Quality assurance automation in autonomous systems’, *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering,* pp. 948-951.

[15] Fulton, N. and Platzer, A. (2018) ‘Safe Reinforcement Learning via Formal Methods: Toward Safe Control Through Proof and Learning’, *The Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18),* Vol. 32, No. 1.

[16] Rajabli, N., Flammini, F., Nardone, R. and Vittorini, V. (2021) ‘Software Verification and Validation of Safe Autonomous Cars: A Systematic Literature Review’, *IEEE Access*, Vol. 9, pp. 4797-4819.

[17] Culley, J., Garlick, S., Esteller, E. G., Georgiev, P., Fursa, I., Sluis, I. V., Ball, P. and Bradley, A. (2020) ‘System Design for a Driverless Autonomous Racing Vehicle’, *2020 12th International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP)*, pp. 1-6.

[18] Fürst, S. (2019) ‘System/ Software Architecture for Autonomous Driving Systems’, *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*, pp. 31-32.

[19] McDermid, J.A., Jia, Y. and Habli, I. (2019) ‘Towards a Framework for Safety Assurance of Autonomous Systems’, *AISafety@IJCAI.*

[20] Hardin, D. S. (2020) ‘Verified Hardware/Software Co-Assurance: Enhancing Safety and Security for Critical Systems’, *2020 IEEE International Systems Conference (SysCon)*, pp. 1-6.

[21] Alberri, M., Hegazy, S., Badra, M., Nasr, M., Shehata, O. M.and Morgan, E. I. (2018) ‘Generic ROS-based Architecture for Heterogeneous Multi-Autonomous Systems Development’, *2018 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, pp. 1-6.

[22] Aniculaesei, A., Grieser, J., Rausch, A., Rehfeldt, K. and Warnecke, T. (2018) ‘Toward a Holistic Software Systems Engineering Approach for Dependable Autonomous Systems’, *2018 IEEE/ACM 1st International Workshop on Software Engineering for AI in Autonomous Systems (SEFAIAS)*, pp. 23-30.

[23] Beri, P. S. and Mishra, A. (2019) ‘Dynamic Software Component Authentication for Autonomous Systems using Slack space’, *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*, pp. 905-910.

[24] Lyons, D. and Zahra, S. (2020) ‘Using Taint Analysis and Reinforcement Learning (TARL) to Repair Autonomous Robot Software’, *2020 IEEE Security and Privacy Workshops (SPW)*, pp. 181-184.

[25] Wesel, P.V. and Goodloe, A. (2017) *‘Challenges in the Verification of Reinforcement Learning Algorithms’,* NASA [Online] https://ntrs.nasa.gov/citations/20170007190 (Accessed on 2 August 2021).

[26] Zhang, Z., Zhang, D. and Qiu, R. C. (2020) ‘Deep reinforcement learning for power system applications: An overview’, *CSEE Journal of Power and Energy Systems*, Vol. 6, No. 1, pp. 213-225.

[27] Gambi, A., Mueller, M. and Fraser, G. (2019) ‘AsFault: Testing Self-Driving Car Software Using Search-Based Procedural Content Generation’, *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pp. 27-30.

[28] Fehlmann, T. and Kranich, E. (2017) ‘Autonomous real-time software & systems testing’, *The 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement*, pp. 54-63.

[29] Koopman, P. and Wagner, M. (2016) ‘Challenges in Autonomous Vehicle Testing and Validation’, *SAE International Journal of Transportation Safety 4(1), pp.* 15-24.

[30] Tahir, Z. and Alexander, R. (2021) *Coverage based testing for V&V and Safety Assurance of Self-driving Autonomous Vehicles: A Systematic Literature Review* [Online] https://www.researchgate.net/publication/349913706_Coverage_based_testing_for_VV_and_Safety_Assurance_of_Self-driving_Autonomous_Vehicles_A_Systematic_Literature_Review (Accessed on 10 August 2021).

[31] Meltz, D. and Guterman, H. (2019) ‘Functional Safety Verification for Autonomous UGVs—Methodology Presentation and Implementation on a Full-Scale System’, *IEEE Transactions on Intelligent Vehicles*, Vol. 4, No. 3, pp. 472-485.

[32] Jha, S., Banerjee, S. S., Tsai, T., Hari, S. K. S., Sullivan, M. B., Kalbarczyk, Z. T., Keckler, S. W. and Iyer, R. K. (2019) 'ML-Based Fault Injection for Autonomous Vehicles: A Case for Bayesian Fault Injection', *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 112-124.

[33] Natella, R., Cotroneo, D., Duraes, J. A. and Madeira, H. S. (2013) 'On Fault Representativeness of Software Fault Injection', *IEEE Transactions on Software Engineering*, Vol. 39, No. 1, pp. 80-96.

[34] Kuutti, S., Bowden, R., Joshi, H., Temple, R.D. and Fallah, S. (2019) 'Safe Deep Neural Network-Driven Autonomous Vehicles Using Software Safety Cages', *2019 International Conference on Intelligent Data Engineering and Automated Learning*, pp. 150-160.

[35] Debouk, R., Czerny, B. and D'Ambrosio, J. (2011) 'Safety Strategy for Autonomous Systems', *International System Safety Society Conference*.

[36] Zhang, J. and Li, J. (2019) *Testing and verification of neural-network-based safety-critical control software: A systematic literature review* [Online] https://www.researchgate.net/publication/336577178_Testing_and_verification_of_neural-network-based_safety-critical_control_software_A_systematic_literature_review (Accessed on 12 August 2021).

[37] Kim, B., Kashiba, Y., Dai, S. and Shiraishi, S. (2017) 'Testing Autonomous Vehicle Software in the Virtual Prototyping Environment', *IEEE Embedded Systems Letters*, Vol. 9, No. 1, pp. 5-8.

[38] Bhat, A., Aoki, S. and Rajkumar, R. (2018) 'Tools and Methodologies for Autonomous Driving Systems', *Proceedings of the IEEE*, Vol. 106, No. 9, pp. 1700-1716.

[39] Gustavsson, J. (2016) 'Verification Methodology for Fully Autonomous Heavy Vehicles', *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pp. 381-382.

[40] MISRA (2001a) *Development Guidelines for Vehicle based Software* [Online] https://ia800108.us.archive.org/26/items/misradevelopmentguidelines/misra_dev_guidelines.pdf (Accessed on 18 August 2021).

[41] MISRA (2001b) *Software in Control Systems* [Online] https://ia800108.us.archive.org/26/items/misradevelopmentguidelines/misra_report4_sw.pdf (Accessed on 18 August 2021).

[42] MISRA (2001c) *Software Metrics* [Online] https://ia800108.us.archive.org/26/items/misradevelopmentguidelines/misra_report5_sw_metrics.pdf (Accessed on 18 August 2021).

[43] MISRA (2001d) *Verification and Validation* [Online] https://ia800108.us.archive.org/26/items/misradevelopmentguidelines/misra_report6_vv.pdf (Accessed on 18 August 2021).

[44] Vuori, M. (2011) *Agile Development of Safety-Critical Software* [Online] https://researchportal.tuni.fi/files/1102763/vuori_agile_safety-critical_software.pdf (Accessed on 22 August 2021).

[45] Salay, R., Queiroz, R. and Czarnecki, K. (2017) *An Analysis of ISO 26262: Using Machine Learning Safely in Automotive Software* [Online] https://arxiv.org/pdf/1709.02435 (Accessed on 23 August 2021).

[46] Liu, L., Lu, S., Zhong, R., Baofu, W., Yao, Y., Zhang, Q. and Shi, W. (2020) 'Computing Systems for Autonomous Driving: State-of-the-Art and Challenges', *IEEE Internet of Things Journal*, Vol. 8, No. 8, pp. 6469-6486

[47] Ebert, C. and Weyrich, M. (2019) 'Validation of Autonomous Systems', *IEEE Software*, Vol. 36, No. 5, pp. 15-23.

[48] Lera, F.J., Balsa, J., Casado, F., Fernandez, C., Rico, F.M. and Matellan, V. (2016) *Cybersecurity in Autonomous Systems: Evaluating the performance of hardening ROS* [Online] https://buleria.unileon.es/bitstream/handle/10612/5236/waf2016.pdf?sequence=1 (Accessed on 24 August 2021).

[49] Liu, J., Corbett-Davies, J., Ferraiuolo, A., Ivanov, A., Luo, M., Suh, G.E., Myers, A.C. and Campbell, M.E. (2018) ‘Secure Autonomous Cyber-Physical Systems Through Verifiable Information Flow Control’, *Proceedings of the 2018 Workshop on Cyber-Physical Systems Security and PrivaCy,* pp. 48-59.

[50] Madan, B. B., Banik, M. and Bein, D. (2016. ‘Securing unmanned autonomous systems from cyber threats’, *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology* [Online] https://journals.sagepub.com/doi/pdf/10.1177/1548512916628335 (Accessed on 25 August 2021).

[51] Whalen, M. W. and Heimdahl, M. P. E. (1999) ‘An approach to automatic code generation for safety-critical systems’, *14th IEEE International Conference on Automated Software Engineering*, pp. 315-318.

[52] Kitchenham, B., Pretorius, R., Budgen, D., Brereton, P., Turner, M., Niazi, M. and Linkman, S. (2010) ‘Systematic literature reviews in software engineering – A tertiary study’, *Information and Software Technology,* Vol 52, pp. 792-805.

[53] SAE International (2021) [Online] https://www.sae.org/standards/content/j3016_202104/ (Accessed on 18 May 2021).