UNIVERSIDADE DE SÃO PAULO

Instituto de Ciências Matemáticas e de Computação

A Software Architecture Proposal for Autonomous Vehicles

Iago Pachêco Gomes

Dissertação de Mestrado do Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional (PPG-CCMC)

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura: _____

Iago Pachêco Gomes

A Software Architecture Proposal for Autonomous Vehicles

Master dissertation submitted to the Institute of Mathematics and Computer Sciences – ICMC-USP, in partial fulfillment of the requirements for the degree of the Master Program in Computer Science and Computational Mathematics. *FINAL VERSION*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Prof. Dr. Denis Fernando Wolf

USP – São Carlos
December 2019

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados inseridos pelo(a) autor(a)

G633s Gomes, Iago Pachêco
 A Software Architecture Proposal for Autonomous
Vehicles / Iago Pachêco Gomes; orientador Denis
Fernando Wolf. -- São Carlos, 2019.
 146 p.

 Dissertação (Mestrado - Programa de Pós-Graduação
em Ciências de Computação e Matemática
Computacional) -- Instituto de Ciências Matemáticas
e de Computação, Universidade de São Paulo, 2019.

 1. Autonomous Vehicle. 2. Software Architecture.
3. Health Monitoring System. 4. Road Network Model.
5. Behavior Planning. I. Wolf, Denis Fernando,
orient. II. Título.

Iago Pachêco Gomes

Uma Proposta de Arquitetura para Veículos Autônomos

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências – Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientador: Prof. Dr. Denis Fernando Wolf

USP – São Carlos
Dezembro de 2019

ACKNOWLEDGEMENTS

I would like to thank God for all the achievements and experiences I experienced during this project.

I would also like to thank my family for their support and care, without which I could not achieve any goals.

In addition, I would also like to express my most sincere thanks to Professor Denis Fernando Wolf, for his guidance and patience, which were essential during my masters degree. Indispensable were also all the support and friendship of the LRM/LSEC and UESB (State University of Southwestern Bahia) friends.

I would also like to thank all the staff of the Institute of Mathematics and Computer Sciences (ICMC) of the University of São Paulo (USP), especially the employees of the postgraduate secretary of the Graduate Program in Computer Science and Computational Mathematics (CCMC).

Finally, I would like to thank the National Council for Scientific and Technological Development (CNPq) for the financial funding of this project.

*“Imagination is more important than knowledge.
For knowledge is limited to all we now know
and understand, while imagination embraces the entire world,
and all there ever will be to know and understand”
(Albert Einstein)*

RESUMO

IAGO, P. G. **Uma Proposta de Arquitetura para Veículos Autônomos**. 2019. 143 p. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2019.

Os Veículos Autônomos devem transformar o cenário do transporte urbano, aumentando sua eficiência, tornando-o mais acessível e seguro, diminuindo o impacto ambiental, entre outros benefícios. Para tanto, eles são compostos por um conjunto de dispositivos físicos e algoritmos, que os permitem compreender o ambiente externo, seu próprio estado e interagir com outros agentes do trânsito. Por sua vez, estes componentes precisam colaborar uns com os outros para garantir o pleno funcionamento do veículo, o que faz com que o seu desenho arquitetural seja uma etapa imprescindível para o projeto e desenvolvimento. A arquitetura de *software* apresenta a estrutura e organização dos componentes, possibilitando que avaliações e decisões sejam tomadas antes da implementação. Outro aspecto importante, é que tais sistemas estão sujeitos a falhas que podem afetar todo o desempenho do veículo, por conta disso, é essencial detectar e diagnóstica-las. Desta forma, o objetivo deste trabalho é desenvolver uma nova arquitetura para o *CaRINA 2*, o veículo autônomo da Universidade de São Paulo, agregando novas funcionalidades desenvolvidas no grupo de pesquisa e atentando-se principalmente para os requisitos de segurança do sistema, que consiste em detectar e diagnosticar as falhas. Além disso, uma nova estrutura de mapa e um novo método para desenvolvimento dos comportamentos e manobras foram incluídos à arquitetura.

Palavras-chave: Veículos Autônomos, Arquiteturas de *Software*, Detecção e Diagnóstico de Falhas, Descrição de Mapas, Planejamento de Comportamento.

ABSTRACT

IAGO, P. G. **A Software Architecture Proposal for Autonomous Vehicles**. 2019. 143 p. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2019.

Autonomous Vehicles should transform the urban transport scenario, increasing its efficiency, making it more accessible and safe, reducing environmental impact, among other benefits. To do so, they are composed of a set of hardware and algorithms, which allow them to understand the external environment, their own state and interact with other traffic agents. In turn, these components need to collaborate with each other to ensure the full operation of the vehicle, which makes its architectural design an essential step for the project and development. The software architecture presents the structure and organization of the components, enabling evaluations and making decisions prior to implementation. Another important aspect is that such systems are subject to faults that can affect the whole performance of the vehicle, therefore, it is essential to detect and diagnose them. In this way, the objective of this work is to develop a new architecture for the *CaRINA 2*, the autonomous vehicle of the University of São Paulo, adding new functionalities developed by the research group and focusing mainly on safety requirements, which consists of detecting and diagnosing faults. In addition, new map structure and a new method for behavior and maneuver development were also included to the architecture.

Keywords: Autonomous Vehicle, Software Architecture, Fault Detection and Diagnosis, Road Network Description, Behavior Planning.

LIST OF FIGURES

Figure 1 – Winners of DARPA’s Grand Challenge and DARPA’s Urban Challenge . . .	28
Figure 2 – Examples of Autonomous Vehicles in Development	28
Figure 3 – Basic Flow of Actions for an Autonomous Vehicle	29
Figure 4 – Basic Components for an Autonomous Vehicle	30
Figure 5 – <i>Tongji</i> Autonomous Vehicle	37
Figure 6 – <i>Tongji</i> Software Architecture	38
Figure 7 – A1 Autonomous Vehicle	39
Figure 8 – A1 Software Architecture	40
Figure 9 – <i>BerthaOne</i> Autonomous Vehicle	41
Figure 10 – <i>BerthaOne</i> Software Architecture	41
Figure 11 – IARA Autonomous Vehicle	42
Figure 12 – CMU Autonomous Vehicle	43
Figure 13 – CMU Software Architecture	44
Figure 14 – Hierarchy of decision-making processes for an Autonomous Vehicle	46
Figure 15 – Parallel Navigation Architecture	47
Figure 16 – Hybrid Navigation Architecture	48
Figure 17 – OpenStreetMap	52
Figure 18 – OpenDRIVE	53
Figure 19 – Comparison of Finite-State-Machine and Behavior-Tree	55
Figure 20 – Behavior Tree Elements	56
Figure 21 – Fault Types	59
Figure 22 – Model-based General Schematic	60
Figure 23 – Knowledge-based Classification	61
Figure 24 – Example of Bayesian Network for Diagnosis	62
Figure 25 – Types of reasoning in Bayesian Networks	64
Figure 26 – CaRINA II and the Smart Truck	69
Figure 27 – Earlier CaRINA II Software Architecture	70
Figure 28 – CaRINA II Software Architecture	73
Figure 29 – Human Machine Interface for CaRINA II	75
Figure 30 – Map XML Diagram	78
Figure 31 – Map Localization using Bucket Point Region Quad-Tree	81
Figure 32 – Obstacle Message Class Diagram	82
Figure 33 – Obstacle Detection Activity Diagram	83

Figure 34 – CaRINA II Navigation Architecture	84
Figure 35 – Directed Multigraph USP Campus 2	85
Figure 36 – Behavior Tree for Traffic Signs	88
Figure 37 – Behavior Tree for Emergency Stop	89
Figure 38 – Component Based Health Monitoring Hierarchy	92
Figure 39 – Health Monitoring System Interface	93
Figure 40 – Control Diagnostic Hybrid Bayesian Network	97
Figure 41 – GPS Diagnostic Hybrid Bayesian Network	99
Figure 42 – Control Diagnosis Experimental Results - Case 1	112
Figure 43 – Control Diagnosis Experimental Results - Case 2	114
Figure 44 – GPS Diagnosis Experiment - Case 3	117
Figure 45 – GPS Diagnosis Experimental Results - Case 3	119
Figure 46 – University of São Paulo - Campus 2	120
Figure 47 – Speed and Curvature of the Route	121
Figure 48 – Histogram of the Speed and Curvature Error	122

LIST OF TABLES

Table 1 – Autonomy Levels: SAE-J3016	24
Table 2 – Map Features	79
Table 3 – Map Services	79
Table 4 – Behavior Tree Composites for Speed Bump Sign	88
Table 5 – Behavior Tree Composites for Emergency Stop	90
Table 6 – DOP Ratings	100
Table 7 – Control Diagnosis Dataset Division in Training and Test	109
Table 8 – Control Diagnosis Confusion Matrix	109
Table 9 – Control Diagnosis Fault Detection Result - Case 1	111
Table 10 – Control Diagnosis Fault Detection Result - Case 2	113
Table 11 – GPS Diagnosis Dataset Division in Training and Test	116
Table 12 – GPS Diagnosis Confusion Matrix	116
Table 13 – GPS Diagnosis Fault Detection Result - Case 3	118

LIST OF ABBREVIATIONS AND ACRONYMS

AVs	Autonomous Vehicles
BN	Bayesian Network
BT	Behavior Tree
CaRINA II	Intelligent Robotic Car for Autonomous Navigation
CPT	Conditional Probability Table
DGPS	Differential Global Position System
DOP	Dilution of Precision
EKF	Extended Kalman Filter
EM	Expectation Maximization
FD	Fault Diagnosis
FDD	Fault Detection and Diagnosis
FSM	Finite-State Machine
GDOP	Geometric Dilution of Precision
GNSS	Global Navigation Satellite System
GPS	Global Position System
HDM	High Definition Maps
Hybrid-BN	Hybrid Bayesian Network
IF	Intermittent Fault
IMU	Inertial Measurement Unit
LiDAR	Light Detection And Ranging
MTE	Mixtures of Truncated Exponentials
PF	Permanent Fault
PI	Proportional-Integral
PID	Proportional-Integral-Derivative
RADAR	Radio Detecting and Ranging
RAIM	Receiver Autonomous Integrity Monitoring
ROS	Robot Operating System
RTK	Real-Time Kinematic
TF	Transient Fault
V2I	Vehicle-to-Infrastructure Communication
V2V	Vehicle-to-Vehicle Communication

CONTENTS

1	INTRODUCTION	23
1.1	Contributions	25
1.2	Dissertation Overview	26
2	AUTONOMOUS VEHICLES	27
2.1	Autonomous Vehicles System	27
2.1.1	<i>Sensing</i>	29
2.1.2	<i>Perception</i>	30
2.1.3	<i>Decision-Making</i>	31
2.1.4	<i>Control</i>	31
2.1.5	<i>Mapping</i>	31
2.1.6	<i>Human-Vehicle Interface</i>	32
2.1.7	<i>System Monitoring</i>	32
2.1.8	<i>Support Platform</i>	33
2.2	Connected and Autonomous Vehicles	33
2.3	Final Considerations	34
3	AUTONOMOUS VEHICLES ARCHITECTURES	35
3.1	Robotic Architectures	35
3.2	Autonomous Vehicle Architectures	37
3.2.1	<i>Tongji</i>	37
3.2.2	<i>A1</i>	39
3.2.3	<i>BerthaOne</i>	40
3.2.4	<i>IARA</i>	42
3.2.5	<i>CMU</i>	43
3.3	Navigation Architectures	45
3.4	Final Considerations	48
4	TECHNICAL BACKGROUND	51
4.1	Road Network Description	51
4.1.1	<i>OpenStreetMap</i>	51
4.1.2	<i>OpenDRIVE</i>	52
4.1.3	<i>Lanelets</i>	53
4.2	Behavior Planning and Execution	54

4.2.1	<i>Behavior Tree</i>	56
4.3	Diagnostic System	58
4.3.1	<i>Fault Detection and Diagnosis</i>	58
4.3.2	<i>Model-based Approaches</i>	59
4.3.3	<i>Signal-based Approaches</i>	60
4.3.4	<i>Knowledge-based Approaches</i>	61
4.3.5	<i>Bayesian Networks</i>	62
4.4	Robot Operating System - ROS	65
4.5	Final Considerations	67
5	CARINA II	69
5.1	Earlier CaRINA II Architecture	70
5.2	Proposed CaRINA II Architecture	72
5.2.1	<i>Sensing Layer</i>	72
5.2.2	<i>Vehicle Layer</i>	74
5.2.3	<i>System Management Layer</i>	74
5.2.4	<i>Control Layer</i>	75
5.2.5	<i>Mapping Layer</i>	76
5.2.5.1	<i>Road Network Description</i>	77
5.2.5.2	<i>Map Service</i>	78
5.2.6	<i>Perception Layer</i>	80
5.2.6.1	<i>Localization</i>	80
5.2.6.2	<i>Obstacle Detection</i>	81
5.2.6.3	<i>Traffic Signs Supervisor</i>	83
5.2.7	<i>Decision Making and Planning Layer</i>	83
5.2.7.1	<i>Route Planning</i>	84
5.2.7.2	<i>Behavior Planner</i>	85
5.2.7.3	<i>Path Planning</i>	86
5.2.7.4	<i>Behaviors</i>	87
5.3	Final Considerations	89
6	DIAGNOSTIC AND MONITORING SYSTEM	91
6.1	Health Monitoring System	91
6.2	Fault Detection and Diagnosis Methodology	94
6.2.1	<i>Control Diagnostic</i>	95
6.2.2	<i>GPS Diagnostic</i>	98
6.3	Final Considerations	105
7	RESULTS	107
7.1	Diagnostic System	107

7.1.1	<i>Control Monitor</i>	108
7.1.1.1	<i>Case 1</i>	110
7.1.1.2	<i>Case 2</i>	111
7.1.2	<i>GPS Monitor</i>	113
7.1.2.1	<i>Case 3</i>	116
7.2	Software Architecture	120
7.3	Final Considerations	122
8	CONCLUSION	125
	BIBLIOGRAPHY	127
APPENDIX A	PUBLICATIONS	143

INTRODUCTION

Autonomous Vehicles (AVs) are intelligent and robotic vehicles, which do not require human direction or supervision to drive on streets, thus being able to deal with traffic laws, respect signs, interact with other vehicles, pedestrians, etc. In order to achieve so many objectives, they are made up of a set of sensors, actuators and algorithms that provide the necessary tools for the autonomous system to be able to represent and understand the environment and to carry out the planned actions.

With the available data from the environment, and the computational performance of the algorithm used to convert the data into useful information, Autonomous Vehicles should improve the urban traffic, since they will make better use of road space, will be able to avoid accidents, coordinate their actions to improve the traffic flow, improve accessibility, as they do not need humans to driver in traffic, among other benefits and applications.

However, in the current state of the research field, there are still no vehicles that meet all these goals, the so-called fully-autonomous, or level 5 of autonomy. Most of the current projects are still prototypes and research vehicles. Still, there are other initiatives that have already begun to use them in public roads or in more controlled environments. In addition, in these projects the models of vehicles go from conventional cars to luxury cars, and also to trucks, buses, agricultural vehicles, among others ([LUTIN, 2018](#)).

The autonomy levels of the vehicle are determined by the number of functions it can perform autonomously, that is, without human intervention. This levels are classified by the international standard SAE-J3016, showed in Table 1. Level 0 presents vehicles without automation. At levels 1 and 2, a single specific feature (steering, acceleration or braking), or more than one is controlled by the vehicle at the same time, in this order. For level 3, the autonomous system is able to fully control the vehicle, although when an unexpected situation occurs the vehicle gives the control back to the driver, who needs to be paying attention to autonomous driving. Finally, for levels 4 and 5, the vehicle is called fully-autonomous, which means that it must deal with any

situation, including malfunction of its components. The difference between levels 4 and 5 is that for the former, the driver can choose to disable the autonomous operation and get the control back, but for level 5 this option is not available.

Table 1 – Autonomy Levels: SAE-J3016

Level	Name	Description
<i>Level 0</i>	without automation	all driving responsibilities are assigned to the driver
<i>Level 1</i>	driving assistance	automatic control of a single subsystem
<i>Level 2</i>	partial driving automation	automatic control of more than one subsystem
<i>Level 3</i>	conditional driving automation	fully control of the vehicle at specific scenarios and environment
<i>Level 4</i>	high driving automation	fully control of the vehicle when solicited
<i>Level 5</i>	fully driving automation	fully control of the vehicle at any time

Source: Adapted from [SAE \(2014\)](#).

This classification is very important for the analysis of the current research and state-of-art on Autonomous Vehicles, knowing that several car assemblers already offer models that comply with level 2 (such as the parking assistant, or adaptive cruise control), and some already level 3, such as Tesla ([MERCER; MACAULAY, 2018](#)).

One matter worth mentioning is that at level 3, when a situation of risk or an unexpected event occurs, the system returns the control to the driver. But starting at level 4, it must be able to deal with all these situations by itself, and this includes, for example, failure or instability in sensors or software components. This responsibility adds a lot of complexity to the autonomous systems, which need to have many algorithms to do the analysis of the environment and planning, as well as constantly monitor the integrity of their components.

Another issues raised by the increase of the autonomy of the autonomous system, is that many different behaviors must be present in the system, which makes the choice of techniques for behavior planning and the maneuver execution a critical tactical decision in the design and implementation of the system, whereat the flexibility, reusability, scalability and maintainability are important matters. Besides that, a good and rich representation of the environment is also an essential component for the autonomous system, this is present in the map structure, also known as road network description, which stores information of the road geometry, surrounding elements of the roads, and semantic or context information such as traffic signs and traffic controllers (e.g. traffic lights).

In view of so many components and aspects of the autonomous system, the need for an architectural design that studies the relationship between them is essential to ensure the

full functioning of the Autonomous Vehicles. Thus, this work aims to redesign the software architecture of *CARINA II* (*Intelligent Robotic Car for Autonomous Navigation*, an autonomous vehicle of the University of São Paulo, developed by the Mobile Robotic Laboratory), as well as to develop the system for Fault Detection and Diagnosis for the GPS (Global Position System) and the controllers. The GPS is used for localization of the vehicle, in turn, the controllers are responsible to generate the brake, throttle, and steering commands. In addition, another contribution of this work is to create a structure for the road network description and for behavior execution.

Regarding the software architecture, the study of styles and patterns is a very important step, since each one has a set of characteristics that regulate the organization of the components and the way in which they interact. These characteristics must be respected and executed when the architecture is implemented. In turn, the fault detection and diagnosis of the system is a very well-founded area, having several approaches, including the use of Bayesian Networks, which combines the characteristics of probabilistic models, representing very well the cause and effect relationship between faults and symptoms.

Furthermore, different standards of map description have been reviewed (i.e, OpenStreetMap, OpenDRIVE and Lanelets), but none of them fit the desired needs, which led to the creation of a model of its own. Finally, the choice of techniques for modeling the behavior of the Autonomous Vehicle is very important for the continuity of the work in the *CaRINA II*, since it is increasingly aimed at increasing the autonomy of the vehicle with new works. Due to this, an analysis of the Finite State Machines and Behavior Trees has been carried out, among which the latter presents better characteristics when considering development criteria for more complex systems, being chosen to model the behaviors of the vehicle.

1.1 Contributions

The main purpose of this Masters dissertation is to present a new design of the software architecture for *CaRINA II*. Besides that, three specifics contributions have been proposed in the new architecture, being:

- **Road Networks Description:** a new model for road networks description has been developed, based on the work of [Silva and Grassi \(2018\)](#), to meet the needs of the path planning, behavior planning, and the mapping modules of the software architecture. In addition to the description of the road network geometry and topology, semantic information has also been added, aligning the model to augmented map standards, which is a very important feature for Autonomous Vehicles.
- **Fault Detection and Diagnosis:** Hybrid Bayesian Networks with Residual Evidence have been developed to detect the presence of faults in the controllers and the GPS sensor.

These networks are components of the Health Monitoring System, and they combine two different approaches for Fault Detection and Diagnosis, classified as model-based and knowledge-based techniques, using the characteristics of each method to facilitate detection of the faults patterns.

- **Behavior Modeling:** since the Autonomous Vehicles need to be able to perform many different maneuvers, it is desired to use techniques to develop this action that is more modular, flexible, scalable, and easy-to-maintain. Therefore, Behavior-Trees, a well-known technique in the Game Industry, has been studied and applied to accomplish this task.

1.2 Dissertation Overview

In addition to this chapter the work is divided into:

- **Chapter 2** introduces the basic concepts and components of the Autonomous Vehicles and Connected Vehicles;
- **Chapter 3** analyses some architectural models for Autonomous Vehicles;
- **Chapter 4** presents the main theoretical concepts applied in this master degree project, including the discussion about *map design*, *behavior architecture*, and *fault detection and diagnosis* techniques;
- **Chapter 5** presents the earlier and new designs of the *CaRINA II*;
- **Chapter 6** discusses the *health monitoring module* and the diagnostic components of the new architecture;
- **Chapter 7** presents and discusses the results of this works;
- **Chapter 8** presents the conclusions of the work.

AUTONOMOUS VEHICLES

This chapter provides basic concepts related to Autonomous Vehicles (AVs), in order to highlight which components are essential for the autonomous system and its architectural organization. In this way, it is possible to understand with clarity the importance of the system architectural design, as well as introduces the discussion about the related works presented in Chapter 3 and the architectural proposal of *CaRINA II*, presented in Chapter 5.

2.1 Autonomous Vehicles System

Autonomous Vehicles (AVs) are intelligent and robotic vehicles capable of navigating in urban traffic, highways, and so on, without the need of human intervention, dealing with all adversities and respecting the traffic laws (LIU *et al.*, 2017). In favor of being successful in these objectives, they combine a variety of technologies from different areas of knowledge such as Computer Science, Mechanical Engineering, Electrical Engineering, Automotive Engineering, among others (BIMBRAW, 2015).

Although they are not yet present in all roads and cities, the viability and technological trend is undeniable, given the great progress in research and development achieved in recent years, as well as the commitment of government authorities, research institutions, and private companies in the promotion and deployment of the technology (LUTIN, 2018; FAGNANT; KOCKELMAN, 2015; ANDERSON *et al.*, 2014).

A great milestone for the area were two major events known as *DARPA's Grand Challenge* and *DARPA's Urban Challenge*, which were promoted by the DARPA (*Defense Advanced Research Projects Agency-USA*) (FAGNANT; KOCKELMAN, 2015). The first one occurred twice, in the years 2004 and 2005, with the purpose of showing the viability of the technology challenging the participants to navigate in an off-road environment. There were no winners in the 2004 edition. Upon the 2005 edition, the winner was the *Stanley* vehicle (Fig. 1a) from Stanford

University (THRUN *et al.*, 2006).

The DARPA's Urban Challenge occurred in 2007, with BOSS vehicle (Fig. 1b) from Carnegie Mellon University as the winner. In this edition, the participating teams were challenged to develop vehicles capable of navigating in an urban environment, respecting traffic-signs and dealing with some common scenarios in the day-to-day traffic (BUEHLER; IAGNEMMA; SINGH, 2009).

Figure 1 – Winners of DARPA's Grand Challenge and DARPA's Urban Challenge

Source: Thrun *et al.* (2006), Urmson *et al.* (2009).

Since then, the study of Autonomous Vehicles became the focus of several research groups, vehicle assemblers, such as Mercedes-Benz, Volkswagen, Ford (Fig. 2a), Tesla, and information technology companies, such as Google (Fig. 2b), Baidu, Uber (Fig. 2c), etc. (MERCER; MACAULAY, 2018; PHILIP; SARAVANAGURU, 2018; ANDERSON *et al.*, 2014).

Figure 2 – Examples of Autonomous Vehicles in Development

Source: Hawkins (2018), Waymo (2018), Johnson and Fitzsimmons (2018).

Fagnant and Kockelman (2015), Bagloee *et al.* (2016) and Kistangari *et al.* (2017) point out that the reason for such popularity is the benefits that the AVs offer users, and consequently the entire urban infrastructure. The authors list some of them as being: reduction of traffic accidents; improvement in road usage; traffic jam reduction; increase in accessibility; better use of time spent during trips; reduction in the emission of pollutants and better use of energy consumption (in the case of electric cars); among many others.

To accomplish all objectives, the autonomous systems are composed by a set of algorithms and technologies, which, according to Kato *et al.* (2015) e Liu *et al.* (2017) can be

categorized as sensing; perception; planning; decision-making; and, human-vehicle interface. [Anderson et al. \(2014\)](#) explain that Autonomous Vehicles inherited from robotics what can be considered their basic behavior, or basic action flow, shown in Figure 3, which consists of: *Sense*; *Plan*; and, *Act* ([MURPHY; MURPHY, 2000](#); [SIMPSON; JACOBSEN; JADUD, 2006](#)).

Figure 3 – Basic Flow of Actions for an Autonomous Vehicle

Source: Adapted from [Simpson, Jacobsen and Jadud \(2006\)](#).

Thus, the AVs uses several sensors to measure data from its surrounding, which are used by a set of algorithms to extract useful information, and which, in turn, feed a collection of planning and decision-making algorithms, for then generate commands executed by the controllers ([LIU et al., 2017](#)). As stated in [Anderson et al. \(2014\)](#), the flow-chart showed in Figure 3 is repeated several times, even in parallel, among the numerous components that allow the vehicles to reach their goals.

Figure 4 shows a basic design of software architecture for Autonomous Vehicles, which has some commons components groups, also called layers, present in most of the software architecture of autonomous mobile robotics and the related works present in chapter 3. This architecture is organized as: *sensing*; *perception*; *decision-making*; *control*; *mapping*; *human-vehicle interface*; *system monitoring*; and, the other softwares responsible to coordinate the execution of the vehicle, called *robotic middleware* and *hardware platform*, which gives support for the whole autonomous system.

2.1.1 Sensing

The sensing layer of an autonomous system is composed of a set of sensors responsible for measure data from the external environment or from the vehicle itself, which can be classified, respectively, as exteroceptive and proprioceptive sensors ([SOLUTIONS, 2007](#)). The main functionality of these devices is to obtain data to be interpreted and converted into useful information by the *Perception* layer, allowing the system to create a representation of the vehicle state and its surroundings with a certain degree of reliability ([ANDERSON et al., 2014](#)).

An important fact of sensing layer, highlighted by [Ji \(2018\)](#), is that there are several sensors in robotics, which one capable of measure specifics kinds of data from the environment or the vehicle. Thus, the sensory ability of autonomous systems are better than the ability of human drivers, which increases the system efficiency in dealing with traffic situations.

Figure 4 – Basic Components for an Autonomous Vehicle

Source: Elaborated by the author.

Some of the sensors are very commonly used, e.g.: GPS (Global Position System) and IMU (Inertial Measurement Unit) which measure and estimate the position of the vehicle; Cameras, to detect and track obstacles, traffic signs, and traffic lights, navigable area, and so on; LiDAR (Light Detection And Ranging), used for creating maps, detect obstacles, estimate the pose of the vehicle, among others; and, RADAR (Radio Detecting and Ranging), also used for obstacle detection and tracking (BRUMMELEN *et al.*, 2018; KUUTTI *et al.*, 2018).

2.1.2 Perception

The perception layer uses a set of algorithms from different disciplines such as Computer Vision, Computer Graphics, Probability, and Machine Learning, to extract useful information from the data measured by the sensors (BRUMMELEN *et al.*, 2018). This information is used to create a representation of the vehicle state and its surroundings (ANDERSON *et al.*, 2014), required by the components of the decision-making layer.

According to Liu *et al.* (2017), the main components of perception are the localization, and detection and tracking of obstacles. However, other features can be assigned to the layer, e.g.: scene segmentation, to differentiate roads and sidewalks, and so on; traffic signs and traffic lights detection; mapping; and stereo vision for estimates the depth from camera images and detect obstacles, or navigable area. An important fact pointed out by Brummelen *et al.* (2018) is the need

for fusion algorithms to combine information from different sensors ensuring more reliability, accuracy, and robustness. Kalman Filter and Bayesian Filter are well known approaches to achieve that.

2.1.3 Decision-Making

The decision-making and planning algorithms are responsible to analyze all information provided by the *Perception* components and choose actions which lead the vehicle to fulfill its objectives and also concerning about safety of the passengers and other transit agents (LIU *et al.*, 2017). To do so, this information is analyzed according to different heuristics, which can, for example, lead to the planning of routes and trajectories, to avoid obstacles, the evaluation of traffic rules, or the choice of maneuvers, among many others activities (VERES *et al.*, 2011).

In practice, the decision-making problem is one of the most complex and critical challenges for AVs deployment, being related to its level of autonomy (see Tab. 1) (FURDA; VLACIC, 2010). Thus, the greater the autonomy more decisions will have to be made, because of this, several algorithms can be used, each one being indicated for a certain decision domain such as Artificial Neural Networks, Probabilistic Models, Fuzzy Models, Rule-Based Models, etc. (VERES *et al.*, 2011; LIU *et al.*, 2017).

2.1.4 Control

Once the actions have been defined by the planners and decision-making algorithms, the control module is responsible for executing them. For this, another collection of algorithms called controllers are used, now responsible for converting decisions into acceleration, braking, and steering commands, allowing the vehicle to follow the trajectories and perform maneuvers (PENDLETON *et al.*, 2017).

According to Liu *et al.* (2017) one way to accomplish this task is to divide the movement into lateral and longitudinal controllers, which calculates actions considering the speed and movement constrains of the vehicle. Briefly, lateral control uses the Ackermann geometry equations to calculate the steering angle that will take the vehicle from its current position to the desired one calculated by the path planner. In turn, the longitudinal control uses the same reference path and calculates the acceleration and braking required to achieve a desired speed (JO *et al.*, 2015).

2.1.5 Mapping

The map is one of the most important representation of the external environment, which allows the Autonomous Vehicles navigate. It can have information about the road network and intersections; position of static obstacles; traffic signs; landmarks; and, so on (BRUMMELEN *et al.*, 2018). This information can be used to assist the vehicles in the algorithms of localization,

navigation, decision-making and control, that is, in almost all tasks needed for autonomous operation (ZANG; CHEN; TRAJCEVSKI, 2018).

As stated by Brummelen *et al.* (2018), most of the autonomous vehicles that are in the test phase, use maps constructed beforehand, that is before the vehicle begins autonomously navigate, it travels along the route collecting sensory information used for the construction of the map, and using this map the vehicle will be able to navigate. The authors state that this method is quite effective, however, constant changes in the environment, as well as the amount of information stored can drastically reduce efficiency, scalability, and maintainability in its use.

Because of this, Zang, Chen and Trajcevski (2018) and Liu *et al.* (2017) point to the existence of High Definition Maps (HDM) as a probable solution for the use of maps that require a high dimensionality of information. Like other maps, HDMs are constructed beforehand, but are constantly updated and stored as a Cloud service (WHAIDUZZAMAN *et al.*, 2014), and can be made available to vehicles on demand through the internet or VANETs (Vehicular Adhoc NETWORKs) (BRUMMELEN *et al.*, 2018).

2.1.6 Human-Vehicle Interface

The interaction between the autonomous system and users is a very important issue for the successful deployment of the technology (KOOPMAN; WAGNER, 2017). According to Koo *et al.* (2015) the way AVs will interact with their customers is still a challenge for developers and designers since the public is even more heterogeneous than conventional vehicles, once these vehicles can be used by users without driver license such as children and teenagers, as well as people with disabilities (LITMAN, 2017).

As stated in Koopman and Wagner (2017), this interaction plays an important role because it directly impacts on the confidence that users will have in the operation of autonomous vehicles. Thus, Koo *et al.* (2015) elucidates two essentials information that will need to be clear to the passengers, the first is what actions the vehicle is taking, and the second is the reason to do that. According to the author, only then will a bond of trust be reached.

2.1.7 System Monitoring

An autonomous system is composed by many software and hardware components, which are eventually subject to failures. Thus, it is essential to identify and treat them. According to Gusikhin, Filev and Rychtycky (2008) the system monitoring module has the responsibility of analyze the whole data flow of an autonomous system and assess whether any unit does not behave as it should, thus preventing an error from propagating and impairing the performance of the autonomous vehicle.

The authors highlighted that basically there are two approaches to analyze the information and identify faults, the first one uses mathematical models of the system and calculates the

deviation from the observed values and the expected ones. The second approach learn from data the pattern that models the behavior of the fault, such as Artificial Neural Networks, Statistics Models, Support Vector Machine, among others ([GUSIKHIN; FILEV; RYCHTYCKYJ, 2008](#)).

2.1.8 Support Platform

As seen so far, the autonomous system is composed of many components that need to be executed in parallel and in real time. Thus, the infrastructure that supports this execution is: Operating System, which executes the components of software; The Robotic Middleware, responsible for coordinating the interaction and order of execution of the components, in other words, it manages all processes of the autonomous system, in addition to providing the means by which they communicate; and, Hardware Platform, which is composed by all the physical units of the standalone vehicle, such as sensors and actuators.

2.2 Connected and Autonomous Vehicles

All the aforementioned components allow the autonomous vehicles to perform all their tasks, improving road safety, passenger comfort, traffic logistics, among others. However, they are not alone in urban traffic, so they share the streets with other cars, pedestrians, motorcyclists, etc.

Thus, according to [Philip and Saravanaguru \(2018\)](#), if vehicles operate in isolation, they will not be able to contribute fully to the improvement of cities and their citizens. Because of this, a requirement considered very important for AVs and their implementation is the ability to interact with other agents, exchange information or receive guidance ([YANG; OZBAY; BAN, 2017](#)). When the vehicles communicate with each other or other traffic agents they become connected, also called Connected Vehicles (CV) and Connected Autonomous Vehicles (CAV).

As stated in [Guanetti, Kim and Borrelli \(2018\)](#), the connected autonomous vehicles are able to operate autonomously and communicate with other vehicles and road users, as well as the infrastructure of the city. When the vehicle communicates with other vehicle, the communication is classified as Vehicle-to-Vehicle (V2V) , other kinds of communications are: Vehicle-to-Infrastructure (V2I) ; Vehicle-to-Pedestrian (V2P); or, Vehicle-to-Anything (V2X), which is the generalization for the communication with any other agent ([PHILIP; SARAVANAGURU, 2018](#)).

Through this interaction, the vehicles will be able to collaborate with each other, providing information, as well as allowing their negotiation, for example at the intersection, in order to obtain better results, considering all the urban transport infrastructure.

- **Vehicle-to-Vehicle (V2V):** allows vehicles to communicate with each other, making it possible, for example: emergency stop warning; coordination of platoons; negotiation of maneuvers; data exchange; etc.

- **Vehicle-to-Infrastructure (V2I):** allows vehicles to exchange information with the city's infrastructure, thus, for example, the vehicles can receive data on all traffic conditions and plan better routes, as well as coordinate their actions with regard to signaling, among other actions. In addition, information on accidents or road conditions can be sent to the city management system, further benefiting urban transport management.

According to [Shladover \(2018\)](#), research on autonomous vehicles and connected vehicles has advanced for a long time separately, however, the tendency is for them to be developed together, given their complementary characteristics. Thus, the author clarifies that in addition to the challenges inherent to AV and the complexity of its algorithms, the connected autonomous vehicles will have other issues that need to be solved, such as the security of the communication infrastructure. Still, connectivity will bring many benefits that have helped in the autonomous operation, as well as in organizing all the traffic.

2.3 Final Considerations

This chapter presents some concepts and definitions about autonomous vehicles and connected vehicles, emphasizing the description of the components of an autonomous system. Thus, it is possible to notice that the autonomous vehicles are able to improve the quality of life and safety of the people, because they have several sensors and algorithms that coordinate their actions, leading them to a better analysis of traffic situations, consequently, allows the choice of better actions to avoid accidents and improve the flow of urban transportation.

In addition, the connectivity and ability to communicate with other traffic agents, such as other vehicles, pedestrian, and the city's infrastructure, should further enhance the benefits provided by AVs, enabling them to have a global sense of traffic and coordinate their actions for more safety and convenience.

AUTONOMOUS VEHICLES ARCHITECTURES

This chapter discusses some important aspects of robotic architectures, as well as the analysis of some architectural models of Autonomous Vehicles available in the literature. These related works were used to guide the new architecture model for *CaRINA II* presented in Chapter 5.

3.1 Robotic Architectures

As seen so far, an autonomous system is composed of several components (see Chapter 2), so it is necessary to develop an architectural design that allows an abstract view of its operation and organization, allowing decisions regarding project execution to be better evaluated. In turn, the architectural design must follow patterns and styles that establish a common language between architects and engineers, determining rules that guide the development and understanding of the architecture. Each pattern or style has its own characteristics and is often complementary, so it is very common to use more than one to describe the architecture.

For the point of view of Software Engineering, a software architecture presents the structural descriptions of a system in different perspectives, for example, how its components communicate, memory allocation and processing, how its elements are distributed in the network, or the order of execution of functions for reaching the desired purpose, etc (ZHU, 2005). Despite this variety of representations, some elements are common to all, defined by Gimenes and Huzita (2005) and Bass, Clements and Kazman (2003) as the *components*, that are the software units that performs specific and well-defined functions, having public interfaces through which it establishes relationships with other components of the architecture, and the *connections* that define the relationships between components, which can establish order of priority, precedence, constraints, and also the structure through which the components exchange information and their

meanings.

The layered architecture style establishes a topological and hierarchical organization of the software components that are grouped according to some criterion, each group or layer has a particular level of abstraction and offering services to its neighbor (FIELDING; TAYLOR, 2000). A very important aspect for the implementation of this architectural model is that the components in each layer must be independent of the components of others, so they communicate only through a well-defined interface. The consequence of this constraint is that the components or layers can be replaced without affecting the system, provided that the interface is preserved, in this way, the modularity, scalability, and reuse of components, end up being extended, something quite desirable for the development of software (ZHU, 2005).

According to Kortenkamp and Simmons (2008) the design of robotic architectures, although based on software engineering, has differences due to the particularities of the robotic system, among them, the need to interact with dynamic environments and deal with uncertainties. In order to do so, the authors highlight some essential requirements, such as real-time control, control of actuators and sensors, support of concurrence between components, dealing with unexpected situations, handling uncertainties, being able to carry out plans at different levels of abstraction, etc.

Because of the inherent complexity of the area, some good practices for software development are adopted, one of the main being the decomposition of the system into modules, which reduces the difficulty in implementing, integrating and testing each functionality, as well as improving maintenance and scalability (KORTENKAMP; SIMMONS, 2008; GIMENES; HUZITA, 2005). Moreover, according to Chavan, Murugan and Chavan (2015) and Kortenkamp and Simmons (2008), the division of modules hierarchically into layers also reduces their complexity by organizing the components at different levels of abstraction.

The robotic architectures then represent the structure in which the robotic components are organized. A well-known architectural model was called 3-Layer Architecture, which was organized in the reactive control component, reactive planning execution, and deliberative planning, the latter being responsible for the coordination of the other two and for time-consuming computations tasks (GAT; BONNASSO, 1998). This style was applied in many robots, however, with the increasing of components, functionalities and complexity of the systems, other approaches using a variable number of layers were adopted, such as the RCS (Real-Time Control System) developed by NASA/NBS (KORTENKAMP; SIMMONS, 2008).

For autonomous systems, Behere (2012) divides architectural models into two main concepts, the cognitive and the control. The cognitive architectures has as main objective to model the perception, the planning and the decision making. In turn, the second paradigm prioritizes real-time control, sensor fusion, and security. In practice, the autonomous vehicle architectures have both included paradigms, thus, representing with greater clarity the complexity of an autonomous robotic system.

Another point highlighted by several authors, among them [Taş et al. \(2016\)](#) and [Ulbrich et al. \(2017\)](#), is that the autonomous vehicle architectures need to also be concerned with the reliability and robustness of the system, requiring that approaches that deal with hardware and software failures are included in the architectural model, and therefore implemented in the system (see chapter 4).

Next, some models of autonomous vehicle architectures found in the literature are analyzed, focusing mainly on the organization of the components in the architectural design, such as the layered divisions, navigation pipeline, and safety aspects.

3.2 Autonomous Vehicle Architectures

3.2.1 Tongji

The *Tongji* vehicle (Fig. 5) was developed by Tongji University of Changshu City - China ([ZONG et al., 2018](#)). It is one of the participants of the IVFC (Intelligent Vehicle Future Challenge) competition organized by National Natural Science Foundation of China (NSFC), as a program to promote the study of Artificial Intelligence and the development of technologies related to Connected Autonomous Vehicles ([WANG, 2017](#)).

Figure 5 – *Tongji* Autonomous Vehicle

Source: [Zong et al. \(2018\)](#).

According to [Zong et al. \(2018\)](#), the vehicle has sensors typically found in other projects such as cameras, radar, ultrasonic sensor, LiDAR, GPS and IMU. The Figure 6 presents its architectural design. According to the authors, the architecture has two hierarchical representations in layers, one horizontal and one vertical. The vertical organization is divided into three layers, the *algorithm* layer, the *data transmission* layer, and the *operating system* layer. The former is then subdivided horizontally into four other layers, which are: *perception*; *decision making and planning*; *control*; and, *chassis*.

Figure 6 – Tongji Software Architecture

Source: Zong *et al.* (2018).

In the *Perception* layer the sensing modules are grouped, which are responsible for measure data from sensors. Besides that, the layer also has the algorithms that obtain information from the environment (detection of traffic signals, pedestrians and other vehicles, road lanes, navigable area, and obstacles detection), the global and local vehicle localization system, sensor fusion techniques, and some mapping components. In the *Decision-Making and Planning*, there are user interaction modules, traffic rules evaluation and route and path planning. In turn, *Control* gets the information from the planning system and produces steering, braking, acceleration and gear shifting commands, which in turn are executed by the chassis components.

Tongji has a separate communication infrastructure for architectural components. These elements are organized in the *Data Transmission Layer*, which, among other functionalities, is responsible for keeping a logger of the entire flow of information in the system. Finally, the *Operating System Layer* is responsible for managing the processes of the autonomous system. Another very important feature is called *Process Management*, which monitors all system components and is capable of detecting failures as well as restarting the system as a failure recovery protocol.

3.2.2 A1

The A1 vehicle (Fig. 7) was the winner of the *Autonomous Vehicle Challenge* (AVC) competition held in 2012 in South Korea. The competition was promoted by the Hyundai Motor Group, to boost the development and research on autonomous vehicles in the country, and had challenges such as respecting traffic laws and signs, dealing with the presence of other vehicles and pedestrians on the roads, among others (JO *et al.*, 2015).

Figure 7 – A1 Autonomous Vehicle

Source: JO *et al.* (2015).

As stated in JO *et al.* (2015), the vehicle has eight laser scanners, a stereo-camera, two GPS receivers (one Real-Time-Kinematic GPS, and other Differential GPS), and an IMU (Inertial Measurement Unit). In addition, the A1 software components are distributed on two industrial embedded computers. The software architecture is then divided into five main functions, the location, perception, planning, control and management of the system. These components are then organized in four layers, shown in Figure 8, named: *Sensor Interface*; *Autonomous Driving Algorithm*; *Actuator Interface*; and, *Development Interface*.

The *Sensor Interface* layer gathers the drivers responsible to measure data from the environment using the sensors. On the other hand, the *Autonomous Driving Algorithm* layer is responsible for the perception, planning and control functions. For localization, A1 uses an Interacting Multiple Model (IMM) filter-based information fusion, which is able to change in run-time the kinematics and dynamic models of the vehicle under different driving conditions (JO; CHU; SUNWOO, 2012). The vehicle also uses a GPS-bias correction to improve the accuracy of the localization (JO; CHU; SUNWOO, 2013).

For perception, the vehicle uses Support Vector Machines for detection and classification. Beyond that, for the planning, the system divided it into three stages, the global routing, behavior reasoning, and local motion planning. The former finds the route to go to a destination using a digital map, while the second is responsible to determine the overall behavior of the vehicle using the perception information and the path planned. This task is performed by a rule-based approach which uses a Finite-State Machines (FSMs). The local motion planning then generates the motion based on the route and decisions, it is also responsible for obstacles avoidance. Finally, for control, lateral and longitudinal controllers are used. The lateral controller assumes

Figure 8 – A1 Software Architecture

Source: Jo *et al.* (2015).

an Ackermann geometry, and the longitudinal controller is a Proportional-Integral-Derivative (PID) controller.

The commands generated by the longitudinal and lateral controllers are then executed by the components of the *Actuator Interface* layer. The last layer, named *Development Interface*, is in charge of system's security features such as the emergency button that passes the command of the vehicle to the driver by means of a physical button and the monitoring of the components of software and hardware. According to the authors, each software unit periodically sends a signal to the diagnostic module, which then is able to determine if all the components are active and operating. When a faulty component is detected, the system then switches its operating mode to paused, allowing the driver to take control.

3.2.3 BerthaOne

BerthaOne (Fig. 9) is an autonomous vehicle developed by the same team of the vehicle AnnieWAY, participant in the DARPA and GCD (Grand Cooperative Driving Challenge, 2011) competitions, as a result of a collaborative project between the Karlsruhe Institute of Technology and the FZI Research Center for Information Technology, in Germany. The vehicle was also present at the event that became known as the Bertha Benz Memorial Route (Taş *et al.*, 2018).

According to Taş *et al.* (2018), in terms of sensors, the vehicle has cameras, Radar, LiDAR, GPS, IMU, and is able to use V2V (Vehicle-to-Vehicle) and V2I (Vehicle-to-Infrastructure)

Figure 9 – BerthaOne Autonomous Vehicle

Source: Taş *et al.* (2018).

communications to provide more information about the environment and the situation of the vehicle. All sensors and software components are managed on two computers, one of which is specific to real-time tasks, and the other performs functions that require greater computational power. The framework ROS (Robot Operating System) is used as core of the communication between the components of the architecture.

The BerthaOne software architecture, shown in Figure 10, is organized in layers named: *Sensing*; *Perception*; *Planning*; *Control*; *Human-Machine Interface*; and, *Communication*. *Sensing* is then responsible for measure data using the sensors, while *Perception* uses these data to generate useful information of the environment. An important point in this architecture is to use the *Communication* as source of information for the perception algorithms, by means of communication with other traffic participants. For example, in object fusion, the position information of other vehicles that were obtained via V2V communication is combined with the information generated by the Radar, LiDAR and stereo camera.

Figure 10 – BerthaOne Software Architecture

Source: Taş *et al.* (2018).

The localization is made by three different sources, the GNSS/INS (Global Navigation

Satellite System/Inertial Navigation System), in-vehicle odometry, and vision-only localization. The data can be fused using an Unscented Kalman Filter (UKF), which is also able to reject outliers using a Mahalanobis distance to detect it. The localization is then sent to the object fusion, map matching, and V2V interfaces. The object fusion also receives the obstacles detected by the perception system and the position of other vehicles received by the communication V2V interface.

All this information is then sent to the behavior planner, which uses States Machines to define the behavior of the vehicle, based on a rule-based. The states transition can be triggered by the perception or communication components. After the decision is taken, the *Motion Planner* calculates the optimal motion. The motion is finally executed by the controllers in the *Control* layer.

3.2.4 IARA

The Intelligent Autonomous Robotic Automobile (IARA) is an autonomous vehicle developed by the Laboratório de Computação de Alto Desempenho (LCAD) in the Universidade Federal do Espírito Santo (UFES), Espírito Santo - Brazil. This vehicle is a Ford Escape Hybrid modified to operate autonomously with the addition of sensors and actuators to control the steering wheel, throttle and brake pedals ([CARDOSO, 2017](#)). In 2017, IARA performed a 74km route autonomously, from Vitória-ES to Guarapari-ES.

Figure 11 – IARA Autonomous Vehicle

Source: [Cardoso \(2017\)](#).

According to [Cardoso \(2017\)](#) the vehicle has as sensors: one LiDAR Velodyne HDL 32-E; one SICK LD-MRS 3D LiDAR; one Trimble Dual Antenna RTK GPS; one Xsens MtI IMU; four Point Gray Bumblebee stereo cameras; and, one ZED 2k stereo camera. The author also mentioned that the software architecture is mainly composed by seven modules: *localizer*; *mapper*; *behavior selector*; *path planner*; *motion planner*; *obstacle avoider*; and *controller*.

For localization, the module estimates the position of the vehicle in the map coordinate frame, using the matching between the LiDAR data and the previously recorded map. Besides that, the *mapper* creates an occupancy grid and an online map also using the LiDAR measurements. The *behavior selector* has the responsibility to locally decide the behavior of the vehicle and the velocity to reach the destination, taking into account the traffic rules and global plan, created by the *path planner* module. Finally, the *motion planner* received the behavior selected and the global plan and calculates the motion needed to accomplish the decision (the desired behavior), then the *controller* performs the motion. Other modules present in the architecture are the health monitor, logger, and traffic lights state detector (CARDOSO, 2017).

3.2.5 CMU

The CMU Autonomous Vehicle (Fig. 12) is a research platform developed at Carnegie Mellon University based on a Cadillac SRX modified to autonomous operation by the addition of several mechanical and electrical components. The vehicle is equipped by six LiDARs, six RADARs, two GPS, one RGB camera, and one thermal camera, a wheel speed sensor, and an IMU (WEI *et al.*, 2013).

Figure 12 – CMU Autonomous Vehicle

Source: Wei *et al.* (2013).

As stated in Wei *et al.* (2013), the autonomous operation is basically implemented by an hierarchical control structure, which uses a Proportional-Integral (PI) for longitudinal control. For the lateral control, the vehicle has two different algorithms available, a vehicle-kinematic-model-based for lateral control, based on the Ackermann geometry, for low-speed operation, and other controller which also uses the vehicle dynamics for high-speed operation. The whole design of the software architecture is shown on Figure 13.

Figure 13 – CMU Software Architecture

Source: [Wei et al. \(2013\)](#).

The *perception* layer of the architecture is responsible to create the world model using the sensor data and also uses fusion data algorithm to fuse the data from the RADARs, cameras and LiDARs, on a common representation, which is more reliable than if was built using just one source of information. The position of the vehicle is estimated using a combination of a global localization from the GPS with RTK correction and a dead reckoning using the IMU, and a local localization which uses the lane marking and road shape features to estimate the lateral and longitudinal distance between the vehicle and the lane center from the map.

The navigation pipeline for CMU is then composed by three main stages, the first is the *Mission Planning*, in which the global route is computed taking into account the information about speed limit of the roads, number of traffic-lights and stop signs, number of lane changes needed during the entire route, and other heuristics. After this phase, the behavior needed by the autonomous system to execute the route is generated by the *Behavior Generation* component, to finally have the motion computed using the controllers in the *Motion Planning*.

The CMU autonomous vehicle has the safety as the main concern of the system platform. For this, according to [Wei et al. \(2013\)](#), the system is equipped with a three-layer safety system, with a *remote stop*, an *auto/manual* switch, and an emergency stop. Another safety method used in the vehicle platform was the redundancy of compute unit hardware, which consists in the replication of the software components in at least two computers. Network failure is also taken into consideration, so the vehicle also has two Gigabit Ethernet switches to allow the computers

units and sensors share data.

3.3 Navigation Architectures

The most basic and one of the most important task for autonomous vehicles is navigating without the need of human intervention, however, during this task the vehicles must accomplish many other tasks as important as just to go from the current position to the target destination, as already mentioned in earlier chapters. Each of these tasks, in turn, require different behaviors and actions of the vehicle, becoming very complex tasks.

According to [Paden *et al.* \(2016\)](#) most of autonomous system uses a hierarchical composition of its software components to break down *navigation* into several layers with different levels of abstraction. Figure 14 shows a representation of the architecture design mentioned by the authors. In this approach, the above component breaks the input down into sub-tasks given as input for the below component, until the steering angle, throttle, and brake commands are generated. Therefore, the first component of the navigation pipeline is the *Route Planning*, which receives the destination and a road network model to find a route from the vehicle current position to the target destination. This route can be selected by using different cost functions, which defines how better one solution is compared to others. Common cost functions are: distance traveled; number of traffic lights; speed limit; traffic flow; safety; and so on. As stated in [Bast *et al.* \(2016\)](#) there are also algorithms which uses multiple cost functions to find the best route. The authors also highlighted that algorithms typically used to find the route are Dijkstra's, A* search, Contraction Hierarchies, among others.

The output of the *Route Planning* is a sequence of vertexes (intersections) and edges (roads) in the road network model that guides the vehicle to its destination. This route is then given as input for the layer below, named *Behavior Layer*, which also receives perception information such as obstacles position, traffic signs, and the current state of the vehicle. According to [Paden *et al.* \(2016\)](#), [Wei *et al.* \(2014\)](#) and [Katrakazas *et al.* \(2015\)](#) the *Behavior Layer* is responsible for make tactical decisions about the actions of the vehicle during navigation, such as interaction with other traffic participants, respect traffic rules, and high-level maneuver choice for the vehicle, such as, lane following, lane change, U-turn, overtaking, emergency-stop, among others. The authors emphasize that some of the algorithms used for the decision process and planning are MDP (Markov Decision Process), POMDP (Partially-Observable Markov Decision Process), and Finite State Machines (used by both A1 and BerthaOne, mentioned in previous sections). A different approach for behavior execution, named Behavior-Tree, is presented in section 4.2.1.

After the behavior has been selected, it is sent to the *Motion Planning* layer, which is able to plan the path to perform the maneuver ([PADEN *et al.*, 2016](#)). According to [Katrakazas *et al.* \(2015\)](#) the planning task consists of provide the vehicle a feasible and collision-free path headed for the destination. This path is then converted to low-level commands, such as throttle,

Figure 14 – Hierarchy of decision-making processes for an Autonomous Vehicle

Source: [Paden et al. \(2016\)](#).

brake and steering. The algorithms responsible for this last task are present in the layer named *Local Feedback Control*, and they are called controllers.

While a hierarchical organization of the navigation components makes the task easier to be accomplished, since it is possible to divide a high-level goal into more specific sub-goals through the top to the bottom of the architecture, it also has its disadvantages. As stated in [Wei et al. \(2014\)](#), this organization imposes several constraints to real-time requirements of the autonomous system, since the major problem of hierarchical structures is that the high-level components do not have sufficient detailed information of the trajectory, and the lower-level components do not have permission to reevaluate the decision received to be performed. Thus, these architectures have an overhead to deal with dynamic traffic situations.

The authors then present a different approach for the navigation pipeline, which has components organized in parallel (Fig. 15). As highlighted by the authors, this structure is based on the functioning of ADAS (Advanced Driver Assistance) system, in which the components are independent with each other. In this way, each of the components has a specific task, and performs it in parallel of the behavior of other component. For example, in the architecture presented by the authors in Figure 15, the *ACC Controller* only controls the throttle and brake to keep the vehicle at the target speed while also keeps it at a desired distance from other vehicles ahead. In its turn, the *Lane Centering* only controls the steering wheel placing the vehicle at the center of the road lane. Both components works without interaction, and the output of each one has no impact in the other. This organization also allows the components works at a higher frequency compared to another approaches.

Figure 15 – Parallel Navigation Architecture

Source: Wei *et al.* (2014).

On the other hand, the problem with this approach come up when the system needs to perform complex tasks, such as maneuvers (e.g. lane change), since each component works asynchronously and in parallel, which makes difficult or even impossible to coordinate execution and control of the vehicle. To overcome this problem and also improves the performance with real-time requirements for autonomous navigation system, the authors indicate that a combination of hierarchical and parallel components structure should be used, which allows a faster replanning in dealing with changes in previous decision.

In this hybrid approach, a hierarchical composition is the main core for organizing the components, since it makes possible the execution of complex tasks. The parallel concept is applied with the components that require more interaction between them, to perform faster planning, replanning, and execution of the maneuvers and action. Figure 16 shows the proposed architecture by Wei *et al.* (2014), which was implemented and tested on the CMU Autonomous Vehicle (see section 3.2.5). At this design, the *Mission Planning* is responsible to compute the route and coordinate the lane-level sub-mission, which is the requirement for each segment of the route, e.g. the lane the vehicle should be at the end of the road, the next road it has to

meet to achieve the desired destination, and the decision about when crosses signalized and not signalized intersections. Finally this component also coordinate each segment of the route, that is, when the vehicle finish each piece of the route, the *Mission Planning* send another segment to the layers bellow, until the vehicle reach the desired destination.

The next layer, named *Reference Planner*, according to the authors, assumes traffic-free roads, taking into account only the road geometry, the vehicle kinematics and dynamics, and comfort parameters for the passengers. The output of this layer is a feasible path and a speed profile to be performed. This planning is executed for each segment of the route at the time, received from the layer above. After this, these outputs are given as input to the *Behavioral Planner* which handle traffic scenarios and obstacles. It uses a Prediction and Cost Function Based (PCB) to simulate a finite sample of candidates parameters to tune the controller in the layer below, and chose the best choice based on the cost function. Finally, these parameters are used for the *Controllers*, and the steering, throttle and brake commands are generated and executed. The advantage of the hierarchical composition for this architecture is that the *Behavior Planner* has now more detailed information of the path, which allows it to generate better samples and perform better simulations to choose good parameters for the controllers.

Figure 16 – Hybrid Navigation Architecture

Source: [Wei et al. \(2014\)](#).

3.4 Final Considerations

This chapter presents concepts and characteristics of robotic architectures, as well as a brief historical review of the evolution of architectural models. After that, the architectures of five autonomous vehicles, the *Tongji*, *.AI*, *BerthaOne*, *IARA*, and *CMU*, all participants and winners of great challenges of the area are presented and analyzed.

In the analysis of the architectures, is discussed the structural composition and the adoption of approaches for diagnosis of the autonomous system. The diagnosis was a concern of the *Tongji*, *AI* and *CMU* architectures, which also presented their approaches to post-crash recovery. However, *BerthaOne* does not present its considerations in this regard, even so its architectural design remains very important for analysis, due to the integration of V2V and V2I communication, which tend to increasingly become essential components in autonomous vehicle software architectures. Finally, the chapter presents the component organization of a navigation architecture comparing the hierarchical, parallel and hybrid approaches, making it possible to noticed the importance of a hybrid approach to deal with the dynamic and complex struture of traffic and the autonomous system. These works are used as references for the design of a new architecture for the vehicle *CARINA II* presented in Chapter 5.

TECHNICAL BACKGROUND

This chapter provides the main theoretical concepts applied in this master degree project, and which were used to fulfill some components of the new architecture design for *CaRINA II*. Thus, it will be discussed about the mapping model for Autonomous Vehicles, techniques for behavior execution, approaches for Fault Detection and Diagnosis for an autonomous system, and the robotics framework used in the system development.

4.1 Road Network Description

As already stated in section 2.1.5, a map is an essential component which allows the autonomous vehicle safely and properly perform all its tasks (BRUMMELEN *et al.*, 2018). For this, the structure can store many kinds of information, useful for different components of the autonomous system, such as the road geometry description for path planning, or the topological representation of the roads and intersection, also known as road network, used for route planning. However, conforming to Bender, Ziegler and Stiller (2014) there is no unified standard to meet all the needs of autonomous vehicles, from the description of roads to contextual and semantic information of traffic. Some of the map description standard models already mentioned in the literature are the *OpenStreetMap* (HENTSCHEL; WAGNER, 2010; FLOROS; ZANDER; LEIBE, 2013), the *OpenDRIVE* (BELBACHIR *et al.*, 2013), and the *Lanelets* (ZIEGLER *et al.*, 2014; BENDER; ZIEGLER; STILLER, 2014; POGGENHANS *et al.*, 2018), that are better described in the following subsections.

4.1.1 *OpenStreetMap*

The OpenStreetMap (OSM) project started in 2004 at University College London (UCL). Since then, the project became an open format standard for map description, which offers a collaborative mapping tool, and has a large community around the world for improving the quality of the maps and create new maps (HAKLAY; WEBER, 2008). Because of the works of

contributors, there are many applications for editing existent maps and create new ones, which can be uploaded in the website of the project under the Open Database License (ODbL) version 1.0 licence for data, which means that other users can use, share, and modify the geographic data freely ([OPENSTREETMAP, 2019](#)).

The OSM format is based on XML (Extensible Markup Language) version 1.0, and is mainly composed by three structures (Fig. 17): *node*; *way*; and, *relation* ([OPENSTREETMAP, 2017](#)). The *nodes* are the basic unit of the format, which represents the point in space defined by its geodesic coordinate, an unique id, and may have *tags* related to it, which adds contextual information, e.g. if it represents the position of a traffic sign, or some amenities such as restaurant, school, university, among others.

In turn, the *ways* are lists of *nodes* representing roads, buildings, playground, rivers, railways, so on. For this, there are mainly two kinds of lists, an open or closed. The former is used for example in roads, and the latter occurs when the first and last *node* are equals. A *way* can also have *tags*, e.g. the name of the road or building, or the direction of the road. Finally, the *relations* are groups of elements (*nodes* and/or *ways*) used for defining relationships between them, which can be described using a *tag*.

Figure 17 – OpenStreetMap

Source: Adapted from [OpenStreetMap \(2017\)](#).

4.1.2 OpenDRIVE

The OpenDRIVE is an open format to describe road networks, using XML version 1.0, which is able to represent the road geometry as well as the context information of roads that may influence the behavior of vehicles driving in it, such as traffic signs, traffic lights, and the type of roads and lanes (e.g. highway and sidewalk) ([DUPUIS; STROBL; GREZLIKOWSKI, 2010](#)). The project started in 2006 by VIRES Simulationstechnologie GmbH in partnership with Daimler Driving Simulator, with the purpose of defining a standard of description to be used in simulators and by other softwares ([DUPUIS; HEKELE; BIEHN, 2019](#)).

The format description is built on a hierarchical structure (Fig. 18), in which the high-level elements describe more abstract information, and the low-level more detailed information. Therefore, the standard has mainly four elements: *header*; *road*; *junction*; and, *controller*. The

header, is the first element of the description and holds metadata of the map, such as the name or a geographic reference for transformations between the Cartesian and Geodesic coordinate systems.

The *road* element consists of its geometry description and other properties (e.g. elevation profile, lanes and traffic signs). The properties such as traffic signs are placed with respect to the distance to the initial point of the lane, using a local coordinate system. Besides that, *roads* can be connected with each other directly or by intersections using the *junction* component, which is used to resolve ambiguities between the connections. Finally, a *controller* is used in case of signalized junctions or other road elements which imposes some sort of control in the vehicle.

Figure 18 – OpenDRIVE

Source: Adapted from Dupuis and Grezlikowski (2006).

4.1.3 Lanelets

The Lanelets format for road network description is built on top of the OpenStreetMap standard, using lane corridors as abstraction for the drivable area for navigation. Each lane corridors is composed by a left and a right bounds, which is a list of point in space (BENDER; ZIEGLER; STILLER, 2014). This model was used by the BertaOne Autonomous Vehicle (see section 3.2.3) to perform the Bertha Benz Memorial Route (ZIEGLER *et al.*, 2014).

After the event, a new release of the model was made public, in which the description of the road network was divided into three layers: *physical layer*; *relational layer*; and, *topological layer* (POGGENHANS *et al.*, 2018). The *physical layer* is composed by *points* and *lines strings*, which consist in the primitives elements to create other entities of the map. In turn, the *relational layer* is composed by *lanelets*, *areas*, and *regulatory elements*. In this case, the *lanelets* are the drivable areas, while the *areas* are non-structured environments that may be navigable or not, and the *regulatory elements* defines traffic rules. Finally, the *topological layer* represents the neighbourhood relationship between the elements of the *relational layer*.

4.2 Behavior Planning and Execution

According to [Paden et al. \(2016\)](#) the behavior layer is responsible to decide the appropriate action to be performed by the vehicle, leading him to complete his overall goal. This action is called behaviors or maneuvers, which is defined by [Katrakazas et al. \(2015\)](#) as a high-level motion of the vehicle, such as 'lane following', 'change-lane', 'turning', 'overtaking', 'stop at intersection', 'emergency stop', among others.

An important aspect about maneuvering is that they must respect traffic laws, and especially the safety of the vehicle, passengers, and other traffic agents ([KATRAKAZAS et al., 2015](#)). In addition, to choose the correct maneuver, the decision must take into account the road geometry, the limitations of the vehicle kinematics and dynamics, and also the actions of the other traffic agents, which leads to the need of predict the behavior of the surroundings agents, and deal with uncertainty in the information ([LEFÈVRE; VASQUEZ; LAUGIER, 2014](#)).

As stated in [Paden et al. \(2016\)](#), and [Katrakazas et al. \(2015\)](#), some of the common techniques used for this task are the Markov Decision Process (MDP), Partially Observable Markov Decision Processes (POMDP), Decision-Tree, Finite State Machines (FSM), and Hierarchical Finite State Machines (HFSM). The last two techniques were widely used by the teams participating on the *DARPA's Grand Challenge* and *DARPA's Urban Challenge*, such as the *TerraMax* on the first challenge, and for the second competition the vehicles *BOSS*, *Junior*, *Odin*, *AnnieWAY*, and others ([BUEHLER; IAGNEMMA; SINGH, 2007](#); [BUEHLER; IAGNEMMA; SINGH, 2009](#)). In chapter 3, some of the vehicles presented also use state machines, for example, the *AI* and *BerthaOne*.

The popularity and broad applicability of state machines for the behavior of Autonomous Vehicles is explained by [Paden et al. \(2016\)](#) as being ease of modeling behaviors into states, in which the transition between states can be determined by the perception information and the expected behavior for the specific context and scenarios, and also the fact that the behaviors, or maneuvers, available belong to a finite set of possibilities, which makes the finite state machines a natural choice.

On the other hand, the state machines also have its disadvantages, mainly concern with the scalability, modularity, and reusability for the development of large and complex systems. According to [Colledanchise \(2017\)](#), when there is the need to remove one of the states of the model, every transition to it must be tracked back and also removed, which is a very time-consuming task and demonstrates the lack of scalability, modularity, and maintainability of the technique. Another issues about FSM highlighted by [Colledanchise and Ögren \(2017\)](#) and [Olsson \(2016\)](#) is that FSM is so called *one-way control transfer*, which means that when occurs the transition between states, the flow of the program jumps to another part of the code, without saving the return address on the call stack of the system, such as the jumps instructions for programming languages (e.g. GOTO).

The authors then highlighted that techniques classified as *two-way control transfer* are better choices when the modularity, scalability, and reusability are important for system development. In this kind of approach, when the system jumps to another part of the code, it saves the address of its origin, returning to it in the end of the execution, such as in functions call in programming language.

Behavior-Tree (BT) is a well known approach to model behavior and action applied in the computer games industry since 2005, which is classified as a *two-way control transfer*, since it uses a tree structure for organizing its components and also has logic components, which allows model better control flows than FSM (OLSSON, 2016; COLLEDANCHISE, 2017). This technique is commonly used to create artificial intelligence for Non-Player-Character (NPC), and was used in games such as 'Halo 2', 'Halo 3 & ODS', 'GTA: Chinatown Wars', 'SWAT 4: Bioshock', among others (MARCOTTE; HAMILTON, 2017; CHAMPANDARD; DAWE; HERNANDEZ-CERPA, 2019).

For robotics, according to Colledanchise *et al.* (2015) the Behavior-Trees are a novel alternative to model the reactive behavior of robots. Olsson (2016) presented in his work a comparison between FSM and BT for navigation of an autonomous vehicle, which performs overtaking of other vehicles in front of it. Figure 19a shows the model developed by the author using FSM, and Figure 19b a model using BT to perform the same task. The transitions between states in the FSM are governed by the perception information and the mission progress state, in its turn, the transition between the action in the BT are controlled by especial components, called *control flow nodes*, which will be explained in the next section. By analyzing the two models, is also possible to noticed that the behavior tree has a more readable way to express behavior than states machines, which makes it easier to be used.

Figure 19 – Comparison of Finite-State-Machine and Behavior-Tree

Source: Olsson (2016).

This technique was already used in other works with robots and autonomous vehicles,

such as in the project *iQmatic*, a mining autonomous truck developed by the collaborative project between KTH Royal Institute of Technology, Linköping University, Autoliv, Saab, Combitech and Scania CV AB (COLLEDANCHISE, 2017; SCANIA, 2016). In Siqueira, Plentz and Pieri (2016) BTs were used for semantic navigation in mobile robots, using the context data to adapt the behavior of the robot for patrolling task. In turn, Milger and Gillgren (2015) used a BT for navigation of a simulated Autonomous Vehicles, which also deals with traffic rules, where the decisions regarding traffic rules are encoded into the tree structure. A more recent application of BT was in the CARLA simulator, in which the technique was applied for create the intelligence of NPC vehicles, which interact with the agent been developed by the users, and for creating specifics traffic scenarios (e.g. pedestrian crossing the road suddenly), allowing a more realist traffic simulation (CARLA, 2018). The next section describe with more details how a Behavior-Tree works, and the meaning of its components.

4.2.1 Behavior Tree

Behavior-Tree is a directed rooted tree composed by three kind of nodes (Fig. 20): *root*; *control flow*; and, *execution* nodes. Each tree has only one single *root* node, which is an especial kind of node with a single child and is responsible to start the execution of the tree, sending signals, called *tick*, to its child. As this node starts the signal propagation for the remaining nodes of the tree, it dictated the frequency in which the tree will work (COLLEDANCHISE; ÖGREN, 2017). As stated in Colledanchise *et al.* (2015), a generic node can only execute its task when receives a *tick*, then its returns to its parent the status *running* if the task is not finished yet, or *success* when it has satisfactorily completed the task, or *failure* otherwise.

Figure 20 – Behavior Tree Elements

Source: Elaborated by the author.

The *control flow* nodes are responsible to coordinate the logic and sequence of task executions of the tree. They are non-leaf nodes, and can be classified in *selector*, *sequence*, *parallel*, and *decorator*, each one with a different meaning and purpose application (COLLEDANCHISE, 2017), which are detailed below. These nodes are one of the main difference between BT and FSM, because using then is possible to explicit construct logic sequence of action, which is done by the FSM using rules for each transition between states, in a not very intuitive way as the use of logical components.

- **Selector:** also known as *fallback*, when receives a *tick* send it to its children, one-by-one, from the most-left to the most-right, until one of them returns *running* or *success*, then returns this status to its parent. *Failure* occurs only when all its children return this status. The purpose of this kind of node is to carry out tasks using more than one approach in expectation of one succeeds (COLLEDANCHISE *et al.*, 2015).

This node can also be used for selecting the appropriate behavior to be executed given the context, for example, in Figure 19b there are three behaviors modeled, the 'obstacle avoidance', 'overtaking maneuver', and 'cruise control'. In such a way, Olsson (2016) uses a combination of *selector*, *sequence*, and *condition* nodes to model the switching between these behaviors, using the *condition* nodes to determine the current scenario.

- **Sequence:** sends a *tick* to its children, from the most-left to the most-right, until one of them returns *running* or *failure*, then returns this status to its parent. *Success* occurs only when all its children return this status. According to (COLLEDANCHISE *et al.*, 2015), this kind of node can be used to execute behavior composed for many tasks, in which each task must be performed in sequence, in the designed order, and all must succeed.

Olsson (2016) uses this node to compose specific behaviors for an autonomous vehicle (Fig. 19b), in which each behavior should be performed according to the appropriate environment context. For this, the authors use a *condition* node as the most-left child for the *sequence* node, allowing it to determine if the following components should be executed.

- **Parallel:** as soon as receives a *tick* from its parent, sends it to all its children at the same time. The node returns *success* only if a given number of its children returned *success*, otherwise it returns *failure* if the amount of *running* and *success* received is less than the given number, and *running* on the contrary. This node can be used to perform simultaneously tasks (COLLEDANCHISE *et al.*, 2015), such as demonstrated in the model described in Figure 36 for handling different traffic signs.
- **Decorator:** has only one child, and provides modification to it according to a specific policy. For example, the policy called 'Inverter' reverse the outcome received from *success* to *failure*, and vice versa (COLLEDANCHISE; MARZINOTTO; ÖGREN, 2014). Other examples of policies are: 'Running is Failure', in which if receives a status *running* changes it to *failure*; 'Timeout', that applies a timeout for its child, stopping the execution of it when the desired time is reached, returning *failure* in such case.

The *execution* nodes are then the leaves of the tree and are responsible to properly execute tasks. There are two kinds of nodes, the *condition* and the *action*, described below. The advantage of using tasks (BT) over the states (FSM), is that a task can be easily used for many different trees, on the other hand, a state is encoded into a single state machine, and taking advantage of it in another state machine is a rather labor-intensive task, since the output transitions of that

state may not be equivalent to the new machine. Another important aspect of the *execution* nodes is that they can share information between them, using the called *blackboard*, which is a data structure shared for all components of the tree, which can write and read data in it.

- **Condition:** is responsible for check if a condition is satisfied, returning *success* if true, or *failure* otherwise.

In Figure 19b, Olsson (2016) uses *condition* nodes to determine in which context the vehicle is, e.g., 'oncoming car?' for verifies the need to execute a obstacle avoidance maneuver, or 'car in front?' to perform a 'overtaking' maneuver.

- **Action:** is small tasks that together with all nodes of the tree compose a behavior. For example, the 'overtaking maneuver' in Figure 19b is decomposed into 'turn out', 'pass car', and 'turn in', all executed in sequence. The node return *success* if the task is accomplished, *running* if is performing it yet, or *failure* if the task could not be done.

4.3 Diagnostic System

4.3.1 Fault Detection and Diagnosis

As autonomous systems also need to deal with faults, according to Lanigan *et al.* (2011) it is possible to classify them as critical systems, and therefore, an essential functionality is to be able to continue their operation even in the presence of failures, making use of techniques to detect, identify and recover. According to the authors, such methods are common in automotive systems, however, most of them are applied when the vehicle is stationary, so they are called offline, and the main objective is to identify components that need to be replaced or auxiliary maintenance. However, due to the autonomous system characteristics on-line techniques, which run during vehicle operation, are essential to ensure safety (LANIGAN *et al.*, 2011; CRESTANI; GODARY-DEJEAN; LAPIERRE, 2015).

According to Gao, Cecati and Ding (2015b), a fault can be understood as an uncharacteristic or unacceptable behavior of a specific component of the system. Already symptoms are events or numerical values representing these occurrences, so they are used to detect and isolate them (STANLEY *et al.*, 2013). As stated in Cai, Liu and Xie (2017), there are three types of faults, which are: Permanent Faults (PF); Transient Faults (TF); and, Intermittent Faults (IF). Figure 21 exemplifies their occurrences in relation to the execution time of the system.

- **Permanent Fault (PF):** As Figure 21a shows, permanent failures are characterized by persisting after their first occurrence, that is, once they happen for the first time, they remain present until the component is replaced. For example, when there is an electrical fault in the vehicle and a sensor stops working permanently.

Figure 21 – Fault Types

Source: Cai, Liu and Xie (2017).

- **Transient Fault (TF)**: The transient faults (Fig. 21b), in turn, occurs randomly only once and may or may not cause damage to the system. Like when the vehicle passes through a region that has a strong electromagnetic field and this causes a disorientation in its IMU, or the autonomous system is unexpectedly restarting.
- **Intermittent Faults (IF)**: Unlike transient faults, intermittent faults (Fig. 21c) are defined by their repetitive behavior, in other words, they often appear and disappear repetitively.

According to Cai, Liu and Xie (2017), TF and IF failures are the most difficult to detect and treat due to their unpredictable behavior, which makes it difficult to establish a recognition pattern. Thus, Fault Diagnosis (FD) is responsible to detect, isolate and identify faults in the monitored system (LANIGAN *et al.*, 2011). For that, detect or recognize, just means to check if a system has a failure in a given time. In their turn, isolate and identify, are concerned with determining the location, type and characteristics of the occurrence, respectively (GAO; CECATI; DING, 2015b; STANLEY *et al.*, 2013).

According to Gao, Cecati and Ding (2015b) techniques for detection and diagnosis can be classified among: model-based; signal-based; knowledge-based; and, hybrids. Each category has benefits and disadvantages, as well as unique characteristics, because of this, are generally applied in different contexts, although in very complex systems it is common to observe more than one technique being used, considering the amount of different subsystems and with their own characteristics that need to be monitored (AMIN; IMTIAZ; KHAN, 2018).

4.3.2 Model-based Approaches

As stated in Stanley *et al.* (2013) and Lanigan *et al.* (2011), the techniques in this category are so called because they use system models to identify when a failure has occurred. There are several variations of models, for example: normal or abnormal operation models, when

modeling the desired or undesired behavior of the system; dynamic or static, whether or not they change over time; quantitative or qualitative, when using numerical or non-numerical values; cause/effect models; based on mathematical data (also called First Principles); probabilistic or deterministic; etc.

Stanley *et al.* (2013) highlighted that the main advantages of using model-based fault detection are that they can also be used for diagnosis, are independent of the application, and in many cases, they represent the behavior of the component. The basic idea of model-based approaches is to compare the output data of the monitored component (y) with those predicted by a model ($\hat{y}$), as shown in Figure 22, so it is possible to say whether they are consistent or not (KUESTENMACHER; PLÖGER, 2016). The difference between the actual output and the predicted output is called residue (LOUREIRO *et al.*, 2014). A fault occurs when the residue is not close to zero and can be classified into different types or levels of faults by analyzing its value (JUDALET *et al.*, 2018; GAO; CECATI; DING, 2015b). One of the well-known technique which uses the same approach is called *Observer-Based* (STANLEY *et al.*, 2013).

Figure 22 – Model-based General Schematic

Source: Adapted from Sheriff *et al.* (2017).

Another approach used is called Bond Graph (BG), which consists of a representation of the dynamics of the physical system, represented graphically by links between the components of the system (GAWTHROP; BEVAN, 2007). This technique was used in works such as: Loureiro *et al.* (2014), which used it to detect failures in actuators of an omnidirectional mobile robot, which also had a recovery system that controlled the activation of redundant actuators; Or, in Yu and Wang (2014), which used it to detect different types of failures in the steering system of a mobile robot.

4.3.3 Signal-based Approaches

Unlike the use of models, which map inputs and outputs to identify a system failure, the signal-based approaches use the outputs and system knowledge to identify patterns of errors (DAI; GAO, 2013). In this case, the outputs are generally represented by the component's own signal, so the faults are recognized based on the symptoms identified using signal analysis (GAO; CECATI; DING, 2015b). According to Dai and Gao (2013) some techniques used are common

to the area of signal processing, such as Wavelet Transform, Spectral Estimation, in addition to the method known as Motor Current Signature Analysis (MCSA).

4.3.4 Knowledge-based Approaches

According to [Venkatasubramanian *et al.* \(2003\)](#) and [Gao, Cecati and Ding \(2015a\)](#), the knowledge-based approach is used when the system is too complex and it is not possible to formally define its model or identify patterns in signals. Thus, machine learning techniques are generally used to model these patterns and identify when a failure occurs. According to the authors, for this, it is necessary to have a large amount of historical data available, because of this, usually this category is also known as data-driven.

The authors present several knowledge-based techniques and approaches used for Fault Detection and Diagnosis (FDD). According to [Venkatasubramanian *et al.* \(2003\)](#), these techniques can be classified for the use of qualitative or quantitative data, as shown in Figure 23. Qualitative methods are generally characterized by a rule base, as in Expert Systems or Qualitative Trend Analysis (QTA). In turn, quantitative methods are characterized by using numerical data to identify patterns, using statistical approaches or not. Thus, [Venkatasubramanian *et al.* \(2003\)](#) and [Sheriff *et al.* \(2017\)](#) highlighted some of the well-known quantitative techniques, such as the Artificial Neural Networks (RNA), Principal Components Analysis (PCA), Partial Least Squares (PSL) and the statistical classifiers such as Bayesian approaches.

Figure 23 – Knowledge-based Classification

Source: Adapted from [Venkatasubramanian *et al.* \(2003\)](#).

In [Zhang, You and Jia \(2017\)](#), a Bayesian Network (BN) was used for detection and diagnostics in a set of 18 physical components of a vehicle, which is run at startup time. In the networks design, the components are represented by the nodes, and there are 20 edges that relate them. [Cai, Liu and Xie \(2017\)](#), on the other hand, used a Dynamic Bayesian Network (DBN) to detect failures, considering the degradation of physical components, and Markov chains for the transitions between fault states, classified in: No-Fault; TF; IF; and PF. The model used has two layers that relate the faults with their symptoms. In [Liu and Liu \(2003\)](#) a hybrid expert system

was developed for fault detection. The authors proposed the use of Decision Trees and Decision Making Based on Multiple Attributes with Fuzzy Clusters to model search priorities for faulty devices, increasing the efficiency of the expert system for real time.

4.3.5 Bayesian Networks

According to [Fox \(2018\)](#) and [Russell and Norvig \(2013\)](#), the Bayesian Network (BN) is a powerful Bayesian inference tool, capable of representing uncertain and incomplete knowledge, based on Conditional Probability and Bayes' Theorem. Thus, it is possible to portray the effects of causality among the variables of a system, that is, what are the probable consequences, given that a certain event occurred.

[Russell and Norvig \(2013\)](#) clarify that the technique is composed of a topological representation, using an Acyclic Directed Graph (GDA), and quantitative tables that store the total and conditional probabilities known a priori. According to the authors, the connections in the graph symbolize the influence that a node, called *Parent*, exerts on the other nodes, called *Child*. Figure 24 shows an example of a structure for a Bayesian Network.

Figure 24 – Example of Bayesian Network for Diagnosis

Source: Adapted from [Huang et al. \(2008\)](#).

Among other examples, [Huang et al. \(2008\)](#) presents the model shown in Figure 24 to demonstrate a way to use BN for fault detection in a vehicle. In this scenario, the graph nodes, representing the system variables, were divided into two categories, the symptoms or observations (yellow ellipses) and faulty components (gray ellipses). In turn, the arrows indicate the causal relationship between the variables, as in the direct cause and effect relationship between the *Battery* (faulty component) of the vehicle and the condition *No Lights* (symptom). Also in relation to the model, following the BN definitions, each node is associated with a table, called the Conditional Probability Table (CPT), which stores in form of probabilities the a priori knowledge about the relations, as shown in the *Spark Plugs* variable on Figure 24. The CPT

table is defined by the probabilities of the outcomes of the discrete variable conditioned to the outcomes of its parents.

As stated in [Smith et al. \(2012\)](#), one of the main characteristics of a BN is to satisfy the local Markov property, in which the state of a node is conditionally independent of non-desimpliscendents nodes given the value of its parents. This property simplifies the chain rule for the calculation of the joint probability of a random variable (node), which can be expressed as shown in Equation 4.1.

$$P(X_1 \dots X_n) = \prod_{i=0}^n P(X_i \mid \text{parents}(X_i)) \quad (4.1)$$

where X_i is a random variable (node) of the network, and $\text{parents}(X_i)$ is the set of all its parents.

The approach based on the relationship between faults and symptoms is used by other authors, such as [Yin, Jiang and Li \(2017\)](#) that used BN to diagnose the electrical system of an electric truck. [Cai, Liu and Xie \(2017\)](#) uses a Dynamic Bayesian Networks (DBN) to detect and classify faults in electrical components. According to [Cai, Liu and Xie \(2016\)](#), BN can model situations in which a fault has more than one symptom, or a symptom is common to more than one fault.

Another variation of Bayesian Network that has been applied to fault detection and diagnosis is the Hybrid Bayesian Network (Hybrid-BN), which is composed by discrete and continuous random variables at the same model ([IAMSUMANG; MOSLEH; MODARRES, 2018](#)). According to [Chen and Pollino \(2012\)](#) there are basically two models of Hybrid-BN, the Conditional Gaussian (CG) and the Mixtures of Truncated Exponentials (MTE). The former has restrictions in the parental relationship for discrete variables, not allowing continuous parent for them. The second approach provides a more flexible composition of the network, allowing discrete nodes having continuous parents. In this case, during inferences the continuous parents are discretized into a finite set of intervals to allow them to relate to their discrete children ([MORI; MAHALEC, 2016](#)).

After constructing the model, according to [Russell and Norvig \(2013\)](#) it is possible to make inferences in the network by calculating the posterior probabilities of the occurrence of events. To do so, it is used inference algorithms, which can be classified in exact and approximate inferences. The authors present to the first group, the algorithms of *Inference by Enumeration*, *Variable Elimination*, *Clustering* methods, and *Junction Trees*. The problem of exact inference is the computational complexity of evaluating a very large network. Therefore, stochastic approximation algorithms such as *Likelihood Weighting* and *Monte Carlo Markov Chain* were proposed.

There are basically four types of reasoning, or inferences, that can be performed in the BN, which is classified according to the relations parent/child and evidence/query between

the nodes. Figure 25 shows a graphical representation of these inferences. According to Korb and Nicholson (2010), in the first type (Fig. 25a), named *diagnostic*, the symptoms are given as evidence (known values), and the probability of a cause is updated and queried. In turn, in *predictive* inferences (Fig. 25b), the cause are known and the probability of its effects are queried. In addition, in the *intercausal* reasoning (Fig. 25c), an effect has more than one causes, which are independent of each other. However, if effect and some of its causes are know, the probability of its unknown causes may be modified because of this relationship. The last inference type, called *combined* (Fig. 25d), is the combination of the *diagnostic* and *predictive* inferences.

Figure 25 – Types of reasoning in Bayesian Networks

Source: Adapted from Korb and Nicholson (2010).

Finally, the prior knowledge present in the network can be determined by two means, this information can be provided by a specialist, or be learned from historical data. There are different algorithms to learn the probabilities of the CPT, some of them deal with complete dataset, when all entries are completed, and another ones which deal with missing data, when some of the entries are incomplete. As stated in Darwiche (2009) the Maximum-Likelihood Estimation (MLE) is one the techniques used for estimate the parameters of the BN by maximizing the likelihood, presented in Equation 4.2.

$$\theta^{ml} = \underset{\theta}{\operatorname{argmax}} LL(\theta|D) \quad (4.2)$$

where θ^{ml} denotes the *maximum likelihood estimates* (MLE), θ is the set of all parameters estimated, D is a complete dataset, and LL is the log-likelihood function which is defined by:

$$LL(\theta|D) = \log \prod_{i=1}^N Pr_{\theta}(\mathbf{d}_i) = \sum_{i=1}^N \log Pr_{\theta}(\mathbf{d}_i) \quad (4.3)$$

in which Pr_{θ} is the probability distribution of the estimated parameters θ , and $\mathbf{d}$ is an instance of the dataset.

However, according to Tembo *et al.* (2016) for incomplete dataset most of the time is not possible to compute the likelihood function, thus, the Expectation Maximization (EM)

algorithm divided the task into two steps, the *Expectation-Step* and the *Maximization-Step* (NG; KRISHNAN; MCLACHLAN, 2012). Conforming to Darwiche (2009), the first step complete the dataset using an empirical distribution over the data, and then, in the second step, MLE is used to estimate the parameters. This algorithm is repeated until the convergence condition is met. Thus, EM maximize the called *Expected Log-Likelihood* (ELL), since the estimated parameters is based on the empirical distribution, so, the ELL is defined as follow:

$$\text{ELL}(\theta|D, \theta^k) = \sum_{i=1}^N \sum_{\mathbf{c}_i} [\log \text{Pr}_{\theta}(\mathbf{c}_i, \mathbf{d}_i)] \text{Pr}_{\theta^k}(\mathbf{c}_i, \mathbf{d}_i) \quad (4.4)$$

where $\mathbf{c}_i$ are the missing variables.

4.4 Robot Operating System - ROS

The ROS - Robot Operating System, is a robotic middleware originated from a project started in 2007 by Stanford University and the Willow Garage group. According to Quigley *et al.* (2009), the main motivation for the project was the development of a robotic framework system that addressed the needs of the group and which were not fully met by similar existing tools.

Thus, the authors list the main characteristics of the system: use the peer-to-peer architecture for communication between the software units; allows implementations of algorithms in different programming languages; provides tools for verification, visualization, and simulation of the developed systems; modularize the system components in so-called packages structures; and, be an open-source platform.

Among all the other features, one feature that made the tool well known was the fact that it has a free software license, allowing a community to be created around it, providing support to less experienced users, as well as developing many other features and extensions (JOSEPH, 2015). Another important feature is its versatility, for example, as communication infrastructure between the software units, named nodes, the ROS defaults to Publisher-Subscriber style, but also allows the use of the Client-Server. In addition, the organization of the package system facilitates the implementation of distributed systems. Because of these characteristics, the middleware is widely used in Autonomous Vehicle designs, such as *BerthaOne* (see session 3.2.3), *AutoWare* (KATO *et al.*, 2015), *BWM* (AEERHARD *et al.*, 2015), among others.

The main idea of the Publisher-Subscriber style is that each component can assume two roles, publisher and subscriber, which are, in this order, the producer and the data consumer. The first is responsible for producing some information or generating data, which, in turn, are required and used by components that assume the second role. Thus, when the publisher makes some modification to the shared resources, it is responsible for notifying all components that have expressed an interest in that information (EUGSTER *et al.*, 2003).

According to Tanenbaum and Steen (2007) it is very common to use this pattern in event-based architectures, since the components communicate basically by the propagation of events, also called changes by Buschmann *et al.* (1996). These changes and their notifications occur asynchronously, which according to Eugster *et al.* (2003) can be implemented in two ways:

- **centralized:** when performing an update on a resource, the publisher communicates to a centralizing process, which is responsible for passing on the subscribers on demand.
- **decentralized:** In this approach, changes are notified directly between publisher and subscribers without the need for some intermediary component.

Buschmann *et al.* (1996) identify two models for the style, the first one is called a *push*, and suggests that whenever the producer makes a modification, it will send it completely to consumers, who will have no choice but to receive such data. In the second model, called a *pull*, the producer only warns that an event has occurred, and consumers decide whether to save the data. On the other hand, Eugster *et al.* (2003) decomposes the model into other styles, now analyzing the way consumers manifest their interests in resources:

- **Based on Topics:** published events are identified by nominal topics that are used by subscribers to demonstrate interest in the resource.
- **Based on Content:** In this approach, the subscribers can add constraints related to the content of the information that is sent by publisher, so, for example, a component can request that an event only be propagated to itself if a x value exceed a certain threshold.
- **Based on Types:** in this classification, the added constraint is related to the type of data transmitted in the message.

ROS has a centralized process, named master, that is responsible for indexing each new component started, named node, which can be both publisher and subscriber. The messages between publisher and subscriber are addressed by topics and types, and when a publisher publish a data, the subscriber has the right to decide if it will be saved or not. A single node can be a publisher or subscriber of topics as many as needed. After subscribing to a topic, the message exchange occurs directly between nodes through peer-to-peer protocol (QUIGLEY *et al.*, 2009; JOSEPH, 2015).

Therefore, the communication structure provided by the framework facilitates the development of systems with many components, such as the autonomous systems. It provides an easy-to-use communication interface between distributed components of the system, using both the Publish-Subscriber and Client-Server protocols, all based on topic and types. As an autonomous system architecture usually present many components, such as mentioned in earlier

chapters, and they have different needs for communication, use a framework with these two communication protocols improves the flexibility of the system.

Another important feature of ROS, is that all components, called nodes, use an interface of topics and types to communicate with each other. Moreover, to change a node and keep the architecture working, just mean use the same interface in the new component. This is also a characteristic of layered architecture, and a desired feature for software development, since it is directly related with the flexibility, modularity, maintainability, reusability, and scalability of the software.

Besides that, ROS also allow the scientific community shares its works, since it provides a public and known development tool (JOSEPH, 2015). In addition, another feature that contributes to the adoption of the framework is that by allowing the development in different programming languages, the flexibility and versatility in the prototyping process and implementation of components for the autonomous system ends up being improved, since it is possible to adopt an easier language when testing different approaches, and ultimately to adopt a more efficient language only with the best of approaches in the final deployment.

4.5 Final Considerations

This chapter presents the main theoretical concepts applied in the development of the new design of the software architecture for the *CaRINA II*, with regard to the map standard description, behavior execution, fault detection and diagnosis techniques, and the communication framework (ROS) applied in the development of autonomous system.

For map or road network descriptions, the literature presents the OpenStreetMap, OpenDRIVE, and Lanelets, as some of the standard used. While OSM has a simple structure, which allows different elements of the road and cities to be represented, the disadvantage is that the text format of the resultant map became less human-readable when more complexes structure are included in the map. Another disadvantage of OSM is that to create a map all points should be stored in the file, that is, the standard does not allow an intuitive representation of roads with parameterized curves.

On the other hand, the OpenDRIVE has a more detailed structure with many possibilities to represent roads, intersection, and context environment elements, such as traffic signs and static objects. However, the standard also does not have an intuitive way to represent roads using parameterized curves, but, despite this is more complete than OSM for autonomous driving purpose. Finally, the problem with Lanelets arises because of the use of OSM as background, which may not be the better or perceptive choice for describing all elements of the model, nevertheless, the model also has good representation for autonomous navigation.

For behavior planning and execution different approaches have been used in Autonomous

Vehicles projects, some of them to find the best maneuver or action choice given the current environment context, and other mainly used to conduct the vehicle during the execution of maneuvers. In the latter, Finite-State-Machine is a well known technique used in mobile robots, and used for many of the participants of DARPA's Challenges and current projects of Autonomous Vehicles. However, for large system with many kind of behaviors, FSM has a major disadvantages in terms of scalability, flexibility, reusability, and maintainability. In this context, Behavior-Tree appears as a better choice, since these properties are the main concern for the technique, which is already used in the computer game industry and some projects in mobile robots.

Finally, Fault Detection and Diagnosis is an important aspect to allow Autonomous Vehicle to fully autonomously navigate and achieve the last level of autonomy according to the SAE-J3016 (see Table 1). There are many approaches for this task, but for complex systems, the so-called knowledge-based techniques have good performance since they learn from historical data the fault pattern. One of the techniques used is the Bayesian Networks, a probabilistic graphical model which is able to model the relation cause/effect, similar to the relation symptom/fault. Hybrid approaches are also good choices, since they can perform the task using the best of different approaches together.

CARINA II

This Chapter presents the proposed software architecture design for *CaRINA II*, based on the related works discussed in Chapter 3, the theoretical background presented in Chapter 4, and some requirements that have been raised based on the current scenario of Autonomous Vehicles, and on the software components of the earlier *CaRINA II* architecture.

The *CaRINA* (Intelligent Robotic Car for Autonomous Navigation) project was started in 2010, funded by the National Institute of Science and Technology in Critical Embedded Systems (INCT-SEC), and developed by the Robotics Center of the University of São Paulo, São Carlos (CRob/SC) and the Mobile Robotics Laboratory (LRM). In 2011 the project made public the autonomous vehicle *CARINA II* (Fig. 26a), built from the adaptation of a Fiat Palio Adventure (FERNANDES *et al.*, 2014). In 2015, the group, in partnership with the Scania vehicle assembler company, automated a truck (Fig. 26b), also for autonomous operation (LRM, 2018b).

Figure 26 – CaRINA II and the Smart Truck

(a) CaRINA II

(b) Smart Truck

Source: LRM (2018a).

Several works have already been developed using both vehicles, some of which are listed

in [LRM \(2018a\)](#). Currently, research is conducted mainly in the areas of: trajectory planning; scene perception and understanding; pedestrian detection; detection of traffic signs; detection of obstacles; negotiation between vehicles; control; fault detection on the autonomous system; and trajectory and behavior prediction of other traffic agents.

Because *CARINA II* is also a research platform, its software architecture tends to require constant updates, in view of new work that adds new functionality or enhances existing components. Therefore, the main objective of this work is to update the software architecture described in [Santos \(2015\)](#), to accommodate the new features and requirements of the project. The earlier architecture and the proposed design are presented and discussed below.

5.1 Earlier CaRINA II Architecture

The current platform of *CARINA II* is divided between the components of hardware and software. The former are basically the sensors and actuators used in automation of the vehicle, in addition to the computer running the autonomous system. As sensors, the vehicle has a Velodyne HDL-32E LiDAR, a Point Gray Bumblebee XB3 stereo camera, a Septentrio AsteRx2eH GPS with two-antenna and RTK correction, an IMU Xsens MTi, and a Delphi ESR automotive Radar. For the actuation, it is equipped with a RoboteQ that controls the steering and brake, since the throttle is triggered electronically through a circuit that simulates the signal sent by the vehicle itself.

In turn, the earlier software architecture, described in the work of [Santos \(2015\)](#), uses the layered style divided basically into six layers, being: *Localization*; *Map*; *Vehicle*; *Control*; *Perception*; and, *Diagnostics*. Figure 27 presents its design and components organization.

Figure 27 – Earlier CaRINA II Software Architecture

Source: Adapted from [Santos \(2015\)](#).

- **Localization:**

The location of the vehicle is based essentially on DGPS information with RTK correction. After obtaining them, the data is converted from geodetic coordinates to UTM (Universal Transverse Mercator) and then made available through a topic. In this version of the architecture, the location is used by the control module.

- **Map:**

The map is built offline, after the navigation of the route by a human driver, in which the coordinates in UTM of the position of the vehicle are saved, which are then used to generate *Clothoids* curves. In addition, reference speeds are saved to be used by the control.

- **Control:**

The control obtains from the *Localization* module the current position of the vehicle, and from the *Map* references of the position and speed of the path. Thus, it is divided into lateral and longitudinal controllers. The first one is responsible for the steering, and uses the error between the current position and the desired point, to calculate the desired steering angle. Meanwhile, the second adjusts the vehicle speed for reference, using Proportional and Integral (PI) control.

However, before adjusting the speed or the steering, the module sends its commands to the decision-making algorithm, which verifies if the path being executed is free of obstacles, so in case of obstacles the speed is reduced until the vehicle stops completely, or the obstacle leaves the course.

- **Perception:**

Perception is only responsible for obstacle detection using the Velodyne HDL-32E LiDAR sensor. Thus, it provides information for the decision making to verify if the trajectory of the vehicle is free of obstacles, and thus being authorized its execution.

- **Vehicle:**

This layer is composed by the software components responsible for converting control commands into effective actions. Thus, the layer is composed by the programs for brake, throttle and steering actuation. As the module is also connected directly to the CAN bus of the vehicle, it obtains information about the current state and makes it available to the rest of the software components by means of a ROS message.

- **Diagnostic:**

The diagnostic system monitors the flow of information produced by the autonomous system to check if any risk conditions are present. In this case, when any condition is detected, it issues alerts to the driver. One of the most important elements in the verification is the GPS signal, since navigation is mainly based on this component.

5.2 Proposed CaRINA II Architecture

In the evaluation of the earlier architecture (Fig. 27), one of the major problems encountered was the approach adopted by the mapping, which needs to be loaded by each component that wants to use it, thus the control module obtains a version of the map, while the perception module (obstacle detection) needs to load the map again to use it in identifying obstacles in the trajectory of the vehicle. This approach imposes serious restrictions on the inclusion of new functionalities in the vehicle, reinforcing the need for a new design that is more flexible in that regard. Moreover, the decision-making module is concerned only with obstacles, not taking into account, for example, the traffic signs.

The other modifications made came from the need to integrate in the architecture new functionalities, being the main one of the work carried out in Silva (2018), which presented a new road network description using *Clothoids* and path planning algorithm using it. This work provides different functionalities that can be added in the autonomous system of *CaRINA II*, such as the route planning, and a new approach for behavior planning and also path planning components.

The design of the proposed software architecture is presented in Figure 28. Similar to Zong *et al.* (2018) (see section 3.2.1), two hierarchical visions were adopted, one vertical and one horizontal, which allow the representation of system infrastructure alongside the robotic algorithms. Thus, in the vertical, the design is organized between the *Autonomous System*, with the implementation of the algorithms, and *Operating System*, which illustrates two important requirements for system operation, the use of the robotic middleware ROS and Linux Operating System.

The *Autonomous System* is then horizontally divided into the layers: *Sensing*; *Perception*; *Mapping*; *Decision-Making and Planning*; *Control*; *Vehicle*; and, *System Management*.

5.2.1 Sensing Layer

Similar to Taş *et al.* (2018) and Jo *et al.* (2015), the drivers for reading and to manage the sensors are present in the *Sensing* layer. Having a specific layer for sensor management is an important decision because it improves the flexibility and scalability of the architecture, since the characteristic of layered architectures allows to replace a components of the layer or even the entire layer and keep the architecture working, provided that its communication interface to the other layers is maintained. In this case, if the set of sensors changes, the modification tend to be minimal or even non-existent.

This software architecture includes the sensors: Septentrio AsteRx2eH GPS with two-antenna and RTK correction; Point Gray Bumblebee XB3 stereo camera; and, VelodyneHDL-32E LiDAR. The GPS is used for localization, while the camera for object detection and classification, and the LiDAR for obstacle detection. Other sensory information obtained is provided by the

Figure 28 – CaRINA II Software Architecture

Source: Elaborated by the author.

CAN bus and the RoboteQ feedback, which gives the current state of the vehicle, including the speed, gear, engine speed, steering angle, among others.

5.2.2 Vehicle Layer

In this layer are the software components necessary to control the actuators of the vehicle, executing the commands calculated by the control. The vehicle uses a RoboteQ to control the steering wheel and brake pedal by means of two motors, a DC and other pneumatic, in this order. The steering wheel controller receives the desired steering angle and convert it into the position of the wheel, which has feedback monitored by a potentiometer. In its turn, the brake pedal controller receives the desired percentage to press the pedal, being 0 without pressing and 100 complete pressed, the internal feedback is also made by a potentiometer, managed by the RoboteQ. Finally, the throttle is controlled using an electronic board which converts the percentage of the desired position of the pedal into an electric signal, similar to the produced by the vehicle itself when are being driven by a human.

5.2.3 System Management Layer

According to [Braid, Broggi and Schmiedel \(2006\)](#) and [Jo et al. \(2015\)](#) the *System Management* is basically responsible for three functions of the autonomous system, the *Driving Mode Management* (DMM), *Human Machine Interface* (HMI), and *Health Monitoring System* (HMS). The former deals with the switching between the operation mode of the vehicle, enabling or disabling the autonomous operation due to a trigger event, which can be the user's request or unexpected events, such as in the level 3 of autonomy (see [Table 1](#)) in which if something wrongs or if some event that the autonomous system is not able to handle occurs, the system gives the control back to the driver, in a maneuver called handover. In the current software architecture of *CaRINA II*, this function is not implemented by software yet, however, there are physical buttons in the vehicle which can disable the hardware and allows the driver takes the control back.

The second function, *Human Machine Interface*, is responsible for allowing the users to give commands to the autonomous system, such as the desired destination. As *CaRINA II* is also a research platform other visual feedback are need for the researchers to debug the autonomous operation. For this, simple *User Interface* (UI) and Control Monitoring (CM) were implemented. The UI present the position of the vehicle in the map, and also let the users chose the destination of the vehicle, while the CM shows the path, curvature reference and speed profile that the controllers will execute in the present time interval. Other visual feedback is made by the RViz ([RVIZ, 2018](#)), a powerful ROS tool, in which among other features is able to visual plotting messages data, such as point cloud for LiDAR, and images of the cameras. The architecture has also a plugin developed for RViz, which visually present the obstacle messages (see [5.2.6.2](#)). [Figure 29](#) shows the screenshot of the UI, CM, and RViz.

Figure 29 – Human Machine Interface for CaRINA II

Source: Elaborated by the author.

The last function is the *Health Monitoring System* (HMS), which is essential to allow the Autonomous Vehicles reaches the levels 4 and 5 of autonomy, in which they are called fully-autonomous (see Table 1). This feature is responsible to supervise all software and hardware components, checking the integrity of these components, and emitting alerts in case of faults encountered. In *CaRINA II*, a hierarchical component based monitoring system was implemented, using a hybrid approach for Fault Detection and Diagnosis with Hybrid Bayesian Networks with Residual Evidences for the GPS and Controllers. This system is detailed explained in chapter 6.

5.2.4 Control Layer

As already stated in section 2.1.4, the *Control* layer executes the actions defined by the *Decision-Making and Planning* layer components, guiding the vehicle along the path towards the destination. To achieve that, the *controllers* use the reference planned for the position, curvature, heading, and speed to generate the commands of steering angle, brake pedal percentage, and throttle pedal percentage.

Similar to most of the Autonomous Vehicles presented in the section 3.2 and the earlier software architecture, *CaRINA II Control* layer is divided into two controllers, the lateral and the longitudinal controllers. The lateral controller assumes the Ackermann geometry equations

and uses the current speed, steering angle and position of the vehicle, to compute the lateral and angular errors between them and its reference in the target point in the path, which are used for estimate the desired steering angle. Thus, the point in the path is defined by:

$$T_i = (x, y, \kappa, \theta, l, v) \quad (5.1)$$

where T_i is the i -th point of the path, x and y are the position in UTM coordinate system, κ and θ are the curvature and heading references for the point, l is the position of the point with regard to the distance to the initial point of the path, and v is the speed reference.

In turn, the longitudinal controller receives the speed reference from the above mentioned controller, and uses the current speed of the vehicle to feed a Proportional-Integral (PI) controller that estimates the desired acceleration, keeping the vehicle at the desired speed. The PI has two modes of operation, for braking and acceleration, each one with different constants values. Finally, the acceleration is converted into brake and throttle pedals position, using a normalized function which also takes into account the engine speed and car gear (FILHO *et al.*, 2014).

5.2.5 Mapping Layer

The *Mapping* layer is derived mainly from the work of Silva and Grassi (2018), which presented a road network model with road lanes geometry parametrized using clothoid curves, which can also be used for online interpolation in path planning algorithms. This model is composed of curved lanes, double-lanes, crossing, and roundabouts, organized topologically in graph form (Fig. 35), as follows:

$$M = (I, R) \quad (5.2)$$

in which I is a set of intersections (vertexes), that may be crossing or roundabouts, whereby lanes of different roads are connected. In turn, R is a set of roads, representing the edges of the graph, being its length the weights of the edge.

Therefore, each road r_i belonging to R , being $i \in \{1, \dots, n\}$, and $n \geq 1$ the amount of roads of the map, has a set of lanes L , with $k \geq 1$ lanes, in which each lane l_k is defined by:

$$l_k = (S_l, k, \lambda_i, \lambda_o, r_{off_i}, r_{off_o}, \mathbf{q}_s, \mathbf{q}_e) \quad (5.3)$$

where S_l is a set of parameters of all curves that describe the lane, k is an index that uniquely identify the lane on the road, λ_i , λ_o , r_{off_i} and r_{off_o} are parameters used in path planning algorithm for the entrance and exit of the lane, and $\mathbf{q}_s$ and $\mathbf{q}_e$ are path configuration at the initial and final position of the lane, defined by:

$$\mathbf{q} = (x, y, \kappa, \theta) \quad (5.4)$$

being x and y the position of the point in UTM coordinate system, and κ and θ the references for curvature and heading, respectively.

In turn, each intersection stores a set J , with $m \geq 1$ links between roads. Every j_m is defined by the entrance and exit roads. In addition, each intersection may be a crossing or roundabout, in the latter case also having the radii of each lane and the center of the roundabout in UTM coordinate system stored.

5.2.5.1 Road Network Description

Based on the road network model aforementioned, the OpenStreetMap, OpenDRIVE, and Lanelets standards description were analyzed, however, none of them present a properly structure for accommodate the model proposed in [Silva and Grassi \(2018\)](#), being the parametrized curve for road geometry representation and the human-readable characteristics the main aspects considered. The OpenStreetMap was the most flexible standard because of the simple and versatile structure, however the final result would not be intuitive enough to facilitate the creation and modifications of the maps, since the set of tag names is not sufficiently descriptive. Therefore, a new model description was created to describe the road network model. Figure 30 shows the XML hierarchy structure (Fig. 30a) and the class diagram (Fig. 30b) for this new model.

This XML structure is based on the road network model of [Silva and Grassi \(2018\)](#) with the addition of context road information such as in OpenDRIVE standard. Thus, the structure has three main elements: *Metadata*; *Road*; and *Intersection*.

- **Metadata:** this element provides high-level information about the map, such as the *name* of the city or place described in the road network model. Another important information is the *datum*, a reference point in the map in UTM coordinate system, which can be used to compute the odometry with respect to the map. The last information is the modified *date*, for management purpose (e.g. change logger).
- **Road:** this element describe a single road structure. Each road has at least one *lane* and two *intersections*, one for entrance and other for exit. It also has a set of traffic signs that defines the rules for driven in the road.
 - *Lane:* stores the parameters shown on Eq. 5.3 into the map file. Thus, the attribute *index* is the unique identifier (k) of the lane in the road lanes set, and *init* and *final* are the initial ($\mathbf{q}_s, \lambda_i, r_{off_i}$) and final ($\mathbf{q}_e, \lambda_o, r_{off_o}$) path configuration. Finally, *length* is the size of the lane, and it is also used as the weight for the edges in the graph representation of the road network, and *sharp_length* (S_l) is the set of parameters of all curves that describe the lane geometry.

Figure 30 – Map XML Diagram

Source: [LRM \(2018a\)](#).

- *Traffic Sign*: is a traffic sign representation belonging to the road. Each traffic sign has a *name*, *type*, and *value* which are used for semantic information about the sign, for example, the name '30km' of type 'speed limit' and value of '8.33 m/s'.

A traffic sign also has a position (*pose*) in UTM coordinate system and its localization (*length*) with regard to the distance to the initial point of the lane that it belong to (*lanes*). One constraint add to this model is that each road has at least one traffic sign of type 'speed limit', since it is required for defining the speed profile for navigation.

- **Intersection**: the map has at least two intersections, since each road must have an entrance and exit vertex in the graph representation. Then, each intersection has an *index* that uniquely identifies it in the set of intersections of the map, and a set of links of roads entrance and exit, stored into *roads_in* and *roads_out* attributes. Beyond that, each intersection may be a crossing or roundabout kind (*itype*), and in the latter case the position of the center in UTM coordinate system are stored in *center* attribute, as well as the internal and external lanes radius in the *internal_radius* and *external_radius*, respectively.

5.2.5.2 Map Service

The map model described in the previous section is one of the main components of the new software architecture of *CaRINA II*, since as can be seen in Figure 28 many software components depends on it, Table 2 presents the current use of the map for each component.

Table 2 – Map Features

Component	Description
<i>Localization</i>	lane-level localization of the vehicle
<i>Obstacle Supervisor</i>	lane-level localization of obstacles
<i>Traffic Signs Supervisor</i>	localization of traffic signs for the behavior planning
<i>Route Planning</i>	route computation using Dijkstra algorithm
<i>IHM</i>	map plotting for visual feedback for users and for choosing the destination

Source: Elaborated by the author.

In the previous architecture design, each component should load the map by itself to be used, however, this approach decreases the scalability and flexibility for adding new features in the architecture, therefore, a Client-Server approach was adopted to centralized the task of map management and allow other components to request copies or more specific information of the map with more convenience and ease. Table 3 describes in more details the services currently offered.

Table 3 – Map Services

Service Name	Description
<i>GetTopologicalMap</i>	returns a topological representation of the map for route planning
<i>GetLocalization</i>	returns a lane-level localization of a given point in UTM coordinate system
<i>GetRoad</i>	returns all parameters of a single road
<i>GetIntersection</i>	returns all parameters of a single intersection
<i>GetDatum</i>	returns the reference point information of the map

Source: Elaborated by the author.

Although centralized approaches have the single point of failure problem, which must certainly be taken into account, it also allows the development of complex components such as the *map manager* which, in addition to providing information for the remaining components of the architecture, can be used as a gateway for information coming from communication with the city infrastructure (V2I) or other vehicles (V2V) to update data such as traffic flow of roads and state of traffic lights, as used by the vehicle *BerthaOne* (see section 3.2.3), which includes in its software architecture the information coming via V2V and V2I as new sensors. Thus, the *map*

manager can update the map representation with the verified new information and broadcast them to the remaining components which needs it.

5.2.6 Perception Layer

The *Perception* layer is responsible to transform sensor data into useful information allowing the autonomous system create a representation of the surrounding environment, and consequently takes better decision and safely navigate towards the destination. In the earlier *CaRINA II* software architecture, the LiDAR sensor data was used for obstacle detection avoiding collisions during the vehicle navigation. In this approach, after detecting the obstacle, the speed reference planned is changed to reduce the speed of the vehicle until completely stop or the obstacle leaves the path.

Therefore, in this new design the obstacle detection algorithm was changed, and new perception components added, being: an object detection and classification using YOLO version 3 (REDMON; FARHADI, 2018); obstacle detection using LiDAR with lane-level localization; the lane-level localization of the vehicle; and traffic signs supervisor using the annotated data belonging to the map.

5.2.6.1 Localization

The localization of *CaRINA II* is based on DGPS with RTK correction, which gives the position of the vehicle in the geodesic coordinate system with precision of centimeters. Before being used by the remaining architecture components, the geodesic position is converted to UTM (Universal Transverse Mercator) position, which is a Cartesian coordinate system for localization on Earth, that is, a global reference localization system. The DGPS also gives the orientation of the vehicle in Euler angles.

Another localization provided in the architecture is the lane-level position of the vehicle, that gives information about which road and lane the vehicle is in the map. To achieve that, the component uses a copy of the map to restore all waypoints and stores them into a Bucket Point Region Quad-Tree (SAMET, 1990) (BPR Quad-Tree) as shown on Figure 31. This data structure stores spatial points in buckets, and when there is an overhead in the storage capacity, it recursively subdivides the bucket into four equal sub-buckets while there is still overhead.

The BPR Quad-Tree was changed to store in each point the UTM position of the waypoint, the road and lane indexes, and the position of the point with regard to the distance to the initial point of its lane. In turn, each bucket is defined by four points which defines a square area in the space. The search method in the BPR Quad-Tree was changed to allow search the closest point to the queried, thus, a 1-NN (Nearest Neighbors) is used for searching the queried point using all stored points in the bucket. This search method is $O(c + \log n)$ in the worst case, in which n is the height of the tree and c the capacity of the bucket.

Figure 31 – Map Localization using Bucket Point Region Quad-Tree

Source: Elaborated by the author.

5.2.6.2 Obstacle Detection

Obstacle detection is an essential task for autonomous vehicle safely navigate. The approach adopted in this software architecture makes use of the LiDAR point cloud data and clustering algorithms to detect obstacles around the vehicle. After this, each lane-level position of obstacle is found and the information is stored in the obstacle message (Fig. 32), which is a ROS structure for data exchange between nodes, created to merge information from different sensors, in this case the LiDAR and the map. An example for the use of the structure is associate the semantic information from the *object detection* (e.g. YOLOv3), and the *point cloud clustering*, providing more details about the obstacle detected. Another example is the fusion algorithm presented in [Rosero and Osório \(2017\)](#), which uses information of the RADAR, stereo vision and LiDAR.

The obstacle message has mainly three entities for storing different information about the obstacle detected, being: *Classes*; *ObjectLocalization*; and *ExtraInfo*. The former aims to stores semantic information about the obstacle, such as the result of classification algorithms, and the shape, e.g. 2D or 3D bounding box or polygon representation. In turn, the *ObjectLocalization* provides information about the localization and movement, for example, position, orientation, speed, acceleration, and lane-level localization, which is the road, intersection, and lane indexes. The *ExtraInfo* is used for extra data storage, namely the RADAR detection attributes and point cloud from LiDAR or stereo vision.

The completed flowchart of the obstacle detection with lane-level localization is shown in the activity diagram in Figure 33. The first step consist on remove the floor of the LiDAR point

Figure 32 – Obstacle Message Class Diagram

Source: Elaborated by the author.

cloud using a height threshold. Then, a height map is built, which correspond to discretizing the xy -plane of the point cloud into a grid, which contains cells with equals sizes. If the difference between the minimum and maximum z -coordinate of the points belonging to the same cell is greater than a threshold, this cell is said to be occupied. The next step applies the DBSCAN algorithm for clustering the 2D point cloud, and retrieves the bounding box of each cluster using Convex Hull and Rotating Calipers algorithms (ROSEIRO; OSÓRIO, 2017).

After this, a filter is applied in all obstacles based on the Euclidean distance to the vehicle, remaining just obstacles within a range distance considered a warning region. For each of these obstacles, its lane-level localization is performed by means of a 1-NN algorithm with the trajectory points (Eq. 5.1) provided by the map and belonging to the region. This approach also allows the identification of the obstacles in front or behind the vehicle, since each point has a position with regard to the initial point of the lane, in the same way as the lane-level localization of the vehicle published by the *Localization* module (see section 5.2.6.1).

Figure 33 – Obstacle Detection Activity Diagram

Source: Elaborated by the author.

5.2.6.3 Traffic Signs Supervisor

A new module added to the software architecture is the *Traffic Signs Supervisor*, which uses the annotated information in the map (see section 5.2.5) to provide information about which traffic sign the vehicle is subject to. This information is used by the behavior planning to perform the appropriate action during vehicle navigation. Figure 46 shows a segment of the map with the annotated traffic signs.

After receiving a copy of the map, the module retrieves the position of each sign, and send it to the behavior planning according to the current lane-level position of the vehicle, since every traffic sign has its location in UTM coordinate system, and also with respect to the initial point of the lane. Thus, using this last position, it is possible to select only the closest traffic signs, which have not yet been overtaken by the vehicle.

One important aspect to use a specific software component for traffic sign supervisor, is that in future work the annotated information can be replaced by a module with detection and classification of traffic signs using more traditional approaches of perception. It will be also possible to sensor fusion between the information of the annotated data and the other algorithms, without much impact on the remaining components of the autonomous system.

5.2.7 Decision Making and Planning Layer

As already stated in section 3.3, traditional approaches for navigation architecture in Autonomous Vehicles use a hierarchical organization of the components, starting in the route planning and ending in the controllers (PADEN *et al.*, 2016). However, according to Wei *et al.* (2014) this structure has several drawbacks for real-time application and to deal with dynamic traffic situations. To overcome this problem, the authors indicate that a combination of hierarchical and parallel components structure should be used, which allows a faster replanning in dealing with changes in previous decision.

Therefore, Figure 34 shows the navigation architecture for *CaRINA II*, a subset of the entire architecture presented in Figure 28. This architecture is based on the hierarchical

composition, but uses a parallel organization between the *path planner* and *behavior planner* components of the Autonomous Vehicle navigation pipeline. To improve the performance in dealing with dynamic situations, the task of path planning is divided into *Path Reference Planning*, in which the trajectory of the vehicle is planned considering only the road network, and *Local Path Planning* which deals with the dynamic aspects of the traffic road, such as lane change, emergency stop, lane following, vehicles overtaking, or other maneuvers. The *behavior planner* was also divided into two components, the *Route Segment Planner* and *Behavior Planning*, the former takes as consideration the road network and the latter the perception information.

The difference between the parallel composition presented in Wei *et al.* (2014), and the organization of the *path planner* and *behavior planner* in the CaRINA II architecture, is that these components are not completely independent of each other, such as the controllers in the control layer shown in the authors' work. The parallelism shown in Figure 34 occurs with these components belonging to the same layer in the hierarchical design, which allows the algorithms in both components have more details improving the performance of their tasks. The following subsections describe with more details each component of the architecture.

Figure 34 – CaRINA II Navigation Architecture

Source: Elaborated by the author.

5.2.7.1 Route Planning

After received a destination position chosen by the user via graphic interface, the *Route Planning* component search for the shortest distance route between the current position (P_s) and the destination (P_f), using *Dijkstra* algorithm (DIJKSTRA, 1959) with a directed multigraph, where the intersections are the vertexes, and roads are the edges, as shown on Fig. 35. The length of the road defines the weights of the edges. Before compute the route, two virtual nodes is

included in the graph, P_s and P_f . To include these virtual nodes, four edges are also created, being its weight defined by the distance that P_s is from the final point of its lane, and P_f from the initial point of its lane. The output of this component is a list of roads and intersections with all parameters needed for path planning task.

Figure 35 – Directed Multigraph USP Campus 2

Source: Elaborated by the author.

5.2.7.2 Behavior Planner

- **Route Segment Planner:**

This module uses a high-level abstract description of the road network to decide in which lane the vehicle has to be at the end of the road, to enter in the next intersection, and the lane the vehicle should use to enter in the next road. It also uses the information of the speed limit of the road to fulfilling the request needed for the *Path Reference Planning* module in the next level of the navigation pipeline. Finally, this component also coordinates the execution of each route segment, that is, during the vehicle navigation, the component calculates an approximate time to perform the computation of the path for the next road, based on the length of the road, and request the planning to the *Path Reference Planning* within the available time.

- **Behavior Planning:**

This component is responsible for the maneuver-level planning for the autonomous navigation. It uses the information of the perception modules (e.g. obstacle detection and traffic sign detection), the route segment instruction of the *Route Segment Planner*, the reference path by the *Path Reference Planning*, the feedback of actions received by the *Local Path Planning*, and the lane-level localization.

This module uses the route segment description to know as soon as enter in a new road, if the vehicle should change its lane, and if it needs, a request is sent to the *Local Path Planning* to perform the maneuver. In addition, it is also responsible to provide information and manage the behavior-trees for the maneuvers implemented. In the current version of the architecture, the ‘speed bump’ sign and the ‘emergency stop’ maneuver were developed to prove the applicability, flexibility and other advantages of the technique. Therefore, in section 5.2.7.4 the behavior trees developed are detailed explained.

5.2.7.3 Path Planning

- **Path Reference Planning:**

This component receives a route segment from the *Route Segment Planner* and plan the path for each segment and its junction with the next intersection, using the methods described in [Silva and Grassi \(2018\)](#) and [Silva and Grassi \(2017\)](#), until the vehicle reaches the destination. This module is also responsible for the speed profile for the segment planned, which uses the speed limit of the road.

After planned a segment, the component send it to the *Behavior Planning* and *Local Path Planning*, and wait for the next segment. The path output is named reference path because the planning task considers only the road network, since the obstacles and other dynamic aspects for navigation are taken for consideration by the *Behavior Planning* and *Local Path Planning*.

- **Local Path Planning:**

It is responsible for the maneuver-level path planning, after receiving commands from the *Behavior Planning* describing each maneuver to be performed, and the reference path for the route segment from the *Path Reference Planning*. The maneuvers can be classified as those that need a replanning in the path or just in the speed profile, for example, a ‘lane-change’ maneuver needs a different path, and the ‘emergency stop’ just that the vehicle stop avoiding the crash. In this version of the software architecture are implemented three maneuvers, being: lane-change ([SILVA; GRASSI, 2018](#)); emergency stop; speed bump traffic sign.

Therefore this module is able to apply constraints in the maximum allowed speed or change the reference path. Besides this, it sends short horizontal length segments of the overall path to the *Control* layer to be performed. The frequency with each segment is sent depends on the current speed of the vehicle and the path planned, that is, if the vehicle is at high speed, the segments are sent more often, otherwise, less frequently, but is always guaranteed that the *Control* layer has a path a few meters ahead of the vehicle. This approach is important because it allows a short delay between both modules (*Local Path Planning* and *Controllers*). The length of the path sent to the next layers is also

important to allow sudden changes in the path, for example, an emergency maneuver to avoid collisions.

5.2.7.4 Behaviors

Behaviors are specific actions, or high-level motions, that allows the vehicle to achieve the appropriate level of autonomy and safely perform all its tasks. As already mentioned in section 4.2, FSM is well-known technique for this task, on the other hand, has serious disadvantages considering scalability, flexibility, maintainability, and reusability, which are important characteristics for large and complex systems. Therefore, in this software architecture an approach that is recently being applied in autonomous robots was chosen to model the behaviors of *CaRINA II*, namely Behavior-Tree. Currently two behaviors were modeled using the behavior tree, being the speed bump sign handling, and emergency stop.

- **Speed Bump Sign:**

Speed bumps are vertical deflections in roads used to require vehicles to reduce their speeds in order to improve safety in the near area. This sign is usually located close to pedestrian crossing or school zones. Thus, the behavior expected to Autonomous Vehicle handling this sign is to reduce its speed to a safety reference, being able to sudden stop if this becomes necessary.

Therefore, the behavior-tree developed for this task, shown in Figure 36, has two conditions and three actions, which are known as composites, and uses two different control flows. The meaning of each composite is described in Table 4, and the operation of each control flow in section 4.2.1.

The first step is to check if no traffic signal has been detected, if true, then the execution of the tree ends, otherwise the ‘speed bump’ *sequence* is started. In this *sequence*, first it is checked if the sign detected is indeed a speed bump, and if is true the remaining actions is executed, otherwise, the tree ends. The *actions* in the sequence of this behavior sends a speed constraint to the *Local Path Planning* module, and waits until the vehicle overtake the sign, removing the constraint in the end.

Using the same tree is possible to add behavior for other traffic signs. To do that, a *parallel* control flow should be used as parent of the *sequences*. Then after the *condition* ‘no traffic sign detected?’ returns failure, the *selector* sends a tick to the *parallel* node, which send ticks at the same time for all traffic signs *sequences*. Thus, different traffic signs conditions are analyzed at the same time.

- **Emergency Stop:**

The emergency stop is a maneuver that reduces the speed of the vehicle, when an obstacle is detected, until it completely stops or the maneuver is no longer necessary, e.g. when the

Figure 36 – Behavior Tree for Traffic Signs

Source: Elaborated by the author.

Table 4 – Behavior Tree Composites for Speed Bump Sign

Component	Class Type	Composite	Description
<i>no traffic sign detected?</i>	IsSign	condition	checks if a traffic sign was detected
<i>is speed bump?</i>	IsSign	condition	checks if a speed bump sign was detected
<i>constrain speed to 3m/s</i>	ReduceSpeed	action	sends a speed constraint to <i>Local Path Planning</i>
<i>track speed bump</i>	TrackSign	action	tracks the position of a traffic sign
<i>remove constraint</i>	ReleaseConstraint	action	release a specific constraint in the <i>Local Path Planning</i> module
<i>release speed bump sign</i>	ReleaseSign	action	remove a traffic sign of the queue of signs that should be considered by the <i>Behavior Planning</i>

Source: Elaborated by the author.

obstacle leaves the path. Therefore, the behavior-tree developed, shown in Figure 37, has one condition, three actions, and four control flows. The meaning of each composite is described in Table 5.

The first step of this behavior is to check if there is an obstacle in front of the vehicle, to do that it uses a *condition* which return success if any obstacle is detected in the desired position (e.g. in front of the vehicle at the same lane, or behind the vehicle in its left side

lane). A *decorator* called ‘Inverse’ is used to invert the result of this node, thus, when there is an obstacle it returns failure, leaving the *sequence*, otherwise it returns success and execute the next *action* of the *sequence*. This *sequence* is called ‘obstacle free’, hence the final *action* is to remove any obstacle constraint in the *Local Path Planning* module. However, if there is an obstacle detected, the next *sequence*, called ‘obstacle sequence’, is executed. Then, its first *action* is to calculate a reduce speed rate (α) using the safety indicator named Time-To-Collision (ZHENG; SAYED, 2019) and the Time-To-Stop, and send it to the *Local Path Planning* in its final *action*.

The Time-To-Collision (TTC) indicator uses the speed of the vehicle and the distance to the obstacle to compute the approximate time to reach the obstacle, and the Time-To-Stop (TTS) uses the speed of the vehicle and its maximum deceleration to estimate the time until stop completely. Both estimates allow to assess the danger of the situation, since if TCC is close to TTS, then the vehicle needs to reduce its speed with greater urgency, otherwise, it can be reduced smoothly.

Figure 37 – Behavior Tree for Emergency Stop

Source: Elaborated by the author.

5.3 Final Considerations

This chapter presents the earlier and proposed software architecture of the *CaRINA II* vehicle. After this, the proposed architectural design is presented, with details of all the layers

Table 5 – Behavior Tree Composites for Emergency Stop

Component	Class Type	Composite	Description
<i>has an obstacle been detected in the ego lane?</i>	HasObstacle	condition	checks if there is an obstacle in front of the vehicle
<i>release obstacle constraint</i>	ReleaseConstraint	action	remove an obstacle constrain
<i>calculate reduce speed rate (TTC)</i>	ObstacleWarning	action	calculates a speed decrease rate using Time-To-Collision metric
<i>constrain speed to α</i>	ReduceSpeed	action	sends a speed constraint to <i>Local Path Planning</i>

Source: Elaborated by the author.

and software components. This design has several new components than the earlier architecture, some of them coming from works already done by researchers in the laboratory group, and others are contributions of this master dissertation.

The major difference between the two designs is the inclusion of a new road network description with semantic traffic information, such as the position of the traffic sign. This model was built based on the work of [Silva and Grassi \(2018\)](#), and the OpenDRIVE standard ([DUPUIS; HEKELE; BIEHN, 2019](#)), and it supports the path planning and behavior planning algorithms providing more autonomy to the vehicle. A *map manager* was also implemented using Client-Server paradigm, making ease the use of the map by other components of the architecture.

For perception, a lane-level localization using a Bucket Point Region Quad-Tree was also implemented, and allows the vehicle to locate itself on the map. Besides that, an obstacle detection with lane-level localization using the LiDAR, and a traffic sign supervisor using the annotated data were also included. This new information and environment representation made possible the development of behavior using the Behavior-Tree technique, currently a lane-change, speed bump sign, and emergency stop are available.

These behaviors are used by the *Behavior Planning* component belonging to the hybrid navigation architecture, which is based on the traditional hierarchical architectures, but also has a parallel organization between the path planning and behavior planning components. Behavior-Tree was chosen as technique for model the maneuvers of the vehicle, once it has important advantages compared to Finite-State-Machines, such as the scalability, flexibility, maintainability, reusability, which can be seen by the Tables 4 and 5 that different behaviors use the same component implementation (class type).

DIAGNOSTIC AND MONITORING SYSTEM

This chapter describes the *Health Monitoring System* developed in the *CaRINA II* software architecture, detailing the general approach based on a Component-based Hierarchical Monitoring system, and the Fault Detection and Diagnosis using a Hybrid Bayesian Network with Residual Evidences method for the GPS and Controllers components of the architecture.

6.1 Health Monitoring System

A *Health Monitoring System* is responsible for monitoring hardware and software components of the autonomous system, so that safety measures are taken as soon as an abnormal condition is detected, avoiding the error propagation, which may lead the vehicle to hazardous situations (ESPERON-MIGUEZ; JOHN; JENNIONS, 2013; LANIGAN *et al.*, 2011). As mentioned in section 4.3, this component is common in automotive systems, although most of the approaches adopted are for offline verification, mainly used for detecting faulty components that need to be changed or for maintenance. However, for Autonomous Vehicles on-line techniques are essential.

Therefore, for *Health Monitoring System* accomplishes its task some steps are needed, the first is that the fault or the abnormal condition must be detected, thus if a fault is present in the system, it will be possible to take actions to recover, such as return the control to the driver, or recuperate by itself, using protocols for the specific situation. Some of the Autonomous Vehicle presented in section 3.2 has adopted recovery protocols, being ‘reboot the system’ in *Tongji*, ‘stopping the vehicle’ in *AI*, and ‘hardware redundancy’ in *CMU*.

Hence, in *CaRINA II* software architecture a *Health Monitoring System* was implemented using a Component-based Hierarchical approach, shown in Figure 38, in which the monitoring task with Fault Detection and Diagnosis is divided into component units organized hierarchically (LANIGAN *et al.*, 2011). Each level of this hierarchy is responsible to check a subsystem, and

the top level aggregate all information produced by the sub-levels. This approach is similar to that adopted in the Smart Truck (Fig. 26b), described in author's previous work (GOMES *et al.*, 2018).

Figure 38 – Component Based Health Monitoring Hierarchy

Source: Gomes *et al.* (2018).

The monitoring and diagnosis system was developed by observers, which analyze components of the autonomous system. Each observer uses a model of the component's functioning, so it is able to recognize when some unexpected behavior is present. Thus, the hierarchy level has only two layers. At the bottom, there are three components for monitoring critical modules of the autonomous system, being the *Control Monitor* for the controllers, the *CAN Monitor* for the vehicle state estimation, and the *GPS Monitor* for the localization. The last monitor, named *Nodes Monitor*, is responsible to check if all components of the autonomous system are running during the vehicle operation. The top layer aggregate all diagnostic information produced in a single message with the overall diagnosis of the autonomous system. Each of these monitors are explained below in more detail.

- **Control Monitor:**

The *Control Monitor* analyzes the lateral and longitudinal controllers. Each of them estimate an error for the heading, lateral position, curvature, and speed of the vehicle. These estimations feed a Hybrid Bayesian Network, which do the Fault Detection and Diagnosis task. This system is detailed described in section 6.2.1.

- **CAN Monitor:**

The *CAN Monitor* basically examines how often messages used for the vehicle state estimation are being published in ROS. The purpose is to ensure that the message that stores the state of the vehicle is being produced at the proper frequency.

- **GPS Monitor:**

The *GPS Monitor* verifies the information produced by the Location and the sensor data, taking into account: RTK correction status; covariance of the sensor data; number of connected satellites; if the sensor is posting some wrong orientation; and the results of integrity and quality tests produced by the sensor. Similar to the *Control Monitor*, these information feed a Hybrid Bayesian Network for Fault Detection and Diagnosis, described in section 6.2.2.

- **Nodes Monitor:**

The *Nodes Monitor* has a list of all autonomous system software units (ROS nodes), and periodically checks if they are running. Each unit received a rating that determines its importance to the full operation of the vehicle. The scale rating is composed by *essential*, *important*, and *optional* classes, since the *CaRINA II* is also a research platform, so in certain tests, not all software components needed to be running. Thus, when a component classified as *essential* is not running, the diagnosis returns an error. Otherwise, if only components considered *important* are not running, a warning is issued. Finally, with all the *essential* and *important* components being executed, then the analysis returned a safe state.

The top-level component, named *Diagnostic Aggregator*, receives the output of all monitors and produces a single message with the overall system state. The outputs of the components can be ‘OK’ if is everything working as desired, ‘WARN’ if an abnormal situation is present, but that can not be considered a fault yet, and ‘ERROR’ if a fault is detected. This information is visually presented for the operator, for appropriate decisions to be taken. Currently, in this version of software architecture, no automatic measure is executed by the autonomous system, being the responsibility of the driver to pay attention to the diagnosis and to make the decisions about the state of the system. Figure 39 shows the visual feedback for the *Health Monitoring System* using the ROS Nodes Monitor tools.

Figure 39 – Health Monitoring System Interface

Source: Elaborated by the author.

6.2 Fault Detection and Diagnosis Methodology

The first step of a *Health Monitoring System* is the Fault Detection, which detects the occurrence of faults or abnormal conditions. To do that, different approaches can be used, as stated in section 4.3, each one having specific characteristics for each monitored system or fault (STANLEY *et al.*, 2013). For example, when the system has a known behavior, which is possible to be modeled, model-based approaches are recommended techniques because of their simplicity and efficiency, however, if the fault signature is very complex, knowledge-based or data-driven methods are common choice, since they can find the fault pattern through the analysis of historical data (NASRI *et al.*, 2019).

After a fault is detected, its cause can be found by means of Fault Diagnosis techniques (LANIGAN *et al.*, 2011). On the other hand, according to Mallick and Imtiaz (2013), even if there are good results with current techniques for detecting and diagnosing faults, a single method is not sufficient to meet all the requirements for a diagnostic system, thus, Amin, Imtiaz and Khan (2018) highlighted that hybrid approaches are being studied, in which more than one technique are used together for accomplished the task.

Mallick and Imtiaz (2013) used a combination of PCA (Principal Component Analysis) for fault detection, and Bayesian Network (BN) for diagnosis in a Dissolution Tank problem. In this approach, the information from the PCA variables contribution is used as evidence for a BN to find the most likely cause of the fault. In turn, D'Angelo *et al.* (2014) applied the information of a fuzzy sets theory to feed a BN, for the problem of turn-to-turn short circuit detection in the stator winding of induction motors. The fuzzy sets theory is used for knowledge representation, while the BN for fault detection.

In Amin, Imtiaz and Khan (2018), PCA, T^2 statistics, and BN, were combined and tested in two benchmarks, the continuous stirred tank heater, and the Tennessee Eastman chemical dataset. In this approach, PCA and T^2 are used for fault detection and the variable contributions plot, which are used in the BN for diagnosis. Finally, in Schwall and Gerdes (2002) residuals from the system model were used as continuous evidence for a Hybrid Dynamic BN (Hybrid-DBN) for offline diagnosis, which considers temporal dependence in the diagnosis. Using the residuals in the Hybrid-DBN, the thresholds often used to detect fault in model-based methods are no longer needed.

The proposed Fault Detection and Diagnosis methodology applied in *CaRINA II* software architecture, is also based on the use of residuals with Hybrid Bayesian Networks (Hybrid-BN), for monitoring the lateral and longitudinal controllers, and also the GPS sensor and localization system. This approach was adopted because model-based methods have excellent performance in detecting faults in controls systems and also in the multipath or antenna masking problems of GPS. Otherwise, threshold-based approaches for multi-models systems needs many adjustments in the values of the thresholds.

Thus, similar to [Schwall and Gerdes \(2002\)](#) the residual values are used as continuous evidence of a Hybrid-BN, which deals with the detection and diagnosis task. Beyond that, BN also allows modeling of multi-symptoms for single fault, which is essential for the FDD, because both the controllers faults and the GPS faults have many ways of being demonstrated. However, different from [Schwall and Gerdes \(2002\)](#), a new layer was added to the Bayesian network, called the symptoms layer, between the evidence and the faults nodes, to simplify the construction of Conditional Probability Tables, facilitating the learning of fault patterns.

Therefore, the developed Hybrid-BN has three layers, the evidence, symptoms, and faults, graphically represented as blue, purple and red ellipses, respectively (Fig. 40 and Fig. 41). Evidence can be discrete or continuous nodes, so for discrete nodes a finite set of outcomes is defined, and for the continuous nodes a probability density function, which in the case of the residuals takes the form of a Gaussian Distribution, defined by:

$$X \sim Normal(\mu, \sigma^2) \quad (6.1)$$

in which, X is a random variable, μ is the mean of the distribution, and σ^2 is the variance.

Continuous nodes also have a finite set of intervals used in the discretization of their values during inferences with discrete nodes. In addition, in the FDD task, two types of inference or reasoning in the network are used, being the *predictive* for fault detection, and then the *combined* for diagnosis, since both the fault and evidence are known, thus, the probability of the symptoms are updated to find the symptom that most contributes to the fault.

After the definition of the network model, the Conditional Probability Tables for discrete nodes, including the symptoms and fault nodes, are learned using Expectation-Maximization (EM) algorithm (see section 4.3.5), since the dataset has missing data, which is common in real-environment application. Only the CPT for discrete nodes are learned, and the parameters of the continuous node are known by parse the dataset, providing the probability density function and its specifications. Therefore, in the next sections, the FDD using Hybrid-BN with residual evidence for the controllers and GPS sensor monitoring are detailed described.

6.2.1 Control Diagnostic

The *Control* layer of the *CaRINA II* software architecture has two controllers, the lateral and longitudinal controllers (see section 5.2.4). The former is responsible to compute the desired steering angle, assuming the Ackermann geometry and kinematic constraints. This controller estimates the lateral, angular, and curvature errors, using the current vehicle state and the target state, towards to the next position of the path. In turn, the longitudinal controller is in charge of keep the vehicle at the desired speed. Thus, the current state of the vehicle and the target state can be defined as follows:

$$\mathbf{q}_{cur} = (x_{cur}, y_{cur}, \kappa_{cur}, \theta_{cur}, v_{cur}) \quad (6.2)$$

$$\mathbf{q}_{ref} = (x_{ref}, y_{ref}, \kappa_{ref}, \theta_{ref}, v_{ref}) \quad (6.3)$$

in which, x and y are the position in UTM coordinate system, κ is the curvature, θ is the heading, and v the speed. The subscript *cur* refers to the current state, and the *ref* to the target state.

A fault in any of the controllers means that the vehicle has a large residue between the current and target state. For example, if the longitudinal residue is large, it implies that the vehicle is failing to reach the reference speed, or if the lateral residue is large, it is possible to invade an opposite lane of the road. Similarly, if the angular residue is substantial, the vehicle can be heading for a wrong direction, and the curvature residue can contribute to this behavior.

Therefore, the Hybrid Bayesian Network with residual evidences is presented in Figure 40. The evidence layer has four nodes, being: *longitudinal error*; *angular error*; *curvature error*; and, *lateral error*. Each of these evidence nodes has a symptom parent, the *longitudinal residue*, *angular residue*; *curvature residue*, and *lateral residue*, respectively. The evidence nodes are continuous and assumes the form of Gaussian Distributions (Eq. 6.1) and each one has a defined interval used in the discretization process. In addition, the symptoms are discrete nodes with five outcomes, being: *very large*; *large*; *medium*; *small*; and, *very small*. The fault node, named *control fault*, is also a discrete node, but it has three outcomes, the ‘yes’ when a fault is detected, ‘no’ for normal operation, and ‘warn’ an intermediary state between the no fault to the fault conditions. The symptoms are explained below.

- **longitudinal residue:** it is associated with the *longitudinal error* (Eq. 6.4), thus this symptom measure the magnitude of the absolute error, which means that when the absolute different between the current speed of the vehicle and the desired speed, or reference speed, is large, the outcome of the symptom is also *large*.

$$longitudinal_error = v_{ref} - v_{cur} \quad (6.4)$$

- **angular residue:** it is related with the *angular error* (Eq. 6.5), and also measure the magnitude of the absolute error. This measurement is important for vehicle navigation since allows to detect when the vehicle has deviated from the direction that leads to the destination.

$$angular_error = \theta_{ref} - \theta_{cur} \quad (6.5)$$

- **curvature residue:** this symptom node is parent of the *curvature error* node, which is associate with the steering angle. Thus, it is possible to detect faults in the steering of the vehicle. The outcomes of this node also refers to the magnitude of the absolute error.

Figure 40 – Control Diagnostic Hybrid Bayesian Network

Source: Elaborated by the author.

$$curvature_error = \kappa_{ref} - \kappa_{cur} \quad (6.6)$$

- **lateral residue:** it is correlated with the *lateral error* (Eq. 6.12), which is an estimation of the lateral deviation between the current position of the vehicle and the closest point in the reference path. This value is used for compute the desired steering angle, and if large, may indicate for example that the vehicle is invading an opposing lane or sidewalk, since the vehicle's reference path is in the center of the lane..

The *lateral error* is estimated using a cross-track error equation (*XTE*) (ROUNSAVILLE; DVORAK; STOMBAUGH, 2016) with the current position of the vehicle and the closest point in the reference path, as described below:

- *1° Step:* calculates the heading vector between the positions, using Eq.6.7.

$$X = \begin{bmatrix} x_{cur} - x_{ref} \\ y_{cur} - y_{ref} \end{bmatrix} \quad (6.7)$$

- *2° Step:* creates virtual vector, *P*, with the same direction of *X*, being:

$$P = \begin{bmatrix} \cos(\theta_{ref}) \\ \sin(\theta_{ref}) \end{bmatrix} \quad (6.8)$$

- 3° Step: calculates the projection (S) and rejection (R) vectors.

$$S = (X^T \times P) P \quad (6.9)$$

$$R = X - S \quad (6.10)$$

- 4° Step: finds the sign of the XTE relation using Eq. 6.11.

$$Z = \left(\begin{bmatrix} \sin(\theta_{ref}) \\ -\cos(\theta_{ref}) \end{bmatrix} \times X \right) \quad (6.11)$$

- 5° Step: calculates XTE , or the *lateral_error*, using:

$$lateral_error = \text{sign}(Z) \|R\| \quad (6.12)$$

6.2.2 GPS Diagnostic

Localization is an essential component of the autonomous system being presented, for example, in the vehicle routing and for obstacles detection. In *CaRINA II* the localization is based on DGPS (Differential Global Position System) with RTK (Real-Time Kinematic) correction, which gives precision of centimeters. However, according to [Bijjahalli, Ramasamy and Sabatini \(2017\)](#), navigation systems based on the Global Navigation Satellite System (GNSS) suffer from problems related to the propagation and reception of satellite signals, such as signal multipath, signal loss, antenna masking, and adverse satellite geometries.

In signal multipath, the signals from satellites reflected from other structures, such as buildings and trees, are received more than once by the sensor at different time intervals, introducing errors in position estimation. In turn, in the signal loss, the strength of the signal is below the minimum required by the receivers preventing its use. The antenna masking problem also occurs due to the surrounding structures, more specifically when there is an obstruction between the antenna and the satellite, degrading signal quality. Finally, the receivers use trilateration with the signals from the satellites to estimate its positions, thus, the positions of the visible satellites greatly influence the accuracy of the sensor's position estimation ([BIJAHALLI; RAMASAMY; SABATINI, 2017](#); [DUTT et al., 2009](#)).

Therefore, the Hybrid Bayesian Network developed to monitor the GPS, shown in Figure 41, has twelve nodes in the evidence layer, six symptoms in the symptom layer, and a fault node. This network also has evidence obtained from residuals of the vehicle's motion model but is not limited to this, since the faults that the sensor is subject to is more complex. Thus, other sources of information were used to characterize some important information that allows the identification of the faults. Another difference between this network and that used for the controllers monitoring is that it contains continuous nodes modeled using other probability

Figure 41 – GPS Diagnostic Hybrid Bayesian Network

Source: Elaborated by the author.

density function than only Gaussian distribution. Each of the symptoms nodes and its evidence is explained below.

• DOP evaluation:

One of the sources of faults or low accuracy in GPS position estimation is the arrangement of satellites being used by the sensor. This organization is also called satellites geometry. According to [Dutt et al. \(2009\)](#) the position of the receiver belongs to an area shaped by the intersection of the coverage regions of each satellite, using an algorithm named 3-Dimensional Multilateration, consequently, if the satellites are close to each other, the intersection of their coverage areas will be larger, increasing the imprecision in the position of the receiver, otherwise, if the satellites are more spread, the area of the intersection will be smaller, allowing a more accurate estimate position.

Dilution of Precision (DOP), also known as Geometric Dilution of Precision (GDOP) is a quantitative measure of the quality of satellite geometry ([LANGLEY et al., 1999](#)).

There are different components of geometric quality, such as the Horizontal Dilution of Precision (HDOP) for horizontal position (latitude and longitude), the Vertical Dilution of Precision (VDOP) for altitude, the Time Dilution of Precision (TDOP) for uncertainty in the receiver clock, and Position Dilution of Precision (PDOP) which combines the HDOP and VDOP by measuring the uncertainty in the 3 coordinates (latitude, longitude and altitude), as shown in Equation 6.13. The overall measurement is given by the Equation 6.14, combining the PDOP and TDOP (DUTT *et al.*, 2009).

$$PDOP = \sqrt{HDOP^2 + VDOP^2} \quad (6.13)$$

$$GDOP = \sqrt{PDOP^2 + TDOP^2} \quad (6.14)$$

As stated in Langley *et al.* (1999), the lower the DOP value, the better is the geometry of the satellites. Thus, Dutt *et al.* (2009) presented the Table 6 based in other researches, which qualifies the DOP values in linguistic terms. This table was used as outcomes of the symptom, named ‘DOP Evaluation’, which has as evidence the continuous node ‘PDOP’. Only PDOP was used in the network, since it already combines the horizontal and vertical uncertainty of the satellites geometry.

Table 6 – DOP Ratings

DOP Value	Ratings
1	ideal
2-4	excellent
4-6	good
6-8	moderate
8-20	fair
20-50	poor

Source: Dutt *et al.* (2009).

- **RAIM status:**

In addition to DOP, which quantifies the quality of the satellite geometry, the RAIM (Receiver Autonomous Integrity Monitoring) algorithm is a method based on statistical test, applied to monitor the reliability and separability of the sensor (HEWITSON; LEE; WANG, 2004; HEWITSON, 2003). The former means the efficacy of detect the presence of outliers in signals received from the satellites, while the latter quantifies the capability of the sensor to identify the source of the outliers.

According to Wang and Ober (2009) the algorithm consists of three steps, being: Detection; Identification; and, Adaptation. The first step is responsible to detect the presence of an abnormal condition, while the second identify the source of outliers, and the latter remove the outlier from the set of measurements of the sensor. These steps are repeated while there

is outlier. Two statistical tests are used in the AsteRx2eH GPS sensor, being the ‘overall mode test’ for the first step, and ‘w-test’ to identify the outliers (SEPTENTRIO, 2014).

Moreover, the algorithm also measures the External Reliability of the outlier detection method, which quantifies the effect of undetected outliers in the final solution. Thus, the sensor provides the Horizontal External Reliability Level (HERL) for the horizontal position (latitude and longitude), and the Vertical External Reliability Level (VERL) for the vertical position (altitude). As stated in Wang and Ober (2009) and Brogi and Tromboni (2015), it is possible to determine a threshold for these values, specific for each application, in order to ensure that the position solution is suitable.

In the developed Hybrid-BN, the results of the RAIM algorithm were included by means of three evidence nodes and a symptom node. The results of the external reliability (HERL and VERL) were modeled as continuous nodes using the Bernoulli probability distribution (nodes ‘*HERL position*’ and ‘*VERL position*’), which is an especial case of binomial distribution, since as mentioned in Brogi and Tromboni (2015) the values can be above or below a specified threshold. In turn, the result of the ‘overall mode test’ is included through the discrete node ‘*RAIM integrity*’, with three possibles outcomes, being: *successful* when the measurements pass the test; *failed* when they fail; and *not-available* when is not possible to perform the test. The symptom node, named ‘*RAIM status*’, also has three outcomes, which evaluates the overall status of the algorithm taking into account the result of the integrity test and the external reliability values, thus, *no* is used when the results meet the desired status, *yes* when they do not satisfy these conditions, and *warn* for intermediate results.

- **status number of satellites:**

As stated in Dutt *et al.* (2009), the receiver requires at least four satellites to be able to estimate its position. This number also contributes with DOP value, since the more satellites visible to the sensor, the more likely it is that the DOP value is lower, indicating quality in the satellites geometry. Another consideration that can be made about the number of satellites is that it can decrease the antenna masking effects, since with more satellites the occlusion of a signal may not affect the accuracy of the estimated position.

Therefore, this information was included in the network with the evidence ‘*number of satellites*’ and the symptom node ‘*status number of satellites*’. The symptom has as outcomes the qualitative terms that define whether the number of satellites is ‘*not enough*’, ‘*enough*’ or ‘*good*’ for the position estimation.

- **status RTK correction:**

Due to the importance of GNSS for different applications, several techniques were developed to improve the accuracy of the position estimation by the receivers. The Differential GPS (DGPS) techniques use one or more stationary base to provide corrections data to the

receivers in real time. According to [Kaplan and Hegarty \(2005\)](#), these techniques can be classified in: absolute or relative positioning; local area, regional area, or wide area; and code-based or carrier-based.

Relative or absolute are related to the desired coordinate system for receiver position, relative to the position of the stationary base, or in a global reference frame, respectively. In turn, the local, regional, or wide areas are associated with the coverage area of the correction system. Finally, code-based uses code measurement (i.e. pseudoranges), while carrier-based uses carrier-phase measurements, being considered superior to the first approach.

Conforming to [Langley \(1998\)](#) and [Kuutti et al. \(2018\)](#), Real-Time Kinematic (RTK) is a well-known technique for correction of the satellites signals, classified as carrier-based, which is based on the principle of two receptors, the rover (i.e. placed in the vehicle) and the base (i.e. stationary receptor with a known position). The base is constantly calculating corrections in relations to its coordinates and sending them to the rover in real-time. This systems provides a position estimation with centimeter-level of accuracy.

Therefore, in traditional implementation of RTK, the base sends the correction data through radio waves, however this approach requires the rover to act in a region limited by signal coverage. An important modification of the method was to make available the correction data on the internet allowing a wide coverage area. The rover, in its turn, access the data using a network link (i.e. 3G or 4G) ([COLOMBO, 2008](#)). On the other hand, even though the coverage area of the correction system has been increased, another point of failure has been added, which occurs when the rover loses its connection with the internet and stops receiving the data, or when there is a delay in reception.

For these reasons, a continuous evidence node, named '*age RTK*', and a symptom node, called '*status RTK correction*', were added to the Hybrid Bayesian Network in order to monitor the status of the RTK correction delay, which is also possible to inform when the receiver is not receiving any correction data (i.e. no internet connection). The symptom node, in its turn, has five outcomes, being: *good*, when the delay is very small; *medium*, for normal age; *bad*, for delays above the normal condition; *very bad*, when the age is too large; and, *no correction*, by the time the receiver is not receiving any correction date.

- **pose variance:**

Variance is a statistical measure that quantifies the dispersion of the data to the center of its distribution or statistical mean. Thus, it is possible to detect abnormal condition (i.e. outlier) just observing the behavior of the variance and co-variance between the random variables when a new measure is included in the historical sequence of the data. Therefore, in the Hybrid BN, the variance values measured by the sensor are presented by the three continuous evidence nodes, named '*x variance*' (x coordinate), '*y variance*' (y coordinate),

and ‘*theta variance*’ (yaw orientation), in which they use a Gaussian distribution to model the reading values.

A symptom node, called ‘*pose variance*’ evaluates these information with five outcomes which quantifies the magnitude of the variance in descending order, being, in this order: *very large*; *large*; *medium*; *small*; *very small*. The higher the symptom value, the greater the fluctuation of the data in relation to the mean of the distribution, which may mean large oscillations in the estimated positions.

- **pose residues:**

When a GPS sensor fails, it can be propagated and affect the vehicle location system leading to dangerous situations, for example, if the current estimated position differs laterally a considerable distance from the previous position after a short period of time, this can lead to a very abrupt adjustment control to correct the lateral error of the vehicle. However, considering the vehicle’s equations of motion, this new estimated position may not be possible due to the kinematic constraints of the motion, thus leading to consider it coming from fault. Therefore, for this type of fault, model-based methods are effective choices for fault detection.

A well-known technique is the Extended Kalman Filter (EKF), a nonlinear version of the Kalman Filter, which is a specific case of Dynamic Bayesian Network (DBN) for state estimation based on the Markov assumption, that is, the next state of the system depends only on the current state. Thus, EKF is an algorithm with two main steps, the *prediction* and *update*, in which the former uses system dynamic function to estimate the system state, while the latter uses the observed information to correct this estimation (RIBEIRO, 2004).

An important information provided by the EKF equations is named innovation, which is similar to residue in model-based fault detection, thus, if the value is above a threshold it is possible to stated that the observed system deviate from the normal conditions, since the difference between the previous state and the current state is higher than the expected or allowed (EYKEREN; CHU; MULDER, 2012).

As stated Ribeiro (2004) and Eykeren, Chu and Mulder (2012), in the first step, named *prediction*, the algorithm uses the system equations (f), which must be differentiable, and the control vector (u) to estimate the state $\hat{q}$, and the covariance of the estimated state P , as shown in Equations 6.15 and 6.17, respectively.

$$\hat{q}_{k|k-1} = f(\hat{q}_{k-1}, u_k) + w_k \quad w_k \sim \text{Normal}(0, Q) \quad (6.15)$$

$$z_k = h(\hat{q}_k) + v_k \quad v_k \sim \text{Normal}(0, R) \quad (6.16)$$

$$P_{k|k-1} = \Phi_k P_{k-1|k-1} \Phi_k^T + \Gamma_k Q_k \Gamma_k^T \quad (6.17)$$

in which $\hat{q}_{k|k-1}$ is the estimated state at current time instance k given the state at time $k-1$, w_k and v_k are the process and observation noise with covariance Q_k and R_k , in this order. $\Phi = \frac{\partial f}{\partial q}$ and $\Gamma = \frac{\partial f}{\partial u}$ are the Jacobians, and z_k the observation values at time k .

After estimate the state vector $\hat{q}$ using the system dynamics equation, the algorithm uses the observation vector z_k at time k and the covariance of the state estimated (P) to update the state vector and the covariance using the Equations 6.20 and 6.21, in this order. However, before update this variables, a Kalman gain is computed, and also an innovation vector, which is the difference between the observation and the predicted observations. Conforming to Ribeiro (2004) the innovation r_k carried the novelty from the observation for the system state estimation. Another property of r_k is that it represents a white noise with zero mean, similar to residue in model-based methods.

$$K_k = P_{k|k-1} H^T (H_k P_{k|k-1} H_k^T + R_k)^{-1} \quad (6.18)$$

$$r_k = z_k - h(\hat{q}_{k|k-1}) \quad (6.19)$$

$$\hat{q}_{k|k} = \hat{q}_{k|k-1} + K_k r_k \quad (6.20)$$

$$P_{k|k} = [I - K_k H_k] P_{k|k-1} \quad (6.21)$$

where h is the measurement function, $H = \frac{\partial h}{\partial q}$ is the Jacobians for the observation function, K_k is the Kalman gain, z_k is the observation vector, r_k is the innovation vector, and I and identity matrix.

Therefore, in the Hybrid-BN three continuous evidence nodes were included for each variable of the system state, presented in Equation 6.22. The systems equations (Eq. 6.24) were defined according to the vehicle motion and Ackermann geometry. These nodes are children of the symptom node, named *pose innovation*, which has five outcomes, representing the magnitude of the algorithm innovation (Eq. 6.19), in this case, a *large* or *very large* outcome constitute a fault condition, while the remaining outcomes indicates normal condition (*small* or *very small*) or awareness state (*medium*).

$$\hat{q}_k = (\hat{x}_k, \hat{y}_k, \hat{\theta}_k) \quad (6.22)$$

$$u_k = (v_k, \kappa_k) \quad (6.23)$$

$$f(\hat{q}_{k-1}, u_k) = \begin{cases} \hat{x}_k = \hat{x}_{k-1} + v_k \cos(\hat{\theta}_k) \Delta t \\ \hat{y}_k = \hat{y}_{k-1} + v_k \sin(\hat{\theta}_k) \Delta t \\ \hat{\theta}_k = \hat{\theta}_{k-1} + \kappa_k v_k \Delta t \end{cases} \quad (6.24)$$

where x_k and y_k are the UTM coordinates of the vehicle position at time k , θ_k is its yaw orientation, v_k is the velocity, κ_k is the curvature, and Δt is the time variation between each iteration of the algorithm.

6.3 Final Considerations

This chapter describes the Health Monitoring System of CaRINA II, a Component-based Hierarchical system which monitors four important modules of the autonomous system, being: *CAN Monitor*; *Nodes Monitor*; *Control Monitor*; and, *GPS Monitor*. The first two are responsible for verifying whether the vehicle's state information is being published in the architecture, and whether the components required for autonomous actuation are being performed. In turn, the last two monitors are in charge of Fault Detection and Diagnosis with the lateral and longitudinal controllers, and with the GPS sensor and localization module.

The Fault Detection and Diagnosis approach adopted is based on a hybrid method which combines model-based and knowledge-based techniques, more specifically the residue estimation with observers and a Hybrid Bayesian Network, similar to [Schwall and Gerdes \(2002\)](#) and other works which also use residues. On the other hand, different of these work, a new layer, named *symptom* layer, was included in the network for online detection and diagnosis, and also different source of information were combined with the residue due to the complexity of the fault signature.

The evaluation of the proposed approach will be discussed in the chapter 7. An important consideration for Health Monitoring Systems for Autonomous Vehicles is that from level 4 of autonomy the autonomous system must be able to handle fault situations, so diagnostic and prognostic information of system components must be included in the decision-making increasing the reliability of the system. One area of study the use of diagnostic information in decision making is called *Integrated Vehicle Health Management* ([BENEDETTINI et al., 2009](#)), which involves the monitoring of vehicle condition and the use of its information in the decision process, for example, for choose of maneuvers, mission planning, system recovery of fault, or others. The current *CaRINA II* software architecture, the decision which takes into account the diagnosis is responsibility of the human driver, however in future works this functionality will be added to the architecture.

RESULTS

This chapter presents and discusses the results obtained through experimental tests involving the software architecture proposed for *CaRINA II* and presented in this Masters dissertation. Section 7.1 describes the results of the Hybrid Bayesian Network with Residual Evidence for Fault Detection and Diagnosis applied in the ‘Control Module’ and ‘GPS Module’ of the Health Monitoring System (see chapter 6). In turn, Section 7.2 details the results of the overall software architecture (see chapter 5) with a real experiment.

7.1 Diagnostic System

According to Venkatasubramanian *et al.* (2003) a Fault Detection and Diagnosis system should meet performance requirements which is also useful for comparing different approaches. Some of these requirements are: the time for detection and diagnosis; isolability, which is the ability to distinguish between different faults; robustness, which is an indication of the ability to withstand noise and uncertainties; and, classification error estimation, since the FDD techniques are also classifiers.

Therefore, confusion matrix analysis is applied for evaluate the proposed Hybrid Bayesian Network with Residual Evidence for detecting and diagnosis faults in the controllers and the GPS sensor. This method allows to measure the performance of supervised classification algorithms with respect to a testing dataset. Thus, a two-dimensional matrix is constructed, in which the rows represent the true outputs, and the columns the predicted outputs (HOSSIN; SULAIMAN, 2015). With this matrix, different performance metrics can be calculated for analyzing the algorithm, such as accuracy, misclassification rate, sensitivity, precision, among others.

In addition, real case experiments were also performed. In this case, sequences of sensor readings were recorded in different scenarios for evaluate the output of the fault and symptoms nodes. Some of the recorded data include real occurrences of faults and also modified by

Gaussian Noise insertion. In turn, the Hybrid-BNs were developed using the Python wrapper of the 'SMILE Engine'¹ library. The next subsections describe the experimental results of the *Control Monitor* (section 7.1.1) and *GPS Monitor* (section 7.1.2).

7.1.1 Control Monitor

The *Control Monitor* analyzes the lateral and longitudinal controllers, using the estimation of the lateral error, angular error, curvature error, and longitudinal error (see section 6.2.1). These errors, also named residues, are normally distributed (Eq. 6.1), thus a fault is characterized by a large deviation from the mean of the distribution ($\mu = 0$). These residues are inserted in the Hybrid-BN through the continuous evidence nodes named *lateral error*, *angular error*, *curvature error*, and *longitudinal error*. In addition, each evidence has an associated symptom node, named *lateral residue*, *angular residue*, *curvature residue*, and *longitudinal residue*, respectively, which measure the magnitude of the absolute residue.

The parameters of the symptoms and fault nodes, or the a priori knowledge of the network, which are stored in the Conditional Probability Tables (CPT), are learned by means of the Expectation-Maximization algorithm, using a training dataset with labeled entries. Each entry is composed by the values of residues, the outcome of each symptom, and the outcome of the fault node, as shown in Equation 7.1.

$$\mathbf{d}_k^{\text{control}} = (\textit{lateral_error}, \textit{angular_error}, \textit{curvature_error}, \textit{longitudinal_error}, \\ o_{\textit{lateral}}, o_{\textit{angular}}, o_{\textit{curvature}}, o_{\textit{longitudinal}}, o_{\textit{control}}) \quad (7.1)$$

in which *lateral_error*, *angular_error*, *curvature_error*, and *longitudinal_error* are defined by the equations 6.12, 6.5, 6.6 and 6.4, respectively. In turn, $o_{\textit{lateral}}$, $o_{\textit{angular}}$, $o_{\textit{curvature}}$, $o_{\textit{longitudinal}}$ and $o_{\textit{control}}$ are the outcomes of the symptoms nodes and the fault node, in this order.

Therefore, for learning and testing, a dataset was created with real data, collected using the *CarINA II* in autonomous and manual driving modes, and with the CARLA simulator, using the autonomous driving agent winner of the first CARLA Autonomous Driving Challenge², developed by the LRM research team. This dataset was divided into training and testing datasets, as shown in Table 7, with the ratio of 80% for training and 20% for testing. The labeling was done automatically comparing the magnitude of the residues, and then verified by the human expert.

After learning the CPTs of the discrete nodes (symptoms and fault), the testing dataset was used for creating the confusion matrix, presented in Table 8, by means of reasoning with each entry of the dataset. The average time of inference is approximately 2ms, which is sufficient

¹ BayesFusion, LLC.

² CARLA Autonomous Driving Challenge

for real-time applications, such as on-line Fault Detection and Diagnosis for health monitoring autonomous systems.

Table 7 – Control Diagnosis Dataset Division in Training and Test

Control Fault Outcome	training	test	% training	% test
yes	183201	45941	27.03	27.12
warn	70950	18131	10.47	10.70
no	423437	105324	62.50	62.18
<i>total</i>	677588	169396	100.0	100.0

Source: Elaborated by the author.

The accuracy of the overall network model is 87.965%, being the True Positive Rate (TPR), also known as sensitivity or recall, of the *yes* outcome equals to 0.964, which means that when the entry is actually *yes*, the outcome of the *control fault* node is also *yes*. The recall for the other outcomes are 0.948 and 0.831, for the *warn* and *no*, respectively. All values are close to 1.0, indicating that the network has a high True Positive Rate for all of its outcomes. These results are complemented by the True Negative Rate (TNR), which is 0.979, 0.968 and 0.884 for the *no*, *warn*, and *yes*, in this order, meaning that when the true outcome of the entry is not the target class, the outcome of the node is not that class with TNR as confidence, being 1.0 the maximum value.

Table 8 – Control Diagnosis Confusion Matrix

Unnormalized Confusion Matrix					Normalized Confusion Matrix				
True label	yes	44305	401	1235	True label	yes	0.96	0.01	0.03
	warn	863	17199	69		warn	0.05	0.95	0.00
	no	13390	4429	87505		no	0.13	0.04	0.83
		yes	warn	no			yes	warn	no
Predicted label					Predicted label				

Source: Elaborated by the author.

In all the evaluation metrics aforementioned, the outcome *no* has the lowest value. This is related with False Negative (FN), which happens when the true outcome of the node is *no*, but the predicted outcome is not the same. According to the confusion matrix (Table 8), the most common misclassification occurs when the predicted outcome is *yes*, being actually *no* (approximately 13% of the *no* entries of the testing dataset). An important aspect about the entries that result in these classification errors is that all were incomplete queries, that is, some of the evidence was not provided to the network during the inference, being 31.5% of them did not have the lateral, angular, and curvature error information, and the remaining 68.5% do not have longitudinal error information. As consequence, the predicted symptoms outputs of the missing evidence were always ‘*very_large*’, which is an indication of fault.

In addition to the analysis of the confusion matrix, two experiments were performed, using the residue values from two routes traveled (named Case 1 and Case 2) by the *CaRINA II* in autonomous mode. In the first route (Fig. 42a) the driver took the control of the vehicle to force oscillation in the lateral, angular, and curvature errors, and also for keeping the vehicle above the reference speed. In turn, in the second route (Fig. 43a), Gaussian noise was inserted in the residues values in three different moments of each evidence.

7.1.1.1 Case 1

In this experimental test, *CaRINA II* in autonomous mode was used to collect the residues data from the controllers. However, during the navigation, the driver took the control of the autonomous system to force oscillation in the values of the residues. Figure 42a shows the performed path, being remarkable the moments that the vehicle deviates from the reference path. In turn, Figures 42b, 42c, 42d, and 42e, present the value of the residues and the respective output of its associated symptom node.

The outputs of the symptoms indicate that these nodes perform very well the distinction between the different magnitudes of the absolute residue, since this differentiation is outstanding in the residues plot showed in Figure 42. It is important to highlight that the thresholds which determine how large is the residue were also used to create the dataset, which was used to learn the CPTs of the discrete nodes (symptoms and fault node). Therefore, the differentiation performed by the symptoms nodes, point out that these nodes learned a similar value of threshold for each outcome class (*very_large*, *large*, *medium*, *small*, and *very_small*).

The Fault Detection results for this test are presented in Figure 42f, and demonstrates that in 63.2% of the results are *yes*, while the remaining 36.8% are *warn*. The predominance of *yes* output starts at the moment when the driver starts a zigzag driving ($k = 550$, where k is at x-axis of the graphics), since in each moment at least one of the symptoms are indicating a large residue. Before the moment $k = 550$, the output of fault node is *warn* because the *longitudinal residue* symptom node is noting a *medium* size of residue, which can become *large* if it continues growing. Other quantitative results of the analysis of Case 1 are presented in Table 9.

Table 9 presents the ratio relation between the output of the fault node and its symptoms. Therefore, when the output of the *control fault* is *yes*, 62.179% and 8.242% of the time coincide with the *very_large* and *large* outputs for the *lateral residue* symptom, which represents 92% and 100%, of the time when this node (*lateral residue*) outputs these values, respectively. This behavior also occurs for the remaining symptoms nodes, being 77.564% and 3.938% for the *angular residue*, which correspond to 93.3% and 100% of these outputs occurrence, 20.33% and 5.495% for the *longitudinal residue*, both equivalent to 100% of the circumstance, and 10.806% and 20.33% for the *curvature residue*, which corresponding to 97.5% and 86.4% of these events.

Another fact presented in the Table 9 concerns to the *warn* output. In this case, *small* and *very_small* symptoms' outputs are majorities, but the output of the *longitudinal residue* is

Table 9 – Control Diagnosis Fault Detection Result - Case 1

control fault	lateral residue					angular residue				
	very large	large	medium	small	very small	very large	large	medium	small	very small
yes	62.179%	8.242%	7.051%	11.081%	11.447%	77.564%	3.938%	2.656%	14.835%	1.007%
warn	9.277%	0	0	11.792%	78.931%	9.591%	0	0	85.22%	5.189%
no	0	0	0	0	0	0	0	0	0	0

control fault	longitudinal residue					curvature residue				
	very large	large	medium	small	very small	very large	large	medium	small	very small
yes	20.33%	5.495%	74.175%	0	0	10.806%	20.33%	33.974%	21.795%	13.095%
warn	0	0	100%	0	0	0.471%	5.503%	9.434%	11.164%	73.428%
no	0	0	0	0	0	0	0	0	0	0

Source: Elaborated by the author.

always *medium*, which has a great influence for this result, because a medium size of the residue can grows to large or very large, which is a not desirable behavior.

7.1.1.2 Case 2

The second experimentation (Case 2) uses the data obtained with the *CaRINA II* vehicle in autonomous mode, in which a Gaussian noise were inserted in each residue sequence separately. The path performed are presented in Figure 43a, and similar to Case 1, Figures 43b, 43c, 43d, and 43e, show the residues and symptoms' node outputs, and also the data modified by the Gaussian noise (black points). It is important to highlight that in this experiment a noise does not mean a fault, but it was just used to create a different variation of the input data, compared to the predominant patterns in the training dataset. In addition, the instance which the noise were added were randomly chosen, keeping a minimal distance between each moment, and also each symptom has different moments, to allow each to separately influence the network.

Therefore, Figure 43f presents the result for Fault Detection in this experiment. The percentage of outputs for each outcome of the fault node are 15.5%, 68.6%, and 15.9% for *yes*, *no*, and *warn*, respectively. The Gaussian noise inserted in the lateral error resulted in 42.2%, 8.9%, 11.7%, 18.3%, and 18.9% of *very_large*, *large*, *medium*, *small*, and *very_small*, in this order. While for the remaining residues, the percentages in the same order are, 36.1%, 8.9%, 4.5%, 43.9%, and 6.6% for the *angular residue*; 35%, 11.7%, 15%, 11.7%, and 26.6% for the *curvature residue*; and, 10.6%, 11.1%, 23.3%, 35%, and 20% for the *longitudinal residue*.

Table 10 presents the quantitative results for the ratio of the fault's outcomes with respect to the outcomes of the symptoms nodes. Similar to Case 1, when the output of the fault node is *warn*, the majority of outputs of the symptoms nodes are *medium*, *small*, and *very_small*, which represent 97.313%, 93.93%, 98.011%, and 100%, of *warn* presence for the nodes *lateral residue*, *angular residue*, *longitudinal residue*, and *curvature residue*, in this order. In turn, for

Figure 42 – Control Diagnosis Experimental Results - Case 1

Source: Elaborated by the author.

the *no* output, the predominant symptoms' outputs are *small* and *very_small*, which constitute 99.308%, 100%, 98.224%, and 98.616%, of the occurrence of *no* for the *lateral residue*, *angular*

residue, *longitudinal residue*, and *curvature residue*, respectively.

In addition, for the *yes* output of the fault node, 4.175% and 1.426% occurred when the *lateral residue* symptom outcome were *very_large* and *large*, each of which represents 44.1% and 73.7% of the occurrence of these symptoms' output. On its part, the proportion for the *longitudinal residue* are 11.813% (96.7%) and 56.616% (97.2%) for the same outputs, while for the *angular residue* are 8.147% (36.7%) and 19.043% (100%), and for the *curvature residue* are 1.935% (25%) and 2.24% (100%).

Another result that Table 43 presents is that in cases of *no* output for the fault node, they correspond to 0.577%, 1.77%, and 1.315% of *very_large* output for the symptoms *lateral residue*, *angular residue*, and *curvature residue*, respectively, each of which represents 26.9%, 35.3%, and 75% of the occurrence of that specific output for the symptoms. On the other hand, in all these cases, for each *very_large* output for one symptom, the remaining symptoms' outputs were *small* or *very_small*. In addition, in the cases of the *lateral residue* and *curvature residue* all of these entries were modified by the Gaussian noise, which can cause an uncharacteristic inputs, sometimes even impossible to happen at the same time.

Table 10 – Control Diagnosis Fault Detection Result - Case 2

control fault	lateral residue					angular residue				
	very large	large	medium	small	very small	very large	large	medium	small	very small
yes	4.175%	1.426%	0.102%	31.874%	62.423%	8.147%	19.043%	2.851%	66.293%	3.666%
warn	2.687%	0	3.582%	39.9%	53.831%	6.07%	0	11.841%	71.343%	10.746%
no	0.577%	0.115%	0	38.339%	60.969%	1.776%	0	0	76.217%	22.007%

control fault	longitudinal residue					curvature residue				
	very large	large	medium	small	very small	very large	large	medium	small	very small
yes	11.813%	56.619%	16.599%	8.248%	6.721%	1.935%	2.24%	0.102%	0.509%	95.214%
warn	0.397%	1.592%	88.06%	5.075%	4.876%	0	0	2.587%	0.199%	97.214%
no	0	0	0	46.321%	53.679%	1.315%	0	0.069%	0.554%	98.062%

Source: Elaborated by the author.

7.1.2 GPS Monitor

The *GPS Monitor* is responsible to monitor the GPS sensor, which provides data for the localization component of the autonomous system. Therefore it uses different sources of information that characterizes the faults that the sensor is subject to, such as the PDOP value, the outputs of the RAIM algorithm, number of tracked satellites, the delay of the RTK correction, and the pose variance and residues. All this information is used as evidence of a Hybrid-BN, updating the status of the correspondent symptom node, and finally, the fault node (see section 6.2.2).

Figure 43 – Control Diagnosis Experimental Results - Case 2

Source: Elaborated by the author.

The dataset for learning the CPTs of the discrete nodes and testing was created using the data obtained using the *CaRINA II* in autonomous and manual modes. The total amount of

entries of the dataset is 180604, and it was divided in training and test, with the proportion of 80% and 20%, respectively. The data were labeled according to the criteria that characterize a fault, being:

- *Fault 1*: error in computing the heading of the vehicle;
- *Fault 2*: not enough satellites;
- *Fault 3*: no RTK correction or very large delay since last correction;
- *Fault 4*: fair or poor PDOP;
- *Fault 5*: very large residue of the x, y, and θ components of the measured position;
- *Fault 6*: integrity failed as result of the RAIM algorithm.

Table 11 presents the division of the training and testing dataset, and the number of outputs *yes*, *no*, and *warn* for each. Each instance of the dataset is composed by the tuple present in Equation 7.2, with the inputs evidences, the outputs of each symptom, and the output of the fault node. The Expectation-Maximization was also used for learning the CPTs of the discrete nodes, which includes evidence, symptoms, and the fault node. In addition, similar to the *Control Monitor*, a confusion matrix was used to evaluate the performance of the network with the testing dataset, and also a real experiment case was performed, in which real events of faults are presented.

$$\mathbf{d}_k^{gps} = (n_{satellites}, \Delta age_{rtk}, i_x, i_y, i_\theta, \sigma_x^2, \sigma_y^2, \sigma_\theta^2, PDOP, int_{raim}, herl_{raim}, verl_{raim}, o_{satellites}, o_{rtk}, o_{residue}, o_{variance}, o_{PDOP}, o_{RAIM}, o_{GPS}) \quad (7.2)$$

in which, $n_{satellites}$ is the number of tracked satellites; Δage_{rtk} is the time (sec.) elapsed since the last update of RTK; i_x , i_y , and i_θ , are the innovation values from the EKF; σ_x^2 , σ_y^2 , σ_θ^2 , are the variance of each pose component, measured by the sensor; $PDOP$ is the value of the Position Dilution of Precision; int_{raim} , $herl_{raim}$, $verl_{raim}$, are the results of the RAIM algorithm; $o_{satellites}$, o_{rtk} , $o_{residue}$, $o_{variance}$, o_{PDOP} , and o_{RAIM} , are the output of the symptoms *status number of satellites*, *status RTK correction*, *pose residue*, *pose variance*, *DOP evaluation*, and *RAIM status*, respectively. Finally, o_{GPS} is the output of the fault node.

The confusion matrix created using the testing dataset is presented in Table 12. The average time of inference is approximately 3ms, which is also sufficient for real-time application. Therefore, the accuracy of this Hybrid-BN model is approximately 98.44%, being the recalls equal to 0.9995, 0.9994, 0.9588, for the *yes*, *warn*, and *no*, outputs in this order. All values means the confidence that when the output if a specific outcome, they truly are these values, being 1.0 the maximum confidence. Therefore, the specificity or True Negative Rate (TNR) are 0.9777,

Table 11 – GPS Diagnosis Dataset Division in Training and Test

GPS Fault Outcome	training	test	% training	% test
yes	56291	14042	38.96	38.87
warn	34782	8773	24.07	24.29
no	53411	13305	36.97	36.84
<i>total</i>	144484	36120	100.0	100.0

Source: Elaborated by the author.

0.9978, and 0.9996 for the *yes*, *warn*, and *no* outputs, meaning that when the network model outputs differs from an outcome, it has these confidence values for not truly be that outcome.

Table 12 – GPS Diagnosis Confusion Matrix

Unnormalized Confusion Matrix				Normalized Confusion Matrix							
True label	yes	14035	2	5	True label	yes	1.0	0.0	0.0		
	warn	2	8768	3		True label	warn	0.0	1.0	0.00	
	no	490	58	12757			True label	no	0.04	0.0	0.96
		yes	warn	no					yes	warn	no
Predicted label				Predicted label							

Source: Elaborated by the author.

In the confusion matrix (Tab. 12) the majority of misclassification occurred when the expected output is *no*, in which 3.683% of the predictions were *yes*, and 0.44% were *warn*. In the first case, all the predicted outputs of the *pose residues* were *very_large*, however, the true outputs should be *medium*. In turn, in second case, classifications errors were also due to the outputs of the *pose residues*, in which the predicted outputs were *medium*, instead of *small* and *very_small*.

Moreover, an experimental test (Case 3) in a real scenario was also performed to analysis the performance of the Hybrid-BN for FDD implemented in the *GPS Monitor*. Figure 45 presents the route traveled with *CaRINA II* in manual mode, since only the GPS measurements were stored.

7.1.2.1 Case 3

This experimental test aims to analysis the performance of the FDD developed in the *GPS Monitor* of the *Health Monitoring System* implemented in *CaRINA II*, during the navigation in which real faults occurred. Therefore the data were collected using the vehicle in manual driver mode. Figure 44 presents the path traveled, and similar to Case 1 (see 7.1.1.1) and Case 2 (see 7.1.1.2), Figures 45a, 45b, 45c, 45d, 45e, and 45f show the evidence data and the respective output of its associated symptom node.

The outputs of the symptoms indicate that each symptom perform well distinction between the values of its evidences. For example, the results present in Figure 45a, which concerns to the symptom node *DOP evaluation*, approximates of the division shown in Table 6. Similar behavior occurred with the remaining symptoms nodes, for instance, the classification of the number of satellites (Fig. 45b) which obey the minimum stated in Dutt *et al.* (2009) and Septentrio (2014); the rating of the RTK delay, whereat the threshold of each qualitative assignment were found by analyzing the data obtained in the training.

The analysis of the results of the RAIM algorithm (Fig. 45d) are also in agreement with the expected, since it can differentiate when the test integrity failed or succeeded, including the analysis of the threshold for the HERL and VERL. In turn, the outputs of the *pose variance* and *pose residue* performs the differentiation between the magnitude of the variance or the absolute residue. However, different of the Hybrid-BN for the *Control Monitor*, a single symptom node analysis more than one residue, that is, these symptom nodes measures how large is the value of all its evidence together.

Figure 44 – GPS Diagnosis Experiment - Case 3

Source: Elaborated by the author.

Therefore, the results of the Fault Detection are presented in Figure 44b, in which 29.5%, 48.7%, and 21.8% are *yes*, *no*, *warn*, respectively. Table 13 presents the ration of the outcomes of the fault node with respect to the symptoms nodes. In the cases where *Fault 1* occurred (10.5% of the readings), the outputs of the fault node are 81.3% *yes*, and 18.7% *warn*. The *warn*'s outputs are due to the variance and residue results being a *medium* size, that is, when the fault occurred the innovation of the EKF filter was not much large, and the variance of the components (x , y , and θ) was also not much large, which can happen just after the occurrence of the fault or when the current values for the components is close to the new faulty estimation.

Moreover, in the presence of *Fault 2*, which is when the GPS receiver does not have enough tracked satellites, the output of the fault node is *yes*, which is 6.845% of fault alerts. In

the case of *Fault 3*, the GPS receiver received correction data during the entire route, however there were moments of long delays between the corrections, in this case, 93.5% of *very_bad* delays triggered a fault alert, which correspond to 2.708% of the faults. The remaining *very_bad* outputs were classified in *warn* (5.2%) and *no* (1.3%, which is a single reading).

Table 13 – GPS Diagnosis Fault Detection Result - Case 3

GPS fault	DOP evaluation							RAIM status		
	ideal	excellent	good	moderate	fair	poor	not available	yes	warn	no
yes	58.556%	19.18%	8.951%	8.574%	4.739%	0	0	17.413%	0	82.587%
warn	49.949%	18.946%	19.25%	11.348%	0.507%	0	0	0	0	100%
no	61.167%	23.082%	11.757%	3.881%	0.113%	0	0	0	0	100%

GPS fault	pose residue					pose variance				
	very large	large	medium	small	very small	very large	large	medium	small	very small
yes	73.674%	6.469%	7.334%	6.92%	5.604%	14.404%	6.431%	4.513%	1.617%	73.035%
warn	0	4.255%	94.833%	0.912%	0	0	0.253%	3.951%	5.268%	90.527%
no	0	70.517%	0.091%	9.056%	20.336%	0	0	0.068%	2.27%	97.662%

GPS fault	status RTK correction					status number of satellites		
	good	medium	bad	very bad	no correction	not enough	enough	good
yes	79.391%	13.125%	4.776%	2.708%	0	6.845%	39.865%	53.291%
warn	92.806%	5.015%	1.976%	0.203%	0	0	68.237%	31.763%
no	95.301%	4.653%	0.023%	0.023%	0	0	61.144%	38.856%

Source: Elaborated by the author.

For the *Fault 4* cases, just *fair* output of the symptom *DOP evaluation* is present in the traveled route (1.56% of the readings). Therefore, 89.36% are *yes*, 7.1% *warn*, and 3.54% *no*. The *no* (5 readings) outputs occurred along with outputs considered ideal for each symptom, that is, they did not show any warning or failure signals. On its part, the *warn* outputs occurred along with *medium* output for the *pose residue* symptom, while the remaining symptoms outputs were ideals.

All the *very_large* output of the *pose residue* symptom matches the *yes* outputs of the fault node, which corresponds to *Fault 5*. In turn, the output *large* occurred in 70.517% of the *no* fault, however, the remaining outputs of the symptoms are ideals, being: 93.89% (*good*) and 6.11% (*medium*) for the *status RTK correction*; 100% (*no*) for the *RAIM status*; 61% (*ideal*), 25% (*excellent*), and 14% (*good*) for the *DOP evaluation*; 41.1% (*good*) and 58.9% (*enough*) for the *status number of satellites*; and, 99% (*very_small*) and 1.0% (*small*) for the *pose variance*. Besides that, the average mean of the absolute residues are 1.1m, 1.05m, and 0.11° for the *x*, *y*, and *θ* components, respectively, with standard deviation of 0.51m, 0.57m, and 0.18°, in the same order. Finally, in the last fault case, *Fault 6*, when there is an integrity failed as result of the RAIM algorithm, all the fault's outputs are *yes*.

Figure 45 – GPS Diagnosis Experimental Results - Case 3

(a) DOP evaluation

(b) status number of satellites

(c) status RTK correction

(d) RAIM status

(e) pose variance

(f) pose residue

Source: Elaborated by the author.

7.2 Software Architecture

The analysis of the software architecture presented in Chapter 5 were conducted by an autonomous navigation of a route showed in Figure 46, which were performed at the campus 2 of the University of São Paulo - São Carlos, Brazil. The topological graph of this map is shown in Figure 35, and this route is composed by a lane-chage maneuver, two roundabouts, and three speed bump signs. Figure 47a presents the speed profile and the true speed performed by the vehicle, while Figure 47b shows the reference and executed steering angle.

Figure 46 – University of São Paulo - Campus 2

Source: Elaborated by the author.

Figure 47 – Speed and Curvature of the Route

Source: Elaborated by the author.

The speed reference profile in Figure 47a shows the three moments in which the vehicle is approaching the speed bump sign, whereat the maximum allowed speed is 3.0m/s. This limitation were inserted on-line during the navigation by the behavior-tree when the vehicle is approximating the traffic sign. In addition, the Mean-Squared Error (MSE) between the reference speed and the performed speed is 0.564m/s, which indicates that the vehicle navigates close to the desired speed. However, in Figure 47a it is notable that most of the time the true speed is above the speed limit of the roads. Figure 48a shows the histogram of the speed error of the overall route, and also the error of the readings above the speed limit.

The Mean-Squared Error (MSE) of the reading above the speed limit is 0.258m/s, which is very small, thus it not lead to a traffic law infraction. These error occurred during the adjustment to the speed reference performed by the PI (Proportional-Integral) controller in the *Control Layer*. Even though the MSE is very small, this behavior is not desired, therefore different approaches should be taken to prevent this behavior remaining present in the architecture.

One of the approaches that should be adopted is the use of a virtual speed limit below the true value, and the modification of the controller gain constants to decrease the oscillation in reaching the reference speed. Finally, the MSE between the steering angle reference and the performed steering angle is 3.2×10^{-5} rad, indicating good lateral controller and steering wheel actuator compliance. Figure 48b shows the histogram of the steering angle error. Besides that, there were no errors detected by the Health Monitoring System.

Figure 48 – Histogram of the Speed and Curvature Error

Source: Elaborated by the author.

7.3 Final Considerations

This chapter presents the experimental results of this Master dissertation, including the analysis of the *Control Monitor* and *GPS Monitor* of the *Health Monitoring System* (see chapter 6), and the overall evaluation of the software architecture (see chapter 5). To accomplish that, four experimentation were performed, in which the first two experiments (Case 1 and Case 2) analyze the Hybrid Bayesian Network with Residue Evidence for the controllers in the *Control Monitor*, in turn, the third experiment (Case 3) evaluates the Hybrid-BN for the GPS sensor, and, finally, the last experiment examines the overall architecture by means of a route navigated autonomously.

The first two experiments showed that the Hybrid-BN for the controllers obtained positive performance, since in both tests the output *no* occurred in the vast majority of cases in the presence of the symptoms outputs *small* and *very_small*, and *warn* with *medium*, *small* and *very_small*. In turn, the outputs *yes* occurred with *large* and *very_large* residues. However, in Case 2, there were outputs *no* in the presence of *very_large*, although in these cases the presence of this output was accompanied by *small* and *very_small* for the remaining symptoms' outputs, in addition to being modified by Gaussian noise.

Nevertheless, a *very_large* output for any of the symptoms should be considered a fault according to the specifications of fault to this monitor. An alternative to increase the performance of this network in the presence of noise would be to include noises in the training data, as it is done in other machine learning approaches.

The third experiment (Case 3) evaluates the Hybrid-BN for the GPS sensor, which has residue evidence and other source of information that defines the faults signatures. This network was examined in the presence of 6 different faults. For all faults the outputs of the fault node

were always close to the maximum of correct fault alert, being 81.3% (*Fault 1*), 100% (*Fault 2*), 93.5% (*Fault 3*), 89.36% (*Fault 4*), 100% (*Fault 5*), and 100% (*Fault 6*).

However, there is an unexpected behavior for this network, the 70.517% of *no* output in the presence of *large* residue output. The *Fault 5* is characterized by a *very_large* residue, therefore, the *large* output should trigger at least a *warn*, even though in the situations found in the analysis of Case 3, the outputs of the remaining symptoms are considered ideal.

Finally, the last experimentation evaluates the performance of the software architecture in a route navigated autonomously. The Mean-Squared Error (MSE) for the steering angle in the performed route is 3.2×10^{-5} rad, which indicates the compliance of the lateral controller and the steering wheel actuator. In addition, the MSE for the longitudinal controller is 0.564m/s, which can be considered small, thus it also proves the efficiency of the components. On the other hand, the PI controller demonstrated oscillation to achieve the reference speed, which led the vehicle to exceed the limit speed in the average (MSE) of 0.258m/s. This behavior should be corrected by adjusting the controller gains, and using a speed limit value less than the nominal value.

CONCLUSION

Autonomous Vehicles are intelligent and robotic vehicles that use information from the external environment and their own state to navigate and interact with traffic, complying with legislation and dealing with unexpected situations, without the need for human supervision or control. Thus, it is expected that, once they are present on all roads, a significant improvement in urban mobility will occur, making it more effective, more accessible and safer.

To be able to meet so many expectations, the autonomous system is composed of several components that interact with each other. Because of this, a software architecture facilitates understanding of the system, as well as allowing decisions to be made before they are implemented. From the point of view of the system, it can be divided into some basic tasks, such as: sensing; perception; decision-making and planning; control; and, system management. Therefore, the analysis of the software architecture of *Tongji* (section 3.2.1), *AI* (section 3.2.2), *BerthaOne* (section 3.2.3), *IARA* (section 3.2.4), and *CMU* (section 3.2.5), and the approaches for navigation pipeline (section 3.3), were the bases for the development of the *CaRINA II* software architecture proposal described in Chapter 5.

The proposed architecture also benefits from work already done in the laboratory, among them the works of Santos (2015) , Silva (2018) and Rosero and Osório (2017). In addition, as contribution of this Master dissertation, were included in the architecture a new road network model (map), based on the work of Silva and Grassi (2018) (section 5.2.5); a hybrid navigation architecture, based on a hierarchical organization, but with components of the *Path Planning* and *Behavior Planning* layers sharing the same hierarchical level (section 5.2.7); behaviors implemented using Behavior-Trees (section 5.2.7.4); and a Health Monitoring System using Hybrid-Bayesian Network with Residue Evidence for the *Control Monitor* and *GPS Monitor* (chapter 6).

The experimental results presented in Chapter 7 demonstrate good performance of the Hybrid-BNs and also the overall software architecture. In the Hybrid-BNs all symptom nodes

were able to distinguish between the values of their evidence in corresponding classes, and the classifications between the presence of faults (*yes*), the absence of faults (*no*), and the intermediate state (*warn*) are in compliance with the requirements raised for the construction of the networks.

However, based on the analysis of the experimental tests and performance observations made during the project of the architecture, some improvements need to be made. The first of all, is the localization based on the GPS sensor. As mentioned in the evaluation of Case 3, faults alert were triggered in 29.5% of the reading, which is a considerable amount, therefore, only GPS-based localization is not enough, thus fusion-based approaches should be implemented, and also adaptive techniques, such as the Adaptive Extended Kalman Filter, which, for example, can take the diagnosis result to avoid propagation of the sensors errors for the remaining pipeline of the navigation.

Another point that needs improvements is the road network model, that with the addition of more information of the road, including traffic rules, should allow the implementation of more behaviors, increasing the autonomy of the autonomous system. Moreover, integrating software architecture into a simulation platform, such as the CARLA simulator ([CARLA, 2018](#)), is essential for work continuity, as more complex behaviors need to be exhaustively tested before they can be performed in real environments. In addition, testing during the development of the other architectural components is important, and the logistics to do them in real environments is a factor that makes it difficult to add new components to the autonomous system.

BIBLIOGRAPHY

AEBERHARD, M.; KÜHBECK, T.; SEIDL, B.; FRIENDL, M.; THOMAS, J.; SCHEICKL, O. **AUTOMATED DRIVING WITH ROS AT BMW**. 2015. ROSCon 2015. Accessed: 30 jul. 2018. Available: <https://roscon.ros.org/2015/presentations/ROSCon-Automated-Driving.pdf>. Citation on page 65.

AMIN, M. T.; IMTIAZ, S.; KHAN, F. Process system fault detection and diagnosis using a hybrid technique. **Chemical Engineering Science**, Elsevier, 2018. Accessed: 25 jul. 2018. Available: <https://www.sciencedirect.com/science/article/pii/S0009250918303403>. Citations on pages 59 and 94.

ANDERSON, J. M.; NIDHI, K.; STANLEY, K. D.; SORENSEN, P.; SAMARAS, C.; OLUWATOLA, O. A. **Autonomous vehicle technology: A guide for policymakers**. [S.l.]: Rand Corporation, 2014. Accessed: 17 jul. 2018. Citations on pages 27, 28, 29, and 30.

BAGLOEE, S. A.; TAVANA, M.; ASADI, M.; OLIVER, T. Autonomous vehicles: challenges, opportunities, and future implications for transportation policies. **Journal of Modern Transportation**, Springer, v. 24, n. 4, p. 284–303, 2016. Accessed: 19 jul. 2018. Available: <https://link.springer.com/article/10.1007/s40534-016-0117-3>. Citation on page 28.

BASS, L.; CLEMENTS, P.; KAZMAN, R. **Software architecture in practice**. [S.l.]: Addison-Wesley Professional, 2003. Citation on page 35.

BAST, H.; DELLING, D.; GOLDBERG, A.; MÜLLER-HANNEMANN, M.; PAJOR, T.; SANDERS, P.; WAGNER, D.; WERNECK, R. F. Route planning in transportation networks. In: **Algorithm engineering**. Springer, 2016. p. 19–80. Accessed: 31 may 2019. Available: <https://arxiv.org/pdf/1504.05140.pdf>. Citation on page 45.

BEHERE, S. **Architecture support for automobile autonomy: A state of the art survey**. 2012. Accessed: 25 jun. 2018. Available: <http://www.diva-portal.org/smash/record.jsf?pid=diva2%3A882434&dsid=-6100>. Citation on page 36.

BELBACHIR, A.; BOUTTEAU, R.; MERRIAUX, P.; BLOSSEVILLE, J.-M.; SAVATIER, X. From autonomous robotics toward autonomous cars. In: IEEE. **2013 IEEE Intelligent Vehicles Symposium (IV)**. 2013. p. 1362–1367. Accessed: 18. jun. 2019. Available: <https://ieeexplore.ieee.org/abstract/document/6629656>. Citation on page 51.

BENDER, P.; ZIEGLER, J.; STILLER, C. Lanelets: Efficient map representation for autonomous driving. In: IEEE. **2014 IEEE Intelligent Vehicles Symposium Proceedings**. 2014. p. 420–425. Accessed: 17 jun. 2019. Available: <https://ieeexplore.ieee.org/abstract/document/6856487>. Citations on pages 51 and 53.

BENEDETTINI, O.; BAINES, T.; LIGHTFOOT, H.; GREENOUGH, R. State-of-the-art in integrated vehicle health management. **Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering**, SAGE Publications Sage UK: London, England, v. 223, n. 2, p. 157–170, 2009. Accessed: 22 jul. 2018. Available: <http://journals.sagepub.com/doi/abs/10.1243/09544100JAERO446>. Citation on page 105.

BIJJAHALI, S.; RAMASAMY, S.; SABATINI, R. A gnss integrity augmentation system for airport ground vehicle operations. 2017. Accessed: 06 jul. 2019. Available: https://www.researchgate.net/profile/Roberto_Sabatini2/publication/314442175_A_GNSS_Integrity_Augmentation_System_for_Airport_Ground_Vehicle_Operations/links/58c26a2e45851538eb7cee21/A-GNSS-Integrity-Augmentation-System-for-Airport-Ground-Vehicle-Operations.pdf. Citation on page 98.

BIMBRAW, K. Autonomous cars: Past, present and future a review of the developments in the last century, the present scenario and the expected future of autonomous vehicle technology. In: IEEE. **Informatics in Control, Automation and Robotics (ICINCO), 2015 12th International Conference on**. 2015. v. 1, p. 191–198. Accessed: 20 jun. 2018. Available: <https://ieeexplore.ieee.org/abstract/document/7350466/>. Citation on page 27.

BRAID, D.; BROGGI, A.; SCHMIEDEL, G. The terramax autonomous vehicle. **Journal of Field Robotics**, Wiley Online Library, v. 23, n. 9, p. 693–708, 2006. Accessed: 20 jun. 2019. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.20140>. Citation on page 74.

BROGI, M.; TROMBONI, P. A tool for real-time gnss performance analysis in air traffic management. **Journal of Aerospace Technology and Management**, SciELO Brasil, v. 7, n. 4, p. 504–513, 2015. Accessed: 07 jul. 2019. Available: http://www.scielo.br/scielo.php?pid=S2175-91462015000400504&script=sci_arttext. Citation on page 101.

BRUMMELEN, J. V.; O'BRIEN, M.; GRUYER, D.; NAJJARAN, H. Autonomous vehicle perception: The technology of today and tomorrow. **Transportation research part C: emerging technologies**, Elsevier, 2018. Accessed: 17 jul. 2018. Available: <https://www.sciencedirect.com/science/article/pii/S0968090X18302134>. Citations on pages 30, 31, 32, and 51.

BUEHLER, M.; IAGNEMMA, K.; SINGH, S. **The 2005 DARPA grand challenge: the great robot race**. [S.l.]: Springer, 2007. Citation on page 54.

_____. **The DARPA urban challenge: autonomous vehicles in city traffic**. [S.l.]: springer, 2009. Citations on pages 28 and 54.

BUSCHMANN, F.; MEUNIER, R.; ROHNERT, H.; SOMMERLAD, P.; STAL, M. **A system of patterns: Pattern-oriented software architecture**. [S.l.]: Wiley New York, 1996. Citation on page 66.

CAI, B.; LIU, H.; XIE, M. A real-time fault diagnosis methodology of complex systems using object-oriented bayesian networks. **Mechanical Systems and Signal Processing**, Elsevier, v. 80, p. 31–44, 2016. Accessed: 27 jul. 2018. Available: <https://www.sciencedirect.com/science/article/pii/S0888327016300607>. Citation on page 63.

CAI, B.; LIU, Y.; XIE, M. A dynamic-bayesian-network-based fault diagnosis methodology considering transient and intermittent faults. **IEEE Transactions on Automation Science and Engineering**, IEEE, v. 14, n. 1, p. 276–285, 2017. Accessed: 25 jul. 2018. Available: <https://ieeexplore.ieee.org/abstract/document/7495018/>. Citations on pages 58, 59, 61, and 63.

CARDOSO, V. B. **A Model-Predictive Motion Planner for the IARA Autonomous Car**. Master's Thesis (Master's Thesis) — Universidade Federal do Espírito Santo, 2017. Accessed: 12 may 2019. Available: <http://repositorio.ufes.br/handle/10/9855>. Citations on pages 42 and 43.

CARLA. **CARLA 0.9.2: Upgraded ROS-bridge, Traffic scenario engine and a new Agent class to perform driving navigation from client side!** [S.l.], 2018. Accessed: 15 jun. 2019. Available: <http://carla.org/2018/12/24/release-0.9.2/>. Citations on pages 56 and 126.

CHAMPANDARD, A. J.; DAWE, M.; HERNANDEZ-CERPA, D. **Behavior Trees: Three Ways of Cultivating Game AI**. 2019. Available: http://twvideo01.ubm-us.net/o1/vault/gdc10/slides/ChampandardDaweHernandezCerpa_BehaviorTrees.pdf. Citation on page 55.

CHAVAN, P.; MURUGAN, M.; CHAVAN, P. A Review on Software Architecture Styles with Layered Robotic Software Architecture. In: **2015 International Conference on Computing Communication Control and Automation**. IEEE, 2015. p. 827–831. ISBN 978-1-4799-6892-3. Accessed: 20 jun. 2018. Available: <http://ieeexplore.ieee.org/document/7155963/>. Citation on page 36.

CHEN, S. H.; POLLINO, C. A. Good practice in bayesian network modelling. **Environmental Modelling & Software**, Elsevier, v. 37, p. 134–145, 2012. Accessed: 09 jun. 2019. Available: <https://www.sciencedirect.com/science/article/pii/S1364815212001041>. Citation on page 63.

COLLEDANCHISE, M. **Behavior trees in robotics**. Phd Thesis (PhD Thesis) — KTH Royal Institute of Technology, 2017. Accessed: 14 jun. 2019. Available: <http://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1078940&dswid=5755>. Citations on pages 54, 55, and 56.

COLLEDANCHISE, M.; MARZINOTTO, A.; DIMAROGONAS, D. V.; ÖGREN, P. Adaptive fault tolerant execution of multi-robot missions using behavior trees. **arXiv preprint arXiv:1502.02960**, 2015. Accessed: 14 jun. 2019. Available: <https://arxiv.org/abs/1502.02960>. Citations on pages 55, 56, and 57.

COLLEDANCHISE, M.; MARZINOTTO, A.; ÖGREN, P. Performance analysis of stochastic behavior trees. In: IEEE. **2014 IEEE International Conference on Robotics and Automation (ICRA)**. 2014. p. 3265–3272. Accessed: 15 jun 2019. Available: <https://ieeexplore.ieee.org/abstract/document/6907328/>. Citation on page 57.

COLLEDANCHISE, M.; ÖGREN, P. **Behavior trees in robotics and AI: an introduction**. [s.n.], 2017. Accessed: 12 jun. 2019. Available: <https://arxiv.org/abs/1709.00084>. Citations on pages 54 and 56.

COLOMBO, O. L. Real-time, wide-area, precise kinematic positioning using data from internet ntrip streams. **Proceedings of ION-GNSS-2008, Institute of Navigation, Savannah, Georgia**, p. 327–337, 2008. Accessed: 08 jul. 2019. Available: <https://pdfs.semanticscholar.org/cc4f/68fa759080e77378cbe497bd8e04cd3d34af.pdf>. Citation on page 102.

CRESTANI, D.; GODARY-DEJEAN, K.; LAPIERRE, L. Enhancing fault tolerance of autonomous mobile robots. **Robotics and Autonomous Systems**, Elsevier, v. 68, p. 140–155, 2015. Accessed: 25 jul. 2018. Available: <https://www.sciencedirect.com/science/article/pii/S0921889014003157>. Citation on page 58.

DAI, X.; GAO, Z. From model, signal to knowledge: A data-driven perspective of fault detection and diagnosis. **IEEE Transactions on Industrial Informatics**, IEEE, v. 9, n. 4, p. 2226–2238, 2013. Accessed: 26 jul. 2018. Available: <https://ieeexplore.ieee.org/abstract/document/6423903/>. Citation on page 60.

DARWICHE, A. **Modeling and reasoning with Bayesian networks**. [S.l.]: Cambridge University Press, 2009. Citations on pages 64 and 65.

DIJKSTRA, E. W. A note on two problems in connexion with graphs. **Numerische mathematik**, Springer, v. 1, n. 1, p. 269–271, 1959. Accessed: 25 jun. 2019. Available: <https://link.springer.com/article/10.1007%2F01386390?LI=true>. Citation on page 84.

DUPUIS, M.; GREZLIKOWSKI, H. **OpenDRIVE: An open Standard for the Description of Roads in Driving Simulations**. VIRES Simulationstechnologie GmbH, 2006. Accessed: 17 jun. 2019. Available: <http://www.opendrive.org/docs/OpenDRIVEatDSC20061004B.pdf>. Citation on page 53.

DUPUIS, M.; HEKELE, E.; BIEHN, A. **Format Specification, Rev. 1.5**. [S.l.], 2019. Accessed: 18 jun. 2019. Available: <http://www.opendrive.org/docs/OpenDRIVEFormatSpecRev1.5M.pdf>. Citations on pages 52 and 90.

DUPUIS, M.; STROBL, M.; GREZLIKOWSKI, H. Opendrive 2010 and beyond—status and future of the de facto standard for the description of road networks. In: **Proc. of the Driving Simulation Conference Europe**. [s.n.], 2010. p. 231–242. Accessed: 17 jun. 2019. Available: https://www.ifsttar.fr/fileadmin/user_upload/editions/inrets/Actes/Actes_INRETS_A126.pdf#page=233. Citation on page 52.

DUTT, V. S. I.; RAO, G. S. B.; RANI, S. S.; BABU, S. R.; GOSWAMI, R.; KUMARI, C. U. Investigation of gdop for precise user position computation with all satellites in view and optimum four satellite configurations. **J. Ind. Geophys. Union**, v. 13, n. 3, p. 139–148, 2009. Accessed: 05 jul. 2019. Available: <https://pdfs.semanticscholar.org/c755/45f22dabde0325290b3e3076a97bc81daacb.pdf>. Citations on pages 98, 99, 100, 101, and 117.

D'ANGELO, M. F.; PALHARES, R. M.; COSME, L. B.; AGUIAR, L. A.; FONSECA, F. S.; CAMINHAS, W. M. Fault detection in dynamic systems by a fuzzy/bayesian network formulation. **Applied Soft Computing**, Elsevier, v. 21, p. 647–653, 2014. Accessed: 01 jul. 2019. Available: <https://www.sciencedirect.com/science/article/pii/S1568494614001707>. Citation on page 94.

ESPERON-MIGUEZ, M.; JOHN, P.; JENNIONS, I. K. A review of integrated vehicle health management tools for legacy platforms: challenges and opportunities. **Progress in Aerospace Sciences**, Elsevier, v. 56, p. 19–34, 2013. Accessed: 30 jun. 2019. Available: <https://www.sciencedirect.com/science/article/pii/S0376042112000541>. Citation on page 91.

EUGSTER, P. T.; FELBER, P. A.; GUERRAOU, R.; KERMARREC, A.-M. The many faces of publish/subscribe. **ACM computing surveys (CSUR)**, ACM, v. 35, n. 2, p. 114–131, 2003. Accessed: 11 jul. 2018. Available: <https://dl.acm.org/citation.cfm?id=857078>. Citations on pages 65 and 66.

EYKEREN, L. V.; CHU, Q.; MULDER, J. Sensor fault detection and isolation using adaptive extended kalman filter. **IFAC Proceedings Volumes**, Elsevier, v. 45, n. 20, p. 1155–1160, 2012. Accessed: 01 jul. 2019. Available: <https://www.sciencedirect.com/science/article/pii/S1474667016349102>. Citation on page 103.

FAGNANT, D. J.; KOCKELMAN, K. Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations. **Transportation Research Part A: Policy and Practice**, Elsevier, v. 77, p. 167–181, 2015. Accessed: 16 jul. 2018. Available: <https://www.sciencedirect.com/science/article/pii/S0965856415000804>. Citations on pages 27 and 28.

FERNANDES, L. C.; SOUZA, J. R.; PESSIN, G.; SHINZATO, P. Y.; SALES, D.; MENDES, C.; PRADO, M.; KLASER, R.; MAGALHAES, A. C.; HATA, A. *et al.* Carina intelligent robotic car: architectural design and applications. **Journal of Systems Architecture**, Elsevier, v. 60, n. 4, p. 372–392, 2014. Accessed: 30 jul. 2018. Available: <<https://www.sciencedirect.com/science/article/pii/S1383762113002841>>. Citation on page 69.

FIELDING, R. T.; TAYLOR, R. N. **Architectural styles and the design of network-based software architectures**. University of California, Irvine Doctoral dissertation, 2000. Accessed: 11 jul. 2018. Available: <<http://jpkc.fudan.edu.cn/picture/article/216/35/4b/22598d594e3d93239700ce79bce1/7ed3ec2a-03c2-49cb-8bf8-5a90ea42f523.pdf>>. Citation on page 36.

FILHO, C. M.; WOLF, D. F.; GRASSI, V.; OSÓRIO, F. S. Longitudinal and lateral control for autonomous ground vehicles. In: IEEE. **2014 IEEE Intelligent Vehicles Symposium Proceedings**. [S.l.], 2014. p. 588–593. Accessed in: 21 jun. 2019. Citation on page 76.

FLOROS, G.; ZANDER, B. V. D.; LEIBE, B. Openstreetslam: Global vehicle localization using openstreetmaps. In: IEEE. **2013 IEEE International Conference on Robotics and Automation**. 2013. p. 1054–1059. Accessed: 17 jun. 2019. Available: <<https://ieeexplore.ieee.org/abstract/document/6630703>>. Citation on page 51.

FOX, C. Bayesian inference. In: _____. **Data Science for Transport: A Self-Study Guide with Computer Exercises**. Cham: Springer International Publishing, 2018. p. 75–92. ISBN 978-3-319-72953-4. Accessed: 27 jul. 2018. Available: <https://doi.org/10.1007/978-3-319-72953-4_6>. Citation on page 62.

FURDA, A.; VLACIC, L. Multiple criteria-based real-time decision making by autonomous city vehicles. 2010. Accessed: 19 jul. 2018. Available: <<http://eprints.qut.edu.au/76257/>>. Citation on page 31.

GAO, Z.; CECATI, C.; DING, S. X. A survey of fault diagnosis and fault-tolerant techniques part ii: Fault diagnosis with knowledge-based and hybrid/active approaches. **IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS**, v. 62, n. 6, p. 3768 – 3774, jun 2015. Accessed: 26 jul. 2018. Available: <<https://ieeexplore.ieee.org/document/7076586>>. Citation on page 61.

_____. A survey of fault diagnosis and fault-tolerant techniques—part i: Fault diagnosis with model-based and signal-based approaches. **IEEE Transactions on Industrial Electronics**, IEEE, v. 62, n. 6, p. 3757–3767, 2015. Accessed: 25 jul. 2018. Available: <<https://ieeexplore.ieee.org/abstract/document/7069265>>. Citations on pages 58, 59, and 60.

GAT, E.; BONNASSO, R. P. On three-layer architectures. **Artificial intelligence and mobile robots**, AAAI Press Menlo Park, CA, v. 195, p. 210, 1998. Accessed: 28 jul 2018. Available: <<http://flownet.com/ron/papers/tla.pdf>>. Citation on page 36.

GAWTHROP, P. J.; BEVAN, G. P. Bond-graph modeling. **IEEE control systems**, IEEE, v. 27, n. 2, p. 24–45, 2007. Accessed: 26 jul. 2018. Available: <<https://ieeexplore.ieee.org/abstract/document/4140745>>. Citation on page 60.

GIMENES, I. M. d. S.; HUZITA, E. H. M. **Desenvolvimento baseado em componentes: conceitos e técnicas**. [S.l.: s.n.], 2005. Citations on pages 35 and 36.

GOMES, I. P.; BRUNO, D. R.; OSÓRIO, F. S.; WOLF, D. F. Diagnostic analysis for an autonomous truck using multiple attribute decision making. jul 2018. Não publicado. Available: <https://ieeexplore.ieee.org/abstract/document/8588565>. Citation on page 92.

GUANETTI, J.; KIM, Y.; BORRELLI, F. Control of connected and automated vehicles: State of the art and future challenges. **Annual Reviews in Control**, Elsevier, 2018. Accessed: 17 jul. 2018. Available: <https://www.sciencedirect.com/science/article/pii/S1367578818300336>. Citation on page 33.

GUSIKHIN, O.; FILEV, D.; RYCHTYCKYJ, N. Intelligent vehicle systems: applications and new trends. In: **Informatics in Control Automation and Robotics**. [S.l.]: Springer, 2008. p. 3–14. Citations on pages 32 and 33.

HAKLAY, M.; WEBER, P. Openstreetmap: User-generated street maps. **IEEE Pervasive Computing**, Ieee, v. 7, n. 4, p. 12–18, 2008. Accessed: 17 jun. 2019. Available: <https://ieeexplore.ieee.org/abstract/document/4653466>. Citation on page 51.

HAWKINS, A. J. **Ford is bringing its self-driving cars to Miami**. 2018. Available: <https://www.theverge.com/2018/2/27/17055616/ford-self-driving-car-miami-test-av>. Citation on page 28.

HENTSCHEL, M.; WAGNER, B. Autonomous robot navigation based on openstreetmap geo-data. In: IEEE. **Intelligent Transportation Systems (ITSC), 2010 13th International IEEE Conference on**. 2010. p. 1645–1650. Accessed: 30 jul. 2018. Available: <https://ieeexplore.ieee.org/abstract/document/5625092>. Citation on page 51.

HEWITSON, S. Gnss receiver autonomous integrity monitoring: A separability analysis. **ION GPS/GNSS 2003**, p. 9–12, 2003. Accessed: 05 jul. 2019. Available: https://www.researchgate.net/profile/Steve_Hewitson/publication/250220624_GNSS_Receiver_Autonomous_Integrity_Monitoring_A_Separability_Analysis/links/551a8b690cf2f51a6fea5f64.pdf. Citation on page 100.

HEWITSON, S.; LEE, H. K.; WANG, J. Localizability analysis for gps/galileo receiver autonomous integrity monitoring. **The journal of Navigation**, Cambridge University Press, v. 57, n. 2, p. 245–259, 2004. Accessed: 05 jul. 2019. Available: <https://www.cambridge.org/core/journals/journal-of-navigation/article/localizability-analysis-for-gpsgalileo-receiver-autonomous-integrity-monitoring/A6583A2F0CBBD51A7708327E107A2F77>. Citation on page 100.

HOSSIN, M.; SULAIMAN, M. A review on evaluation metrics for data classification evaluations. **International Journal of Data Mining & Knowledge Management Process**, Academy & Industry Research Collaboration Center (AIRCC), v. 5, n. 2, p. 1, 2015. Accessed 17 jul. 2019. Available: https://s3.amazonaws.com/academia.edu.documents/37219940/5215ijdkp01.pdf?response-content-disposition=inline%3B%20filename%3DA_REVIEW_ON_EVALUATION_METRICS_FOR_DATA.pdf&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWOWYYGZ2Y53UL3A%2F20190718%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20190718T014713Z&X-Amz-Expires=3600&X-Amz-SignedHeaders=host&X-Amz-Signature=9ae62260b4c5d7ee33766f65441ed7981fc08d40ce0cbf4ffbf231ef2fedb5fe. Citation on page 107.

HUANG, Y.; MCMURRAN, R.; DHADYALLA, G.; JONES, R. P. Probability based vehicle fault diagnosis: Bayesian network method. **Journal of Intelligent Manufacturing**, Springer, v. 19, n. 3, p. 301–311, 2008. Accessed: 27 jul. 2018. Available: <https://link.springer.com/article/10.1007/s10845-008-0083-7>. Citation on page 62.

IAMSUMANG, C.; MOSLEH, A.; MODARRES, M. Monitoring and learning algorithms for dynamic hybrid bayesian network in on-line system health management applications. **Reliability Engineering & System Safety**, Elsevier, v. 178, p. 118–129, 2018. Accessed: 07 jun. 2019. Available: <https://www.sciencedirect.com/science/article/pii/S0951832018303417>. Citation on page 63.

Jl, X. The impact of self-driving cars on existing transportation networks. In: AIP PUBLISHING. **AIP Conference Proceedings**. 2018. v. 1955, n. 1, p. 040142. Accessed: 17 jul. 2018. Available: <https://aip.scitation.org/doi/abs/10.1063/1.5033806>. Citation on page 29.

JO, K.; CHU, K.; SUNWOO, M. Interacting multiple model filter-based sensor fusion of gps with in-vehicle sensors for real-time vehicle positioning. **IEEE Transactions on Intelligent Transportation Systems**, IEEE, v. 13, n. 1, p. 329–343, 2012. Accessed: 12 may 2019. Available: <https://ieeexplore.ieee.org/document/6095631>. Citation on page 39.

_____. Gps-bias correction for precise localization of autonomous vehicles. In: IEEE. **2013 IEEE Intelligent Vehicles Symposium (IV)**. 2013. p. 636–641. Accessed: 12 may 2019. Available: <https://ieeexplore.ieee.org/document/6629538>. Citation on page 39.

JO, K.; KIM, J.; KIM, D.; JANG, C.; SUNWOO, M. Development of autonomous car - part ii: A case study on the implementation of an autonomous driving system based on distributed architecture. **IEEE Transactions on Industrial Electronics**, IEEE, v. 62, n. 8, p. 5119–5132, 2015. Accessed: 6 jul. 2018. Available: <https://ieeexplore.ieee.org/abstract/document/7056521/>. Citations on pages 31, 39, 40, 72, and 74.

JOHNSON, L.; FITZSIMMONS, M. **Uber self-driving cars: everything you need to know**. 2018. Accessed: 17 jul. 2018. Available: <https://www.techradar.com/news/uber-self-driving-cars>. Citation on page 28.

JOSEPH, L. **Mastering ROS for robotics programming**. [S.l.]: Packt Publishing Ltd, 2015. Citations on pages 65, 66, and 67.

JUDALET, V.; GLASER, S.; GRUYER, D.; MAMMAR, S. Fault detection and isolation via the interacting multiple model approach applied to drive-by-wire vehicles. **Sensors**, Multidisciplinary Digital Publishing Institute, v. 18, n. 7, p. 2332, 2018. Accessed: 01 jul. 2019. Available: <https://www.mdpi.com/1424-8220/18/7/2332>. Citation on page 60.

KAPLAN, E.; HEGARTY, C. **Understanding GPS: principles and applications**. Artech house, 2005. Accessed: 05 jul. 2019. Available: http://d1.amobbs.com/bbs_upload782111/files_33/ourdev_584835O21W59.pdf. Citation on page 102.

KATO, S.; TAKEUCHI, E.; ISHIGURO, Y.; NINOMIYA, Y.; TAKEDA, K.; HAMADA, T. An open approach to autonomous vehicles. **IEEE Micro**, IEEE, v. 35, n. 6, p. 60–68, 2015. Accessed: 17 jul. 2018. Available: <https://ieeexplore.ieee.org/abstract/document/7368032/>. Citations on pages 28 and 65.

KATRAKAZAS, C.; QUDDUS, M.; CHEN, W.-H.; DEKA, L. Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions. **Transportation Research Part C: Emerging Technologies**, Elsevier, v. 60, p. 416–442, 2015. Access: 1 jun. 2019. Available: <https://www.sciencedirect.com/science/article/pii/S0968090X15003447>. Citations on pages 45 and 54.

KISTAMGARI, R. R.; NAIK, D. S.; MUSUNURI, P.; SRIKANTH, S. A critical review on autonomous vehicle. 2017. Accessed: 16 jul. 2018. Available: <http://ijsrset.com/paper/2973.pdf>. Citation on page 28.

KOO, J.; KWAC, J.; JU, W.; STEINERT, M.; LEIFER, L.; NASS, C. Why did my car just do that? explaining semi-autonomous driving actions to improve driver understanding, trust, and performance. **International Journal on Interactive Design and Manufacturing (IJIDeM)**, Springer, v. 9, n. 4, p. 269–275, 2015. Accessed: 20 jul. 2018. Available: <https://link.springer.com/article/10.1007/s12008-014-0227-2>. Citation on page 32.

KOOPMAN, P.; WAGNER, M. Autonomous vehicle safety: An interdisciplinary challenge. **IEEE Intelligent Transportation Systems Magazine**, IEEE, v. 9, n. 1, p. 90–96, 2017. Accessed: 19 jul. 2018. Available: <https://ieeexplore.ieee.org/abstract/document/7823109/>. Citation on page 32.

KORB, K. B.; NICHOLSON, A. E. **Bayesian artificial intelligence**. [S.l.]: CRC press, 2010. Citation on page 64.

KORTENKAMP, D.; SIMMONS, R. Robotic systems architectures and programming. In: **Springer Handbook of Robotics**. Springer, 2008. p. 187–206. Accessed: 12 jul. 2018. Available: https://link.springer.com/referenceworkentry/10.1007%2F978-3-540-30301-5_9. Citation on page 36.

KUESTENMACHER, A.; PLÖGER, P. G. Model-based fault diagnosis techniques for mobile robots. **IFAC-PapersOnLine**, Elsevier, v. 49, n. 15, p. 50–56, 2016. Accessed: 26 jul. 2018. Available: <https://www.sciencedirect.com/science/article/pii/S2405896316308849>. Citation on page 60.

KUUTTI, S.; FALLAH, S.; KATSAROS, K.; DIANATI, M.; MCCULLOUGH, F.; MOUZAKITIS, A. A survey of the state-of-the-art localization techniques and their potentials for autonomous vehicle applications. **IEEE Internet of Things Journal**, IEEE, v. 5, n. 2, p. 829–846, 2018. Accessed: 18 jul. 2018. Available: <https://ieeexplore.ieee.org/abstract/document/8306879/>. Citations on pages 30 and 102.

LANGLEY, R. B. Rtk gps. **GPS World**, v. 9, n. 9, p. 70–76, 1998. Accessed: 05 jul. 2019. Available: <http://www2.unb.ca/gge/Resources/gpsworld.september98.pdf>. Citation on page 102.

LANGLEY, R. B. *et al.* Dilution of precision. **GPS world**, v. 10, n. 5, p. 52–59, 1999. Accessed: 05 jul. 2019. Available: <https://pdfs.semanticscholar.org/c04c/d3dc971734ebe2ba1aa113e417e8b28b248c.pdf>. Citations on pages 99 and 100.

LANIGAN, P. E.; KAVULYA, S.; NARASIMHAN, P.; FUHRMAN, T. E.; SALMAN, M. A. Diagnosis in automotive systems: A survey. **Last accessed Sept**, v. 10, p. 2011, 2011. Accessed: 22 jul. 2018. Available: <https://s3.amazonaws.com/academia.edu.documents/30795812/CMU-PDL-11-110.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&>

Expires=1532289627&Signature=dqJjEWgQyiUlutQprv4UTkvVi%2FQ%3D&response-content-disposition=inline%3B%20filename%3DDiagnosis_in_Automotive_Systems_A_Survey.pdf>. Citations on pages 58, 59, 91, and 94.

LEFÈVRE, S.; VASQUEZ, D.; LAUGIER, C. A survey on motion prediction and risk assessment for intelligent vehicles. **ROBOMECH journal**, SpringerOpen, v. 1, n. 1, p. 1, 2014. Access: 13 jun. 2019. Available: <<https://robomechjournal.springeropen.com/articles/10.1186/s40648-014-0001-z>>. Citation on page 54.

LITMAN, T. **Autonomous vehicle implementation predictions**. Victoria Transport Policy Institute Victoria, Canada, 2017. Accessed: 20 jul. 2018. Available: <<https://www.vtpi.org/avip.pdf>>. Citation on page 32.

LIU, S.; LI, L.; TANG, J.; WU, S.; GAUDIOT, J.-L. **Creating Autonomous Vehicle Systems**. Morgan & Claypool Publishers, 2017. i–186 p. Accessed: 16 jul. 2018. Available: <<https://www.morganclaypool.com/doi/abs/10.2200/S00787ED1V01Y201707CSL009>>. Citations on pages 27, 28, 29, 30, 31, and 32.

LIU, S.; LIU, S. An efficient expert system for machine fault diagnosis. **The International Journal of Advanced Manufacturing Technology**, Springer, v. 21, n. 9, p. 691–698, 2003. Accessed: 27 jul. 2018. Available: <<https://link.springer.com/article/10.1007/s00170-002-1389-9>>. Citation on page 61.

LOUREIRO, R.; BENMOUSSA, S.; TOUATI, Y.; MERZOUKI, R.; BOUAMAMA, B. O. Integration of fault diagnosis and fault-tolerant control for health monitoring of a class of mimo intelligent autonomous vehicles. **IEEE Transactions on Vehicular Technology**, IEEE, v. 63, n. 1, p. 30–39, 2014. Accessed: 25 jul. 2018. Available: <<https://ieeexplore.ieee.org/abstract/document/6710264/>>. Citation on page 60.

LRM. **Laboratório de Robótica Móvel - LRM ICMC**. 2018. Available: <<http://lrn.icmc.usp.br/web/index.php?n=Port.Home>>. Citations on pages 69, 70, and 78.

_____. **Laboratório de Robótica Móvel - LRM ICMC - Caminhão Autônomo**. 2018. Available: <<http://lrn.icmc.usp.br/web/index.php?n=Port.ProjSTruck>>. Citation on page 69.

LUTIN, J. M. Not if, but when: Autonomous driving and the future of transit. **Journal of Public Transportation**, v. 21, n. 1, p. 10, 2018. Accessed: 16 jul. 2018. Available: <<http://scholarcommons.usf.edu/jpt/vol21/iss1/10/>>. Citations on pages 23 and 27.

MALLICK, M. R.; IMTIAZ, S. A. A hybrid method for process fault detection and diagnosis. **IFAC Proceedings Volumes**, Elsevier, v. 46, n. 32, p. 827–832, 2013. Accessed: 01 jul. 2019. Available: <<https://www.sciencedirect.com/science/article/pii/S1474667015383610>>. Citation on page 94.

MARCOTTE, R.; HAMILTON, H. J. Behavior trees for modelling artificial intelligence in games: A tutorial. **The Computer Games Journal**, Springer, v. 6, n. 3, p. 171–184, 2017. Accessed: 14 jun. 2019. Available: <<https://link.springer.com/article/10.1007/s40869-017-0040-9>>. Citation on page 55.

MERCER, C.; MACAULAY, T. **Which companies are making driverless cars?** 2018. Accessed: 17 jul. 2018. Available: <<https://www.techworld.com/picture-gallery/data/-companies-working-on-driverless-cars-3641537/>>. Citations on pages 24 and 28.

MILGER, H.; GILLGREN, S. **Simulation of autonomous vehicles in an urban environment: An investigation on how basic models can create a realistic result**. 38 f. Bachelor's Thesis (Bacharelor) — KTH Royal Institute of Technology, São Paulo, 2015. Accessed: 15 jun. 2019. Available: <http://www.diva-portal.org/smash/record.jsf?pid=diva2%3A815681&dswid=-2335>. Citation on page 56.

MORI, J.; MAHALEC, V. Inference in hybrid bayesian networks with large discrete and continuous domains. **Expert Systems with Applications**, Elsevier, v. 49, p. 1–19, 2016. Accessed: 09 jun. 2019. Available: <https://www.sciencedirect.com/science/article/pii/S095741741500785X>. Citation on page 63.

MURPHY, R.; MURPHY, R. R. **Introduction to AI robotics**. [S.l.]: MIT press, 2000. Citation on page 29.

NASRI, O.; LAKHAL, N. M. B.; ADOUANE, L.; SLAMA, J. B. H. Automotive decentralized diagnosis based on can real-time analysis. **Journal of Systems Architecture**, Elsevier, 2019. Accessed: 01 jul. 2019. Available: <https://www.sciencedirect.com/science/article/pii/S1383762117303910>. Citation on page 94.

NG, S. K.; KRISHNAN, T.; MCLACHLAN, G. J. The em algorithm. In: **Handbook of computational statistics**. [S.l.]: Springer, 2012. p. 139–172. Citation on page 65.

OLSSON, M. **Behavior Trees for decision-making in Autonomous Driving**. Master's Thesis (Master's Thesis) — KTH Royal Institute of Technology, 2016. Available: <http://www.diva-portal.org/smash/record.jsf?pid=diva2:907048>. Citations on pages 54, 55, 57, and 58.

OPENSTREETMAP, W. **OpenStreeMap: Elements**. OpenStreetMap Wiki, 2017. Accessed: 17 jun. 2019. Available: <https://wiki.openstreetmap.org/w/index.php?title=Elements&oldid=1479648>. Citation on page 52.

_____. **Licence/Contributor Terms**. OpenStreetMap Wiki, 2019. Accessed: 18 jun. 2019. Available: https://wiki.osmfoundation.org/wiki/Licence/Contributor_Terms. Citation on page 52.

PADEN, B.; ČÁP, M.; YONG, S. Z.; YERSHOV, D.; FRAZZOLI, E. A survey of motion planning and control techniques for self-driving urban vehicles. **IEEE Transactions on intelligent vehicles**, IEEE, v. 1, n. 1, p. 33–55, 2016. Accessed: 12 may 2019. Available: <https://ieeexplore.ieee.org/abstract/document/7490340>. Citations on pages 45, 46, 54, and 83.

PENDLETON, S. D.; ANDERSEN, H.; DU, X.; SHEN, X.; MEGHJANI, M.; ENG, Y. H.; RUS, D.; ANG, M. H. Perception, planning, control, and coordination for autonomous vehicles. **Machines**, Multidisciplinary Digital Publishing Institute, v. 5, n. 1, p. 6, 2017. Accessed: 19 jul. 2018. Available: <http://www.mdpi.com/2075-1702/5/1/6>. Citation on page 31.

PHILIP, A. O.; SARAVANAGURU, R. A vision of connected and intelligent transportation systems. **International Journal of Civil Engineering and Technology (IJCIET)**, v. 9, n. 2, p. 873–882, 2018. Accessed: 17 jul. 2018. Available: https://www.researchgate.net/profile/Abin_Philip3/publication/323816739_A_vision_of_connected_and_intelligent_transportation_systems/links/5abe0bd545851584fa7042a5/A-vision-of-connected-and-intelligent-transportation-systems.pdf. Citations on pages 28 and 33.

POGGENHANS, F.; PAULS, J.-H.; JANOSOVITS, J.; ORF, S.; NAUMANN, M.; KUHN, F.; MAYR, M. Lanelet2: A high-definition map framework for the future of automated driving. In: IEEE. **2018 21st International Conference on Intelligent Transportation Systems (ITSC)**. 2018. p. 1672–1679. Accessed: 18 jun. 2019. Available: <<https://ieeexplore.ieee.org/abstract/document/8569929>>. Citations on pages 51 and 53.

QUIGLEY, M.; CONLEY, K.; GERKEY, B.; FAUST, J.; FOOTE, T.; LEIBS, J.; WHEELER, R.; NG, A. Y. Ros: an open-source robot operating system. In: KOBE, JAPAN. **ICRA workshop on open source software**. 2009. v. 3, n. 3.2, p. 5. Accessed: 29 jul. 2018. Available: <<https://www.willowgarage.com/sites/default/files/icraoss09-ROS.pdf>>. Citations on pages 65 and 66.

REDMON, J.; FARHADI, A. Yolov3: An incremental improvement. **arXiv preprint arXiv:1804.02767**, 2018. Accessed: 23 jun. 2019. Available: <<https://arxiv.org/abs/1804.02767>>. Citation on page 80.

RIBEIRO, M. I. Kalman and extended kalman filters: Concept, derivation and properties. **Institute for Systems and Robotics**, Citeseer, v. 43, 2004. Accessed: 09 jul. 2019. Available: <<http://users.isr.ist.utl.pt/~mir/pub/kalman.pdf>>. Citations on pages 103 and 104.

ROSEIRO, L. A.; OSÓRIO, F. S. Calibration and multi-sensor fusion for on-road obstacle detection. In: IEEE. **2017 Latin American Robotics Symposium (LARS) and 2017 Brazilian Symposium on Robotics (SBR)**. 2017. p. 1–6. Accessed: 21 jun. 2019. Available: <<https://ieeexplore.ieee.org/document/8215269>>. Citations on pages 81, 82, and 125.

ROUNSAVILLE, J. D.; DVORAK, J. S.; STOMBAUGH, T. S. Methods for calculating relative cross-track error for asabe/iso standard 12188-2 from discrete measurements. **Transactions of the ASABE**, American Society of Agricultural and Biological Engineers, v. 59, n. 6, p. 1609–1616, 2016. Accessed in: 05 jul. 2019. Available: <<https://elibrary.asabe.org/abstract.asp?aid=47588>>. Citation on page 97.

RUSSELL, S.; NORVIG, P. **Inteligência Artificial: Tradução da 3a Edição**. [S.l.]: Elsevier Brasil, 2013. Citations on pages 62 and 63.

RVIZ. **RViz**. 2018. Accessed: 20 jun. 2019. Available: <<http://wiki.ros.org/rviz>>. Citation on page 74.

SAE, S. **J3016 Standard: taxonomy and definitions for terms related to on-Road motor vehicle automated driving systems**. SAE, 2014. Accessed: 1 ago. 2018. Available: <https://www.sae.org/standards/content/j3016_201401/>. Citation on page 24.

SAMET, H. **The design and analysis of spatial data structures**. [S.l.]: Addison-Wesley Reading, MA, 1990. Citation on page 80.

SANTOS, T. C. d. **Uma proposta de arquitetura de software para a simulação e experimentação de veículos autônomos**. Master's Thesis (Master's Thesis) — Universidade de São Paulo, 2015. Accessed: 20 jul. 2018. Available: <<http://www.teses.usp.br/teses/disponiveis/55/55134/tde-12122016-103140/en.php>>. Citations on pages 70 and 125.

SCANIA. **Mine blowing**. SAE, 2016. Accessed: 14 ago. 2018. Available: <<https://www.scania.com/group/en/mine-blowing/>>. Citation on page 56.

SCHWALL, M. L.; GERDES, J. C. A probabilistic approach to residual processing for vehicle fault detection. In: IEEE. **Proceedings of the 2002 American Control Conference (IEEE Cat. No. CH37301)**. 2002. v. 3, p. 2552–2557. Accessed: 02 jul. 2019. Available: <https://ieeexplore.ieee.org/abstract/document/1024028/>. Citations on pages 94, 95, and 105.

SEPTENTRIO. **Firmware User Manual: Applicable to SSRC2 3.3.0**. 2014. Accessed: 08 jul. 2019. Available: <https://www.septentrio.com/en/support/asterx/asterx2eh>. Citations on pages 101 and 117.

SHERIFF, M. Z.; BOTRE, C.; MANSOURI, M.; NOUNOU, H.; NOUNOU, M.; KARIM, M. N. Process monitoring using data-based fault detection techniques: Comparative studies. In: **Fault Diagnosis and Detection**. InTech, 2017. Accessed: 30 jun. 2019. Available: <https://www.intechopen.com/books/fault-diagnosis-and-detection/process-monitoring-using-data-based-fault-detection-techniques-comparative-studies>. Citations on pages 60 and 61.

SHLADOVER, S. E. Connected and automated vehicle systems: Introduction and overview. **Journal of Intelligent Transportation Systems**, Taylor & Francis, v. 22, n. 3, p. 190–200, 2018. Accessed: 17 jul. 2018. Available: <https://www.tandfonline.com/doi/abs/10.1080/15472450.2017.1336053>. Citation on page 34.

SILVA, J. A.; GRASSI, V. Path planning at roundabouts using piecewise linear continuous curvature curves. In: IEEE. **2017 Latin American Robotics Symposium (LARS) and 2017 Brazilian Symposium on Robotics (SBR)**. 2017. p. 1–6. Accessed: 25 jun. 2019. Available: <https://ieeexplore.ieee.org/abstract/document/8215288/>. Citation on page 86.

_____. Clothoid-based global path planning for autonomous vehicles in urban scenarios. In: IEEE. **2018 IEEE International Conference on Robotics and Automation (ICRA)**. 2018. p. 4312–4318. Accessed: 22 jun. 2019. Available: <https://ieeexplore.ieee.org/abstract/document/8461201>. Citations on pages 25, 76, 77, 86, 90, and 125.

SILVA, J. A. R. d. **Piecewise linear continuous-curvature path planning for autonomous vehicles**. Master's Thesis (Master's Thesis) — Universidade de São Paulo, 2018. Accessed: 27 jul. 2018. Available: <http://www.teses.usp.br/teses/disponiveis/18/18153/tde-02032018-095552/pt-br.php>. Citations on pages 72 and 125.

SIMPSON, J.; JACOBSEN, C. L.; JADUD, M. C. **Mobile robot control**. [S.l.: s.n.], 2006. 225 p. Citation on page 29.

SIQUEIRA, F. de L.; PLENTZ, P. D. M.; PIERI, E. R. D. Semantic trajectory applied to the navigation of autonomous mobile robots. In: IEEE. **2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA)**. 2016. p. 1–8. Accessed: 15 jun. 2019. Available: <https://ieeexplore.ieee.org/abstract/document/7945766/>. Citation on page 56.

SMITH, D.; TIMMS, G.; SOUZA, P. D.; D'ESTE, C. A bayesian framework for the automated online assessment of sensor data quality. **Sensors**, Molecular Diversity Preservation International, v. 12, n. 7, p. 9476–9501, 2012. Accessed: 07 jun. 2019. Available: <https://www.mdpi.com/1424-8220/12/7/9476>. Citation on page 63.

SOLUTIONS, A. K. **Robotics**. Hingham, Massachusetts: Infinity Science Press LLC, 2007. Citation on page 29.

STANLEY, G. *et al.* **A guide to fault detection and diagnosis**. 2013. Accessed: 25 jul. 2018. Available: <<https://gregstanleyandassociates.com/whitepapers/FaultDiagnosis/faultdiagnosis.htm>>. Citations on pages 58, 59, 60, and 94.

TANENBAUM, A. S.; STEEN, M. V. **Distributed systems: principles and paradigms**. [S.l.]: Prentice-Hall, 2007. Citation on page 66.

TAŞ, Ö. Ş.; KUHNT, F.; ZÖLLNER, J. M.; STILLER, C. Functional system architectures towards fully automated driving. In: IEEE. **Intelligent Vehicles Symposium (IV), 2016 IEEE**. 2016. p. 304–309. Accessed: 29 jul. 2018. Available: <<https://ieeexplore.ieee.org/abstract/document/7535402/>>. Citation on page 37.

TAŞ, Ö. Ş.; SALSCHIEDER, N. O.; POGGENHANS, F.; WIRGES, S.; BANDERA, C.; ZOFKA, M. R.; STRAUSS, T.; ZÖLLNER, J. M.; STILLER, C. Making bertha cooperate—team anyway’s entry to the 2016 grand cooperative driving challenge. **IEEE Transactions on Intelligent Transportation Systems**, IEEE, v. 19, n. 4, p. 1262–1276, 2018. Accessed: 6 jul. 2018. Available: <<https://ieeexplore.ieee.org/abstract/document/8055618/>>. Citations on pages 40, 41, and 72.

TEMBO, S. R.; VATON, S.; COURANT, J.-L.; GOSSELIN, S. A tutorial on the em algorithm for bayesian networks: application to self-diagnosis of gpon-ftth networks. In: IEEE. **2016 International Wireless Communications and Mobile Computing Conference (IWCMC)**. [S.l.], 2016. p. 369–376. Citation on page 64.

THRUN, S.; MONTEMERLO, M.; DAHLKAMP, H.; STAVENS, D.; ARON, A.; DIEBEL, J.; FONG, P.; GALE, J.; HALPENNY, M.; HOFFMANN, G. *et al.* Stanley: The robot that won the darpa grand challenge. **Journal of field Robotics**, Wiley Online Library, v. 23, n. 9, p. 661–692, 2006. Accessed: 16 jul. 2018. Available: <<https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.20147>>. Citation on page 28.

ULBRICH, S.; RESCHKA, A.; RIEKEN, J.; ERNST, S.; BAGSCHIK, G.; DIERKES, F.; NOLTE, M.; MAURER, M. Towards a functional system architecture for automated vehicles. **arXiv preprint arXiv:1703.08557**, 2017. Accessed: 29 jul. 2018. Available: <<https://arxiv.org/abs/1703.08557>>. Citation on page 37.

URMSON, C.; ANHALT, J.; BAGNELL, D.; BAKER, C.; BITTNER, R.; CLARK, M.; DOLAN, J.; DUGGINS, D.; GALATALI, T.; GEYER, C. *et al.* Autonomous driving in urban environments: Boss and the urban challenge. In: **The DARPA Urban Challenge**. [S.l.]: Springer, 2009. p. 1–59. Citation on page 28.

VENKATASUBRAMANIAN, V.; RENGASWAMY, R.; YIN, K.; KAVURI, S. N. A review of process fault detection and diagnosis: Part i: Quantitative model-based methods. **Computers & chemical engineering**, Elsevier, v. 27, n. 3, p. 293–311, 2003. Accessed: 26 jul. 2019. Available: <<https://www.sciencedirect.com/science/article/pii/S0098135402001606>>. Citations on pages 61 and 107.

VERES, S. M.; MOLNAR, L.; LINCOLN, N. K.; MORICE, C. P. Autonomous vehicle control systems—a review of decision making. **Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering**, SAGE Publications Sage UK: London, England, v. 225, n. 2, p. 155–195, 2011. Accessed: 19 jul. 2018. Available: <<http://journals.sagepub.com/doi/abs/10.1177/2041304110394727>>. Citation on page 31.

WANG, F.-Y. Ai and intelligent vehicles future challenge (ivfc) in china: From cognitive intelligence to parallel intelligence. In: IEEE. **ITU Kaleidoscope: Challenges for a Data-Driven Society (ITU K)**, 2017. 2017. p. 1–2. Accessed: 30 jul. 2018. Available: <https://ieeexplore.ieee.org/abstract/document/8246841/>. Citation on page 37.

WANG, J.; OBER, P. B. On the availability of fault detection and exclusion in gnss receiver autonomous integrity monitoring. **the Journal of Navigation**, Cambridge University Press, v. 62, n. 2, p. 251–261, 2009. Accessed: 05 jul. 2019. Available: <https://www.cambridge.org/core/journals/journal-of-navigation/article/on-the-availability-of-fault-detection-and-exclusion-in-gnss-receiver-autonomous-integrity-monitoring/0D1CEC47B93C4DC10D99600D7834EA2D>. Citations on pages 100 and 101.

WAYMO. **Waymo Journey**. 2018. Accessed: 17 jul. 2018. Available: <https://waymo.com/journey/>. Citation on page 28.

WEI, J.; SNIDER, J. M.; GU, T.; DOLAN, J. M.; LITKOUHI, B. A behavioral planning framework for autonomous driving. In: IEEE. **2014 IEEE Intelligent Vehicles Symposium Proceedings**. 2014. p. 458–464. Accessed: 15 May 2019. Available: <https://ieeexplore.ieee.org/document/6856582>. Citations on pages 45, 46, 47, 48, 83, and 84.

WEI, J.; SNIDER, J. M.; KIM, J.; DOLAN, J. M.; RAJKUMAR, R.; LITKOUHI, B. Towards a viable autonomous driving research platform. In: IEEE. **Intelligent Vehicles Symposium (IV)**, 2013 IEEE. 2013. p. 763–770. Accessed: 25 jul. 2018. Available: <https://ieeexplore.ieee.org/abstract/document/6629559/>. Citations on pages 43 and 44.

WHAIDUZZAMAN, M.; SOOKHAK, M.; GANI, A.; BUYYA, R. A survey on vehicular cloud computing. **Journal of Network and Computer Applications**, Elsevier, v. 40, p. 325–344, 2014. Accessed: 19 jul. 2018. Available: <https://www.sciencedirect.com/science/article/pii/S1084804513001793>. Citation on page 32.

YANG, C. D.; OZBAY, K.; BAN, X. Developments in connected and automated vehicles. Taylor & Francis, 2017. Accessed: 17 jul. 2018. Available: <https://www.tandfonline.com/doi/full/10.1080/15472450.2017.1337974>. Citation on page 33.

YIN, X.; JIANG, C.; LI, B. Fault diagnosis based on fault tree and bayesian network for pure electric trucks. In: IEEE. **Technology, Networking, Electronic and Automation Control Conference (ITNEC)**, 2017 IEEE 2nd Information. 2017. p. 994–998. Accessed: 25 jul. 2018. Available: <https://ieeexplore.ieee.org/abstract/document/8284886/>. Citation on page 63.

YU, M.; WANG, D. Model-based health monitoring for a vehicle steering system with multiple faults of unknown types. **IEEE Trans. Industrial Electronics**, v. 61, n. 7, p. 3574–3586, 2014. Accessed: 25 jul. 2018. Available: <https://ieeexplore.ieee.org/document/6594821/>. Citation on page 60.

ZANG, A.; CHEN, X.; TRAJCEVSKI, G. High definition maps in urban context. **SIGSPATIAL Special**, ACM, v. 10, n. 1, p. 15–20, 2018. Accessed: 19 jul. 2018. Available: <https://dl.acm.org/citation.cfm?id=3231546>. Citation on page 32.

ZHANG, Y.; YOU, L.; JIA, C. Fault detection and diagnosis using bayesian-network inference. In: IEEE. **Industrial Electronics Society, IECON 2017-43rd Annual Conference of the IEEE**. 2017. p. 5049–5053. Accessed: 27 jul. 2018. Available: <https://ieeexplore.ieee.org/abstract/document/8216872/>. Citation on page 61.

ZHENG, L.; SAYED, T. Comparison of traffic conflict indicators for crash estimation using peak over threshold approach. **Transportation Research Record**, SAGE Publications Sage CA: Los Angeles, CA, p. 0361198119841556, 2019. Accessed: 26 jun. 2019. Available: <<https://journals.sagepub.com/doi/full/10.1177/0361198119841556>>. Citation on page 89.

ZHU, H. **Software design methodology: From principles to architectural styles**. [S.l.]: Elsevier, 2005. Citations on pages 35 and 36.

ZIEGLER, J.; BENDER, P.; SCHREIBER, M.; LATEGAHN, H.; STRAUSS, T.; STILLER, C.; DANG, T.; FRANKE, U.; APPENRODT, N.; KELLER, C. G. *et al.* Making bertha drive—an autonomous journey on a historic route. **IEEE Intelligent Transportation Systems Magazine**, IEEE, v. 6, n. 2, p. 8–20, 2014. Accessed: 18 jun. 2019. Available: <<https://ieeexplore.ieee.org/abstract/document/6803933>>. Citations on pages 51 and 53.

ZONG, W.; ZHANG, C.; WANG, Z.; ZHU, J.; CHEN, Q. Architecture design and implementation of an autonomous vehicle. **IEEE Access**, IEEE, v. 6, p. 21956–21970, 2018. Accessed: 6 jul. 2018. Available: <<https://ieeexplore.ieee.org/abstract/document/8340798/>>. Citations on pages 37, 38, and 72.

PUBLICATIONS

Gomes, I. P., Bruno, D. R., Osório, F. S., and Wolf, D. F. (2018, November). Diagnostic Analysis for an Autonomous Truck Using Multiple Attribute Decision Making. In *2018 Latin American Robotic Symposium, 2018 Brazilian Symposium on Robotics (SBR) and 2018 Workshop on Robotics in Education (WRE)* (pp. 283-290). IEEE.

Gomes, I. P., and Wolf, D. F. (2019). A Health Monitoring System with Hybrid Bayesian Network for Autonomous Vehicle. In *19th International Conference on Advanced Robotics (ICAR)*.

Amado, J. A. D., Gomes, I. P., Amaro, J., Wolf, D. F., and Osório, F. S., (2019, June). End-to-End Deep Learning Applied in Autonomous Navigation using Multi-Cameras System with RGB and Depth Images. In *2019 IEEE Intelligent Vehicles Symposium*. IEEE.

Silva, J. A. R., Gomes, I. P., Wolf, D. F., Grassi Jr., V. (2019). Sparse Road Network Model for Autonomous Navigation Using Clothoids. *Transactions on Intelligent Transportation Systems* . **Manuscript submitted for publication.**