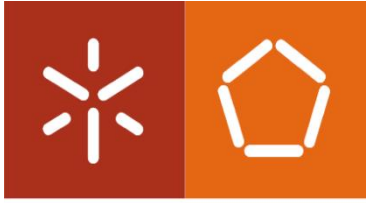


Universidade do Minho
Escola de Engenharia

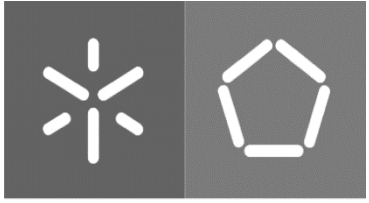
Nicolas Raphäel Tinoco Almeida

# A Driver Workload System for the Cockpit of the Future

Master's Dissertation
Master Degree in Industrial Electronics and Computers Engineering

Dissertation supervised by
**PhD Professor Adriano Tavares**
**MsC Daniel Oliveira**

Guimarães, February of 2018



Universidade do Minho
Escola de Engenharia

Nicolas Raphäel Tinoco Almeida

# A Driver Workload System for the Cockpit of the Future

Master's Dissertation
Master Degree in Industrial Electronics and Computers Engineering

Dissertation supervised by
**PhD Professor Adriano Tavares**
**MsC Daniel Oliveira**

Guimarães, February of 2018

# Agradecimentos

Aos meus pais, Joaquim Almeida e Rosa Tinoco, dirijo as minhas primeiras palavras de agradecimento, pois, com todo o seu suporte, amor e confiança, permitiram traçar o meu caminho académico. Todas as condições e oportunidades que me vão proporcionando levaram a que esta dissertação fosse uma realidade. Porque sem eles tal sucesso não seria possível.

À minha irmã, dedico este segundo parágrafo, pelo suporte e amor que, apesar de nem sempre ser o mais evidente, está intrínseco na nossa relação. E à minha afilhada, pelos momentos de carinho e amor, e por me proporcionar sorrisos e bem-estar mesmo quando os momentos não eram os mais felizes.

Ao meu orientador, Professor Doutor Adriano Tavares, pela excelente orientação e por toda a confiança que depositou em mim para a materialização deste trabalho, mesmo quando os desafios eram grandes. Além disso, agradeço especialmente por todo o conhecimento e rigor transmitidos, e pelo seu papel fulcral na minha formação académica na área adjacente a esta dissertação.

Ao meu "co-orientador", Mestre Daniel Oliveira, pelo rigor e conhecimento que transmitiu, não me exigindo menos do que o melhor. Agradeço ainda pela motivação que sempre me tentou passar e por nunca desistir de me incentivar, mesmo em tempos de maior intranquilidade e ansiedade. E por último, um obrigado pela amizade e disponibilidade, em que nunca ignorou ou abdicou de me ajudar a formar soluções para os problemas que iam aparecendo. Tal suporte e atitudes serão jamais esquecidas.

Aos meus colegas de curso, em especial ao grupo "Orlando's - A Gentleman's club", pela ajuda disponibilizada, amizade e partilha demonstrada ao longo destes cinco anos de vida académica e que continuará para o resto da vida pessoal.

Ao melhor laboratório do CCG, que me acolheu com simpatia e amizade, e suportou as minhas piadas ao longo de mais de um ano. Além disso, sempre se esforçou por me proporcionar todas as condições e recursos necessários para a elaboração e término deste trabalho.

Por último, mas não menos importante, um agradecimento especial à minha namorada, Luciana Rodrigues, pelo constante suporte, paciência, preocupação e carinho que me dedicou ao longo deste percurso. Por sempre acreditar que este trajeto seria concluído com sucesso, um muito obrigado. Um agradecimento também aos meus melhores amigos, Miguel Vieira, Catarina Fernandes e Iolanda Oliveira, aos amigos de longa data (Zé, Bruno, Mica, André, Miguel, Catarina, Rute, Pedro e Ricardo) e a todos os amigos do "Bacocos" por todos os momentos de verdadeira amizade e alegria partilhados, que foram capitais durante o meu percurso académico e pessoal.

À minha família e a todas as outras pessoas que me suportaram direta ou indiretamente: um obrigado!

Dedicado a ti, minha avó...

# Resumo

O surgimento de tecnologias novas e disruptivas tecnologias/tendências estão a revolucionar todos os sistemas projetados, desenvolvidos e fabricados pelos nossos principais domínios da indústria. Esses sistemas tornam-se cada vez mais complexos com a extensão das suas funcionalidades e da sua pilha tecnológica. O automóvel é tradicionalmente um dos sistemas mais complexos e deve tal conotação à importância que representa nas sociedades modernas e ao contexto delicado em que atua. A evolução tecnológica está a mudar a forma como eles operam (*e.g.*, Condução autónoma) e como as pessoas interagem com ele (*e.g.*, Conexão do *smartphone* ao veículo).

Esta mudança de paradigma instigou o surgimento de sistemas avançados de assistência à condução (ADAS), sistemas de informação no veículo (IVIS) e serviços interligados [1]. Esses sistemas fornecem acesso a informação dentro do veículo, permitindo melhorar o conforto e a segurança rodoviária. Além disso, um grande número de funções IVIS são apresentadas em sistemas móveis computorizados, referidos como dispositivos nómadas (*e.g.*, telemóveis) que não podem ser usados durante a condução devido à indução de níveis perigosos de *workload* no condutor [2, 3]. Estas tendências aumentam o *workload* no cockpit, podendo levar à distração do condutor e à diminuição do desempenho da condução, resultando em acidentes fatais [4, 5]. É neste contexto que surge o Projeto "P689 - Cockpit of the Future: HMI Concepts and Functions", com o objetivo de desenvolver novos conceitos, soluções e formas de interação que permitam a redução do *workload* do condutor e evitar possíveis distrações durante a tarefa principal de condução.

A presente tese propõe desenvolver uma solução que aborda parte do problema apontado pelo projeto (*i.e.*, o *workload* do condutor). Um sistema que avalia e gere o nível de *workload* do condutor, classificando-o e propondo ações a serem aplicadas no *cockpit* do veículo, a partir do contexto de condução. O objetivo do sistema é normalizar os níveis de *workload*, permitindo que o condutor se concentre exclusivamente na tarefa essencial e prioritária de condução.

# Abstract

The rise of new and disruptive technology-trends are revolutionizing all systems designed, developed and manufactured by our key industry domains. Those systems are day-by-day becoming more complex with the extend of their functionalities and their technological stack. The car is traditionally one of the most complex systems, and it owes this status due to the importance that it represents on modern societies and the delicate context in which it operates. Its technological evolution is changing the way they operate (*e.g.*, Autonomous Driving) and how people interact with it (*e.g.*, Smartphone connection to the vehicle).

This paradigm shift has instigated the arising and development of convoluted Advanced Driver Assistance Systems (ADAS), In-vehicle Information systems (IVIS) and interconnected services [1]. Those systems provide in-vehicle access to new information, allowing the improve of comfort and enhancing road safety. Furthermore, a great number of IVIS functions are featured on portable computing systems, referred as nomadic devices (*e.g.*, mobile phones) that are not allowed to be used while driving due to the inducement of dangerous levels of driver workload [2, 3]. These trends increase the cockpit workload, which can instigate driver distraction and the decrement of driving performance, resulting in fatal road accidents [4, 5]. Is in this context that the Project "P689 - Cockpit of the Future: HMI Concepts and Functions" emerges, aiming to develop new concepts, solutions and forms of interaction that allow the driver workload reduction and avoid possible distractions, keeping the driver fully engaged in the primary driving task.

The present thesis proposes to develop a solution that addresses part of the problem pointed out by the project (*i.e.*, the Driver Workload). A system that assesses and manages the driver workload level by classifying it and proposing actions to be applied in the vehicle cockpit, based on the driving context. The overall system goal is to counteract abnormal workload levels, enabling the driver to focus on the essential and priority driving task.

# Contents

# List of Figures

# List of Tables

# Acronyms

**ADAS** ..... Advanced Driver Assistance Systems

**AIDE** ..... Adaptive Integrated Driver-Vehicle Interface

**AI** ..... Artificial Intelligence

**ANN** ..... Artificial Neural Network

**API** ..... Application Programming Interface

**BDS** ..... Berkeley Software Distribution

**BN** ..... Bayesian Network

**CAN** ..... Controller Area Network

**CNN** ..... Convolutional Neural Network

**CPU** ..... Central Processing Unit

**DSM** ..... Driver Simulator Mockup

**DVE** ..... Driver Vehicle Environment

**DWE** ..... Driver Workload Estimation

**ECG** ..... Electrocardiogram

**EDA** ..... Electrodermal activity

**EEG** ..... Electroencephalogram

**EMG** ..... Electromyogram

**EOG** ..... Electrooculogram

**FPGA** ..... Field Programmable Gate Array

**GPU** ..... Graphics Processing Unit

**HEM** ..... Horizontal Eye Movement

**HMI** ..... Human-Machine Interface

**HR** ..... Heart Rate

**HRV** ..... Heart Rate Variability

**Hz** ..... Hertz

**IVIS** ..... In-Vehicle Information Systems

**INNOVCAR** . Innovative Car

**IoT** ..... Internet of Things

**MSE** ..... Mean Square Error

**NB** ..... Naïve Baines

**ReLu** ..... Rectified Linear Unit

**RNN** ..... Recurrent Neural Network

**SAVE-IT** ..... SAfety VEhicle using adaptive Interface Technology

**SCL** ..... Skin Conductance Level

**SDHG** ..... Standard Deviation of Horizontal Gaze

**SDLP** ..... Standard Deviation of Lateral Position

**SDVG** ..... Standard Deviation of Vertical Gaze

**SRR** ..... Steering wheel Reversal Rate

**TFS** ..... TensorFlow Serving

**USB** ..... Universal Serial Bus

# Chapter 1

# Introduction

Driver distraction is a major contributing factor to automobile crashes. Statistics say that, in 2014, about 33.000 people died in traffic accidents all around United States of America [4]. Besides, the majority of road accidents had human error as the main cause, remarking driver distraction as the main source [5]. The issue of driver distraction regarding the primary task (*i.e.*, driving task) has become progressively important with the increase technological evolution. This paradigm shift lead to the introduction of complex in-vehicle technology, such as ADAS and IVIS, and instigated the integration of electronic devices (e.g., smartphones, navigation systems, wearables, and wireless Internet) into vehicles [1, 26]. More specifically, the infotainment system uses the center console screen to display the navigation system, to show radio and smartphone information and even to display multimedia content. Moreover, in the place of classical instrument clusters with analog indicators, high-end screens took place for the display of speed information and vehicle warnings. Briefly, IVIS analyzes data concerning traffic, external environment, driving, and the vehicle in order to cater helpful information and to provide a enhanced driving experience. The ADAS alerts the driver to dangerous situations and increases driving safety, based on the driving context [27]. The increasing integration of such technologies allied with the highly complex driving environment (*i.e.*, traffic density, visibility problems due to bad weather conditions, complex road sections) can potentiate the aforementioned problem and cause dangerous levels of workload by providing more distraction sources [26, 28]. Furthermore, it diverts driver's attention away from critical activities undermining the adoption of a safe driving posture [29].

Almost all researches and projects address usually a sub-class of the afore-

mentioned problem, the driver workload (*e.g.*, AIDE [2], SAVE-IT [30] or COMUNICAR [31]). In this context, the driver state is acknowledge and studied as a key factor for the improvement of the interaction between the driver and the vehicle. This interaction is often performed through Human -Machine Interfaces (HMI) available in the car. Although in the early days encompassed only primary interactions (*e.g.*, steering wheel, accelerator and brake), evolution brought the need to drivers to know more about the vehicle state [32]. Despite the driving safety improvement and increased comfort that the integration of those technologies provide, drivers have to cope with a huge amount of information during the driving task [20]. Therefore, a important question remains in the understanding of when and how the information must be provided to the driver. This plurality of systems and functionalities that interact with the driver is leading to the emergence of new and complex challenges that must be addressed during the design of future HMI solutions [33]. Solutions based on adaptive Human-Machine Interfaces allow the adaptation of information that is presented to the driver through multiple HMI devices. The information limitation criteria is based on driving context and the driver state (*e.g.*, Workload level).

In this context, the Project "P689 - Cockpit of the Future: HMI Concepts and Functions" appears with the propose to develop concepts and solutions that allow forms of interaction between the driver and the vehicle that enables the driver workload reduction and the mitigation of possible distractions associated to this type of interactions. The P689 project aims to design and develop an advanced HMI system appropriated for (i) semi-autonomous and autonomous driving, (ii) capable of manage the driver workload and (iii) able to adapt itself, taking into account the context and driving scenario [34].

The present dissertation aims to design and develop a composite system that assesses and manages the driver workload level, based on the driving context and a Driver-Vehicle-Environment (DVE) approach. This system makes use of a Machine-Learning algorithm for the driver's workload classification and, therefore, assessing the workload (Workload Assessor). For the purpose, a Google Machine-Learning framework, TensorFlow [17, 22], is applied. Moreover, based on the classified level, the system proposes a specific action to the HMI system to be applied within the vehicle cockpit. The action intends to bring the driver workload to an acceptable normalized level (Workload Manager). The driver history of applied actions dictates which is the best suited action to be applied, in a specific driving context. Prioritizing the displayed information, propose the transition to a higher vehicle automation level or turn on the cockpit lights are part of the set

of actions available to utilization and their application context is well defined.

## 1.1 Goals

The system that this dissertation intends to design and develop aims to address a set of goals. In order to present those goals, it is essential to understand what kind of systems will be addressed, which functionality they provide and what they imply.

The holistic system can be decomposed in two sub-systems: (i) the Workload Assessor and (ii) the Workload Manager. Its main propose is to be capable of identify and interpret possible correlations regarding the driver's workload, based on driver's physiological features, vehicle telemetry and environment context variables. Furthermore, the system is capable of manage the driver's workload by applying actions to the HMI system according to the current driving context (*i.e.*, the driver, the vehicle and the environment).

The Workload Assessor (WA) concerns the estimation of the driver's workload level based on the driver, the car itself, and external environmental variables [13]. Briefly, the WA should analyze the (i) driver's biometric data (*e.g.*, heart rate, PERCLOS, etc), (ii) the car telemetry (*e.g.*, velocity, acceleration, lateral deviation, etc), and (iii) the driving external environment (*e.g.*, road condition, weather conditions, etc), and, based on these values, estimates the possible workload level held by the driver.

The Workload Manager system has the capability to, based on the abnormal driver workload level, the external context, the driver state and activity, and the vehicle automation level, propose specific actions to counter the abnormal level at which the driver is subjected. The actions chosen to be applied to the HMI system are specific to each driver due to the subjectivity that the reaction to a particular action presents.

The set of goals presented below are the result of the group of tasks that outlines the overall work to be developed, within this dissertation. Briefly, the set of main goals regarding this dissertation are discriminated in the following points:

- Design and Development of a Workload Assessor system (which encompasses a machine-learning algorithm that estimates the driver workload in three levels: *low*, *normal* and *high*);

- Design and Development of a Workload Manager system (which encompasses a management algorithm that proposes actions to the HMI system in order to normalize the driver's workload level);
- Workload Assessor and Manager integration in the Advanced HMI system developed within the project "P689 - Cockpit of the Future: HMI Concepts and Functions", regarding the program INNOVCAR;

## 1.2 Dissertation's Structure

The present dissertation starts with a brief contextualization (Chapter 1) about the new trends and challenges that the automotive industry faces today, the problems that are collaterally generated and possible solutions. Moreover, an overview regarding the dissertation purpose and the P689 project context where it is inserted and where the development took place. Towards the end of this introductory chapter the dissertation goals proposed by the author are made known.

The Chapter 2 encompasses two main sections, which present (i) the theoretical basis on the concepts addressed by this dissertation and (ii) related work developed by the science community and industrial manufacturers that address the problem that this dissertation tries to solve. The first main section tries to present important background in order to fully understand all the succeeding chapters that require a stable knowledge basis for its understanding. In a first phase, concepts regarding Human-Machine Interfaces are given, and several machine learning algorithms are presented and its advantages and disadvantages exposed. Then, mental workload concepts are clarified concerning the driver workload and its implications for driving, the workload model and its variations during the driving task, and various types of measurement tools that enable the workload level estimation. The second main section introduces the related work regarding machine-learning algorithms estimating driver workload level. Moreover, a state of the art section regarding machine learning frameworks or APIs are presented and compared. Its main advantages and disadvantages are exposed and an overview regarding its inner-operation is given.

The Chapter 3 describes the overall system specification. It begins with an overview of the system's architecture and all its modules that were designed within the P689 project. Moreover, the modules developed in this dissertation are highlighted and contextualized concerning the Project structure. The Workload Assessor architecture is also detailed and the components that compose this mod-

ule are specified. Moreover, the integration of TensorFlow framework [17] in the Workload Assessor development process is explained and an overview of the training and deployment pipelines are given. The Workload Manager architecture is specified in the same terms as the Workload Assessor architecture is. The adopted approach for its development is well explained and the justification regarding its design is acknowledged. Furthermore, the experiment setup is specified and its procedure explained. All scenarios that compose the experiment are given and the details regarding how each experiment scenario was performed are presented. Also, the data collection that took place during the experiment is specified as well as all the features that were considered. Lastly, the development environment that make possible the implementation phase is defined and the importance of each component explained.

The implementation description of both Workload Assessor and Manager using a machine-learning approach is performed in Chapter 4. The implementation of the more important modules of each system (*i.e.*, Workload Assessor and Manager) is detailed. These modules range from the model construction (*i.e.*, the machine-learning algorithm) and its training cycle to the mechanism of selection and reward of actions to counteract abnormal levels of driver workload.

Chapter 5 presents the results regarding the performed tests. The evaluation of the Workload Assessor module accuracy and the Workload Manager effectiveness is exposed. Furthermore, metrics that helped in the evaluation procedure are identified and properly explained.

Lastly, the Chapter 6 describes conclusions acquired from the developed work, as well as all the limitations comprised from it. Moreover, some suggestions that aim a future work and the improvement of the developed modules are presented.

# Chapter 2

# State of the Art

This dissertation will embrace a problem regarding the automotive domain, specifically driver's workload. Thus, different concepts concerning Human-Machine Interfaces, Machine Learning algorithms and frameworks, and Mental Workload must be addressed in this chapter, in order to better understand all the aspects and functionalities that will support and help in the realization of the goals that this thesis aims to achieve. Therefore, the explanation of some theoretical concepts concerning the thesis domain, will compose a first section of this chapter. Moreover, a state of the art analysis regarding machine learning frameworks or APIs composes the final part of this introductory section. In the second section of the chapter, solutions and approaches stated by the scientific community and the industry to solve and address the driver workload issue are referred and explained.

## 2.1 Background

This section will address some concepts that will be important in order to better understand some of the knowledge concerning this thesis. Therefore, it is divided in two parts: (i) the first part presents Workload related concepts (Section 2.1.1) and a literature review regarding workload studies developed by the scientific community (Section 2.1.1.4); and (ii) the second part encompasses the explanation of some Machine Learning concepts (Section 2.1.2) and, also, of frameworks that allow the implementation of such algorithms (Section 2.1.2.6).

### 2.1.1 Workload Concepts

In a simplistic way, the *workload* concept can be defined and perceived as a demand placed to humans [7]. However, this is a very simplistic view since it puts more emphasis on external demands. Therefore, *workload* is not reflected only on a particular task, but also, and inherently, reports to a specific person [35]. This means that, not only individual capabilities, but also motivation to perform a task, strategies applied in task performance, as well as mood and operator state, affect experienced load. Regarding the automotive domain, Roskam *et al.* [36] affirms that workload is not only defined by the amount of resources required by a set of concurrent tasks, but also by the resources needed to perform them. Therefore, the author distinguishes three different types of workload: (i) *visual workload* (e.g., how many types of sources have the driver to look?); (ii) *motor workload* (e.g., what should the driver do with their hands or feets?); (iii) *mental workload* (e.g., how many types of information has the driver to process?). In the *mental workload* literature, task demands and the effect of it on the operator are sometimes wrongly indicated with the same term, 'workload'. Therefore, load or workload will be used to describe, during this document, as the effect that the demand has on the operator in terms of stages that are used in information processing and their energetics.

Briefly, workload is the specification of the amount of information processing capacity that is used for task performance. More specifically, workload involves various processes, where neurophysiologic, perceptual and cognitive processes are included. In the concept of mental workload, how the goal is reached (e.g. the sequence of actions) and individual restrictions imposed upon performance (e.g. in terms of accuracy or speed) are included. Consequently, workload depends upon the individual, and regarding to the interaction between operator and task structure, the same task demands will not result in an equal level of workload for all individuals [7].

Directly related to demand is task complexity. Complexity increases with an increase in the number of stages of processing that are required to perform a task. Task demand and complexity are mainly external, but both depend upon subjective goals set for task performance. Difficulty of a task is related to the processing effort (amount of resources) that is required by the individual for task performance, and is dependent upon context, state, capacity and strategy of resources allocation.

Depending on the best suited degree, mental workload can vary between low

and very high levels (i.e., underload and overload, respectively). These two edges of the optimal level (i.e., level at which operator feels comfortable, can manage task demands and maintains a good performance) are classified as inappropriate and can lead to imperfect or inaccurate perceptions, as well as to low levels of attention and capacity, and to insufficient time for a proper information processing. High levels of mental workload occur when task demands exceed performer capacity [37, 38, 10].

In this thesis scope, the assessment of workload is coupled with task demand experienced by the driver specifically because various reactions to task demand variations are possible. Therefore, drivers can adapt their behavior and act respectively concerning an increase in demand. Its behavior change can be accomplished by the modification of their strategy and task goals and comply with a lower performance level. Due to the highly subjective workload assessment, between individuals, strategies will also differ and effectiveness and the required effort to reach the same level of performance will different. Regarding the coping with the demand, the effort will grow while performance remains at the same level. Therefore, in this case, performance measures will not reflect any change and be insensitive to the increase in workload, while other measures, such as self-report ratings or physiological measures, may well give an indication of effort carried out. In scenarios in which a change in the driver strategy occurs, measures of effort may remain unchanged or even show a decrease, while performance measures will indicate decreased task performance [7].

#### 2.1.1.1 Driver Workload

The workload related to the driver must be considered individually by identifying the most affecting factors. Therefore, a model of the main task of the driver is useful in mental workload research in driving. As stated in [7], a simple definition of the role of the primary task is to have a safe control of the vehicle concerning the traffic environment. However, due to the dynamic control of the driving task, that is related with the continuously changing environment, and since this task is influenced by external traffic drivers, the driver primary task definition must be more complex in order to truly represent car driving complexity. Therefore, a more complex task definition is presented in [39]. This definition purposes the task decomposition in minimum a of three levels hierarchically structured:

- *Strategic Level* - at the top level, it is where strategic decisions are made

(*e.g.*, setting a route destination or route-choice while driving);

- *Manoeuvring Level* - at the intermediate level, it is where reactions to situational traffic events take place;

- *Control Level* - at the bottom level, it is where basic control processes of the vehicle happen (*e.g.*, lateral-position control);

Increasing demands at all three levels can surpass driver capacity, and eventually result in affected performance at all levels.

The driver workload can be influenced by several variables. Sources of it can be found both inside and outside of the vehicle (*e.g.*, crossing a complex junction or an important phone call). Due to the highly visual component of driving, the main demands that affect the driver are on visual and mental resources. However, the continuous technology development instigates the introduction of intelligent in-car devices that raise the driver mental effort and the allocation of auditory resources. Therefore, these devices will increase driver mental workload and affect performance negatively.

Another problem that affects the driver is concerned with information processing. The introduction of the aforementioned intelligent devices in the cockpit habitat will impose an overload on information assimilation. Consequently, driver workload will be affected, causing performance deterioration [7].

The paradigm shift that cars are experience is changing the way we drive with the increase of semi- and fully-autonomous driving vehicles surfacing in the market day-by-day. In this scenarios, drivers are not responsible to take manual control of the steering wheel or even the gas pedal. Allied to this condition, new technologies that increase the comfort and the driving experience are removing the driving task responsibility of the driver. Therefore, this technology and paradigm will lead to monotony and have the opposite effect of driver overload. With those events, underload will affect the driver, being as well a workload state that must be assessed and managed.

In order to better understand which factors affect driver workload, a summary is presented in Table 2.1. These factors affect driver workload either by increasing or decreasing it.

**Table 2.1:** Driver Workload factors [7].

| Driver State factors | Driver Trait factors | Environmental Factors |
|---|---|---|
| Monotony | Experience | Road environment |
| Fatigue | Age | Traffic demands |
| Distraction | Strategy | Automation |
| Drugs | Mood | System's feedback |
| Alcohol | | |

#### 2.1.1.2 Mental Workload, Task Demand and Performance Model

As stated in the literature, mental workload, task demand and performance have an inherent relationship. In 1976, Meister [6] proposed an evaluation model that incorporates those concepts. It was divided in three regions, A, B and C, depending on the task demand, workload and the way performance is affected (Figure 2.1):

- In region A, the task demand is low, as well as the workload and, consequently, the performance is considered at an high level. If, for some reason, in this region, an increase in demand takes place, performance efficiency will not be compromised.
- In region B, performance level decrease due to the increase of task demand, possibly resulting in an increase of workload.
- In region C, performance is dramatically decreased due to the increase of task demand level and high levels of workload.

Regarding the simplicity of this model [6], De Waard [7] proposed an improvement to that model by adding a new region D that is located before the region A. The aim of this deactivation region is to represent the effects and implications of monotonous task performance. Considering its low demand level, monotonous tasks may force an increase of task difficulty by reducing capacity, which happens in monotony and boredom. In order to counter the increase in task difficulty, the operator will apply a greater ability to perform it and, consequently, mental workload will increase.

Despite all the notions regarding how and when the driver *mental workload* increases, the knowledge of how much workload remains unexplored/unmapped. In order to try to solve this knowledge gap, De Waard [7] proposes a decompo-

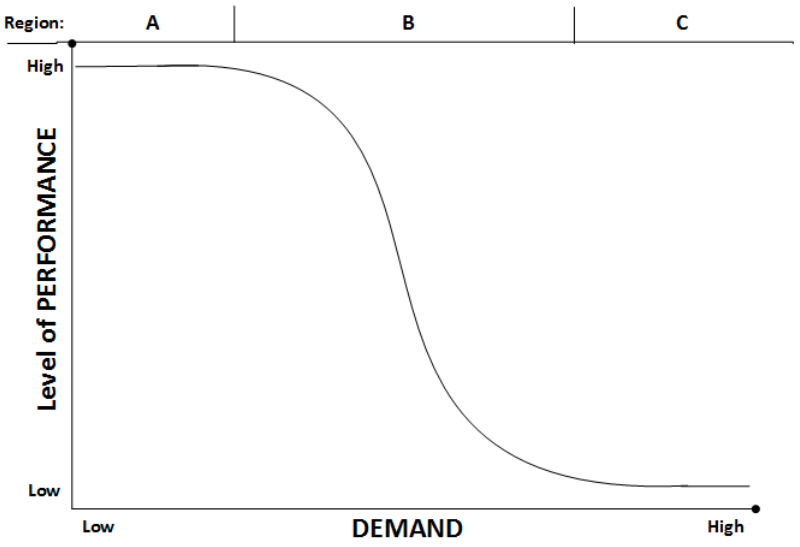



**Figure 2.1:** Mental workload, task demand and performance evaluation model proposed by Meister [6].

sition of the region A in several sub-regions by virtue of delineate a "red line" that represents the exact moment that the decrease performance happens. This proposal divides the region A in three sub-regions: in the middle part of region A2, operator can easily handle with task demand and performance remains at a stable level even when there is an increase in task demand (*i.e.*, there is not an increasing of effort); in sub-region A3, operator is only able to maintain the level of performance by increasing effort, but evaluation measures do not show a decline of performance. A episodic application of extra effort in region A3 is one of the human flexibility advantages and is not considered as critical. However, if effort is continuously required to ensure performance, this can lead to stress and this situation has to be avoided. At this time, there may arise a critical moment, where operator can lose control of the situation (*i.e.*, red line). If effort increases, task demand increases and performance drops, it seems suitable to assume that the critical moment for mental workload is during the transition from sub-region A2 to A3. In turn, the transition from region D to region A1 is associated with monotony practiced by the operator when he undertakes a major effort for not decreasing performance level. Thus, it is through a greater effort that operator does not change his performance. A second workload redline appears at the transition from region A2 to A1, where the operator is effectively counteracting a reduced operator state. When the investment of effort is no longer effective, the D-region is entered where performance is affected [7, 40].

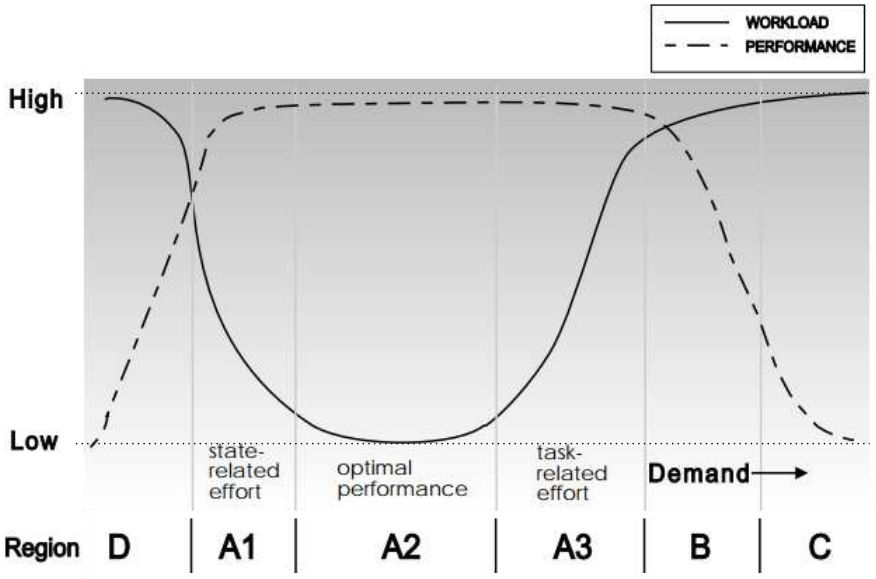



**Figure 2.2:** Mental workload, task demand and performance model divided in 6 regions, proposed by Dick DeWaard [7].

The model representation of task demand, performance and workload is depicted in Figure 2.2 (task demand on the x-axis is not directly linked to region of performance). Task demands relies on the goals that have to be reached by task performance and are not directly related to workload, thus being subjective. Each region in the model just indicates the behavior correlation between workload and task performance. Therefore, the same task can result in performance in region A2 for one person, and for another may require effort compensation and thus region A3 performance for another. Besides, Figure 2.2, presents two types of effort compensation divided in two regions: (i) the A1 region, deactivation is compensated by state-related effort; (ii) the region A3 the compensation is made through task-related effort. However, this model only represents one dimension of mental workload. What is depicted denotes the overall or sum relation between demand, workload and performance.

#### 2.1.1.3 Measurement Tools

A literature review on methodologies for measuring workload shown the need to address the selection criteria for measurement tools [41]:

- **Sensitivity** - if the technique or tool can discriminate between different levels of workload (*i.e.*, is the technique able to reflect changes in workload?);
- **Diagnosticity** - if the technique or tool can distinguish different types of

workload (*i.e.*, how capable is the measure in reflecting demands on specific resources?).

Additional criteria was proposed in [7], regarding primary task interference. Referring that the introduction of a new task can damage performance of a priority task (*i.e.*, the degree to which a technique degrades ordinary or primary task performance).

Regarding the measurement tools in plenitude, the literature propose three workload measurement groups: (i) subjective measure (*i.e.*, self-report measures), (ii) performance measures and (iii) physiological measures [7, 37, 40, 41]. Briefly, the physiological measurement is based on the concept that increased mental demands lead to increased physical response. These changes are measured in signals such as cardiac activity, brain activity, respiratory activity, speech measures and eye activity. The subjective measurement is based on the use of rankings or scales to measure the amount of workload a person is feeling. These measures are devoted primarily to the intermittent question-answer type response to detect varying levels of workload. The two main types of scales used to measure subjective workload are unidimensional and multidimensional scales. Performance measurement relies on examining the capacity of the subject to perform either a primary and a secondary task. By measuring how well a person performs the task, or how their performance worsens with increasing workload, an estimate of mental workload can be determined.

An overview of the aforementioned measures will be given in the following sub-sections, as well as an analysis regarding the relation between measurement tools.

#### 2.1.1.3.1 Self-report Measures

Self-report measures are often known as subjective measures. However, the correct term is self-report since there are other measurement groups that are also subjective (*e.g.*, physiological measures).
Self-report measures consist in the application of measures to the driver, focusing on the perceived level of task demand, and it is widely used in projects and research. These measures are often used by researchers due to the accurate judgment regarding the person's experienced mental load [7]. It is also stated as least intrusive, most flexible, least time consuming, and least expensive form of evaluating workload. However, some may not trust in self-report measures arguing that

physical and mental workload are hard to separate, and the person may be unable to distinguish between what are external task demands and mental effort experienced, and to quantify the mental effort invested [40, 41]. Moreover, some people may not have the ability to detect internal changes [37]. Also as a drawback, they do not provide a continuous form of measurement. Measurements can be taken during the task, but should not affect performance of the primary task performance. For example, if a person needs to complete a NASA-TLX questionnaire [42, 43] while they are driving over a difficult section of road, it violates the safety driving concerns. It was found that it is not necessary to subjectively interview a person during or immediately following the difficult section. However, long delays may interfere with workload score reporting.

Different dimensions of workload, such as performance and effort, are incorporated in self-report measures while at the same time individual fluctuations, operator state and attitude are also taken into account.

Regarding the previous workload model (Figure 2.2), most self-report measures are sensitive to all regions, except region A2. In the A1 and A3 region, ratings of effort could indicate the increase in workload.

The self-report measures are divided in two different topologies: unidimensional or multidimensional. These two topologies are, respectively, related to the evaluation of one or more dimensions concerning mental workload. Unidimensional rating scales are considered the simplest to use because there are no complicated analysis techniques. The multidimensional workload scale is considered to be a more complex and more time-consuming form of measurement, and has from three to six dimensions [44].

### Unidimensional Measures

Unidimensional measures are often considered to be too simple to measure the complexity of workload, but recent research concluded that they outperform multidimensional scales [45]. It is possible to use a single scale to evaluate all tasks, despite their huge diversity in modalities, mental operations and response modes. They are the easiest and least time-consuming to both measure and analyze workload. Some unidimensional measures are briefly explained on the next points:

- The Modified Cooper-Harper (MCH) scale is a 10-point unidimensional rating scale to evaluate the global workload. This study was developed to be a change from the psychomotor Cooper-Harper scale and increase the range of applicability to situations commonly found in modern systems. The MCH scale is used to measure perceptual, cognitive, and communications workload [46, 47].

- The Rating Scale Mental Effort (RSME) scale is a unidimensional scale. Ratings of invested effort are indicated by a cross on a continuous line. The line runs from 0 to 150 mm, and every 10 mm is marked. Along vertical line there are some landmarks identified with a verbal descriptor of effort, ranging from "no effort" to "extreme effort". It should be noted that RSME is increasingly used to assess mental workload on traffic sector [48].

Briefly, although presenting some sensitivity to changes in task difficulty, MCH is not as sensitive measuring mental workload as NASA-TLX (multidimensional scale), or as RSME [40].

#### Multidimensional Measures

Regarding multidimensional measures, is the most widely used and accepted way to assess workload by subjective means. There are currently two main multidimensional measures being used in the real-world and simulated environment, the NASA-Task Load Index (NASA-TLX) scale, and the Subjective Workload Assessment Technique (SWAT). There are also several other scales that are less well known. The multidimensional nature of the scales provide a more in-depth analysis of the many aspects of workload, where the one-dimensional scales cannot. Generally, the multidimensional form of measurement takes more time to complete, so it is hard to use a multidimensional scale during a study. Not only is the gathering of results time-consuming, the analysis takes time as well. In order to better understand how this multidimensional measures work, a brief explanation about the aforementioned measures will be given in the following points:

- The NASA Task Load Index uses six dimensions to assess workload: mental demand, physical demand, temporal demand, performance, effort, and frustration. Twenty-step bipolar scales are used to obtain ratings for these dimensions and score from 0 to 100 is obtained on each scale. This scale uses

a weighting process that requires a paired comparison task. The task requires the operator to choose which dimension is more relevant to workload for a particular task across all pairs of the six dimensions. The workload scale is obtained for each task by multiplying the weight by the individual dimension scale score, summing across scales, and dividing by the total weights. Generally, the NASA-TLX is an extremely good multidimensional scale for measuring mental workload with high sensitivity to changes in workload, however the time needed to complete and analyze the test, presents as a drawback. Recently, a different type of TLX scale was developed called the NASA Raw Task Load Index (NASARTLX). This scale was developed because the collection and analysis of the original TLX scale was cumbersome and labor intensive. The RTLX computes a score by taking the sum of the TLX test and dividing it by six. This new way to score the NASA-TLX was found to be almost equivalent to the original TLX scale with far less time involved for analysis [42].

- The Subjective Workload Assessment Technique uses three levels – low, medium, and high – for each of the three dimensions of time load, mental load, and physiological stress load to assess workload. This multidimensional test uses three steps to complete and analyze workload. The first step is scale development, which combines all the possible combinations of the three dimensions in 27 cards. The person sorts the cards into a ranking that reflects his or her perception of increasing workload. The rankings are used to develop a scale with interval properties. The second step is rating the workload. The third step is to convert the scores into a 0 to 100 scale using the scale developed in step one. When the SWAT scale is compared to the NASA-TLX, the TLX scale is generally considered to be the better scale for measuring mental workload [44].

- The Driving Activity Load Index (DALI), a revised version of the NASA-TLX, is designed to evaluate the workload during the driving task. As in the NASA-TLX, it is a scale rating procedure for six pre-defined factors (*i.e.*, effort of attention, visual demand, auditory demand, temporal demand, interference, and situational stress) that measures the perceptual load, mental workload, and drivers state. To each factor, a six point scale can be attributed, between Low (0) and High (5), and followed by a weighting procedure in order to combine the six individual scales into a global score. The subjects are asked to relate the given driving task to what they consider to

be normal driving conditions. In terms of comparison to the NASA-TLX, the main difference is based in the choice of the main factors that make up the workload score to adapt to the driving context [49].

In this way, it is important to mind that unidimensional scales have been more appropriate when assessing mental workload aims to determine a single general measure [40]. Additionally, if unidimensional scales are used separately in each dimension of the task, they also can provide multidimensional properties [7].

#### 2.1.1.3.2 Performance Measures

Performance measures are based on techniques of direct registration of driver's ability to perform the driving task at a level that is considered acceptable and safe, and also properly maintain the vehicle on the road preserving the safety of all participants in it. Therefore, it may be defined as the effectiveness in accomplishing a task. Generally, it can consider two main ways to measure workload by means of performance: (i) direct measurements (*i.e.*, they are only focused on performance of the main or primary driving task); (ii) indirect measurements (*i.e.*, which associate a secondary task to the primary driving task). The basic assumption for using primary and secondary tasks to measure workload is to assume that people have limited resources. The tasks that demand the same resource structure will reveal performance decrements when time-shared and further decrements when the difficulty of one or both is manipulated. This means that workload can be estimated by measuring the decrease in performance by either the primary or secondary tasks. The primary task measure is a more direct way to measure workload than the secondary task measure, but both are used and at least moderately accepted [7, 40].

#### Primary Task

Primary task performance measurement assesses the workload based on the capability to perform the main task. This is a direct and non-intrusive form of measurement. It must be individually determined for each situation and may include measuring lateral control (steering wheel movements, lateral deviation), lane-keeping behavior, longitudinal control (speed control), and Time-to-Line Crossing (TLC) in driving situations. It was found that steering wheel movements, particularly wheel reversals, are sensitive to changes in workload. Speed has been shown

to decrease as workload increases, it is a sensitive measure, but can be disrupted by changes in traffic. Time-to-Line Crossing is defined as the time required for the vehicle to reach either the center or edge line of the driving lane if no further corrective steering wheel movements are executed, and as mental workload increases, TLC increases. One of the problems associated with strictly using the performance on a primary task is that it does not consider spare mental capacity. For example, two tasks may be performed equally, but one person's mental capacity may be pushed to its limits while another person's mental capacity is not pushed at all. Another problem with using primary performance measures to estimate workload is motivation. When people are more motivated, their workload may increase, but their performance might not increase to the same extent. It is also hard to measure changes to performance due to workload, unless the workload is very high. Changing from a low to medium level of workload probably will not produce a change in performance even though workload is increasing [7].

#### Secondary Task

The secondary task is an additional measure to the primary task. The concept for the secondary task is that it measures the difference between the mental capacity consumed by the main task, and the total available capacity. Poor dual-task performance would suggest competition for many of the same resources, whereas efficient dual-task performance would suggest little resource competition. An advantage of secondary measurement over primary measurement is that it can determine if there is any spare mental capacity. The problem with embedded tasks is that they may not be less important than the primary task. For a secondary task to be used, less importance must be placed on it than the primary task. Artificial tasks like addition or simple mathematical computations are considered good secondary tasks. The problem with artificial tasks is the intrusion factor. Artificial tasks may intrude on other workload measures. The major problem that may occur when secondary tasks are used to measure workload is that they may disrupt primary task performance. Some people may not perform the primary task before they perform the secondary task. This causes problems for measurement of changes in performance of the secondary task. When determining the validity of different measures of workload, it is imperative that the primary and secondary tasks use the same resource. For example, a primary measure that is visual must be coupled with a secondary measure that is also visual to achieve the best measure of performance. Many of the discrepancies between and within the

different measures of workload are due to inaccurate or poor collection of the data. It is important to keep safety in mind when choosing a secondary task because the performance may be degraded to the point that it becomes dangerous when workload is too high [7].

#### 2.1.1.3.3 Physiological Measures

Physiological measures evaluate the answers given by peripheral and central nervous system during driving performance, and may involve, in particular: (a) heartbeat measuring; (b) electroencephalogram; (c) electrocardiogram; (d) evaluation of eye movement and pupil dilation; (e) blood pressure evaluation; (f) assessment of breath levels; (g) assessment of electrodermal activity; and (h) determination of hormone levels. Their main advantage lies in the fact that they do not involve a clear and objective response from the driver, *e.g.*, they do not require an overt response by the operator, and most cognitive tasks do not require overt behavior. Physiological measures are sensitive to changes in mental workload levels before registering clear decrements in driving performance. Moreover, most of the measures can be collected continuously, while measurement is nowadays relatively unobtrusive due to miniaturization. This logic is not always supported since the body also responds physiologically to things other than mental workload. When an increase in mental task difficulty is coupled with increased physical workload, the results may be skewed. Each of the physiological workload measures must be examined individually to find the relation between the physiological responses due to physical activity and mental activity. Most research focuses on five physiological areas to measure workload: cardiac activity, respiratory activity, eye activity, speech measures, and brain activity [7, 40, 50].

##### Cardiac Measure

The most common physiological workload measurement regards the cardiac monitoring. Cardiac measures are often used because they are easy to evaluate and are considered a reliable indicator of workload. Heart rate measurement is the most common and reliable measure of workload by cardiac means. It is an exact measurement since the signals can be measured in the form of beats, which are easily identifiable. Generally, it is assumed that heart rate increases with increasing workload. However, not all studies agree with these conclusions. A percentage

of articles are reluctant of the utilization of heart rate for the workload measurement due to the influence that some factors (*e.g.*, psychological, environmental, and emotional) can have on it [45]. It is important to remember that heart rate varies by individual. When using heart rate as a measurement tool, it is necessary to find a baseline measurement of the heart rate to compare as a reference.

Another cardiac workload measure is heart rate variability (HRV). HRV measures the inter-beat intervals of the heartbeat over time. This measure is not used as extensively as heart rate, but many studies focus on the use of HRV to study workload due to its novelty and promising area of research. When the spectrum is divided into three parts, there is a lower band that is associated with regulation of body temperature from 0.02 to 0.06 Hz, a middle band that is associated with blood pressure regulation from 0.07 to 0.14 Hz, and a high band associated with respiration from 0.10 to 50 Hz. There is not one standard method to measure HRV, although the most common method for scoring HRV encompasses the calculation of the standard deviation or variance of the inter-beat intervals over a given time, or for a given number of beats. On the other hand, some studies shows that speech, respiration, muscle activity, body position, physical fitness, and age can influence the results of HRV. It was also found that psychological factors like physical fatigue may also influence HRV. Another problem with HRV measures is time considerations. Some spectral analysis techniques require a minimum of three to five minutes of data to correctly resolve low frequency components. While driving, this may be a problem because the event of interest may not be long enough to get an accurate measure.

Blood pressure is usually a secondary measurement of workload. It is not widely used because it is a more obtrusive measure than heart rate or heart rate variability. Although blood pressure is found to increase as workload increases, it does not provide any more detailed information about workload than heart rate [51, 52].

### Respiratory Measure

There are several ways to use respiratory measures to find mental workload. Some can be used in real-world settings, while others can only be measured in a laboratory setting. The most common type of respiratory measurement is breathing rate. Other measures include monitoring the volume of air entering and exiting the lungs and measuring the amount of carbon dioxide in expired air. Almost all

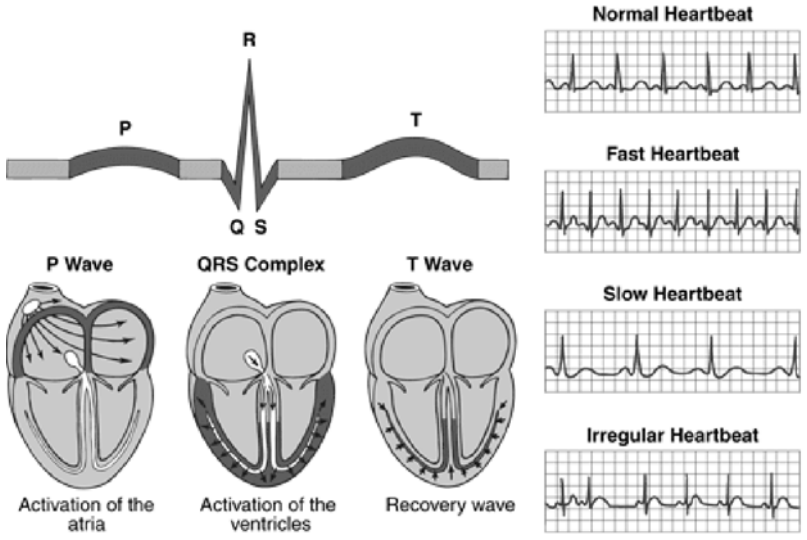



**Figure 2.3:** ECG readings in the form of a $PQRST$ graph or waves. The first wave ($P$ wave) indicates the contraction of the auricles, the second section ($RQS$ wave) represents ventricles contraction and the third $ST$ wave indicates the ventricles relaxation when filling with blood form the auricles. Moreover, it depicts a normal versus abnormal ECG readings [8].

research conducted on respiration uses rate to determine workload. Respiratory rate measures the number of breaths per minute. Measuring breathing rate is a very easy and unobtrusive indicator of mental load. It is generally agreed that an increase in respiratory rate is indicative of increased workload. Respiratory rate has also been used extensively as an indicator of emotional states, stress, arousal, and mental load. However, sensitivity to other factors other than increased mental workload may cause problems in reliability and consistency. Respiration rate influences other measures of workload, for example heart rate variance. Moreover, it is necessary to measure respiration if measuring HRV in order to find a comparison between the two. One problem associated with the measurement of breathing rate is that it can be interrupted by speech. Talking and breathing are interconnected in most real-world situations, so it is hard to apply respiratory rates to situations where speech is involved. Measuring the air volume flow and the amount of carbon dioxide expelled during breathing is not studied as extensively as simply measuring respiratory rate. One reason for not examining the effects of workload on these measures is that it is much harder to calculate the amount of air and carbon dioxide flow than the number of breaths per unit time, without being obtrusive. Most research supports the notion that volume decreases as workload increases. On the other hand, there is conflicting evidence that flow volume is not affected by changes in workload [7].

### Eye Measure

Several measures use physiological changes in the eye to determine mental and visual workload. Despite the fact that the eye is related fundamentally with visual workload, it has been demonstrated that some measures can also precisely predict mental workload regarding some tasks [45]. The measures mostly associated with the eye are horizontal activity, blink rate, and interval of closure. Different measures incorporate eye fixation and pupil diameter. Associated with visual activity is the electrooculogram (EOG) that is a brain activity measure and will be discussed in further ahead. Eye blink rate is the number of eye closures in a given amount of time and the interval of closure is defined as the time spent blinking. Although measuring eye blink rate is easy, the results are mixed. It is generally accepted that eye blinks are good at measuring visual workload. Eye blinks and blink duration decrease with increasing visual workload. Environmental changes may also influence eye blink and its duration. When there are changes in light or air quality, eye blink rate may also change. Several other eye activity measures show promise in measuring visual and mental workload. The most promising is horizontal eye movement (HEM). HEM involves the scanning eye movements that are used to acquire information. Typically, HEM would measure glances at the speedometer, side mirrors, or rear-view mirror. This measure was found to be a good indicator of visual and mental workload, but there are not many studies that examine HEMs to estimate mental workload. As workload increased, it was found that HEMs increase. Pupil diameter may be another good way of estimating mental workload under certain conditions. Pupil diameter is found to increase with increasing mental workload. Eye fixations are another measure used to estimate mental workload and they measure the amount of time the eye spends looking at a selected object [7].

### Speech Measure

Speech measures are rarely studied as tools for measuring workload. One possible reason for not using speech to find workload is that it is difficult to take exact measures of different aspects of speaking. The six measures most often used to measure speech are pitch, rate, loudness, jitter, shimmer, and a derived speech measure. It was found that the three speech measures affected by workload are pitch, loudness, and rate. These three measures all increase as task difficulty in-

creases. The derived speech measure was found to have the most correlation to workload demands. Since not much research is devoted to using the voice to measure workload, the information is not corroborated. This may lead to problems when exact measures are needed to determine workload [53].

### Brain Activity

All the previous physiological means for measuring workload employ indirect means to gather data, since all these data are influenced by signals the brain sends when experiencing different amounts of mental load. The brain is responsible for processing information, making decisions and initiating actions on the external environment. It is generally agreed that the most precise measurement of mental workload comes directly from measuring the activity of the brain, not to mention that it provides good temporal resolution of cognitive activity. Although brain activity measurement does not directly interfere with the task, the gathering of the data may be distracting and intrusive. A major problem with measuring brain activity is that specialized equipment is needed and, in some cases, can be intrusive.

The electroencephalogram (EEG) is by far the most studied and accepted form of workload measurement that uses brain activity. EEG signals are generally classified into four bands: up to 4Hz (Delta waves), 4 to 8 Hz (Theta waves), 8 to 13 Hz (Alpha waves), more than 13 Hz (Beta waves). When there is an increase in mental workload, the EEG shows that the Alpha waves disappear and are replaced by Beta waves. Generally, as mental workload increases, theta waves increases and alpha decreases. One well known problem regarding this type of measure regards to physical movements that may cause problems in the analysis of the EEG. The electrooculogram (EOG) is primarily used for measuring saccadic eye movements. This measure is another form of measuring eye blink rate and eye closure interval. Not much research is presently being conducted on the benefits of using electrooculogram for workload measurement. This may be due to the intrusiveness of the measure. This type of measure seems to be a good indicator of visual workload, but there are not enough studies to determine whether the electrooculogram results agree with other types of visual workload measures like eye blink rate. The EOG is intrusive, and also expensive to implement [54].

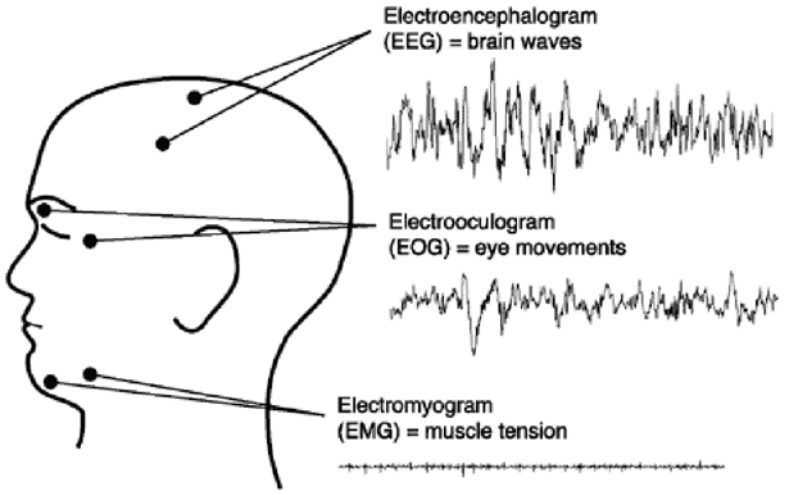



**Figure 2.4:** Representation of the appropriate placement position of electrodes for the measurement of Electroencephalogram, Electrooculogram, and Electromyogram signals [9].

#### Others

There are several brain activity measures that are not studied as extensively as the aforementioned. These measures may hold some promise in mental workload measurement. The use of the electromyogram (EMG) is a new but promising measure of mental workload. The EMG measures task irrelevant facial muscles that are not required in the motor performance of that task. Different facial muscles are found to be differentially sensitive to changes in mental workload. Other changes that deal with physiological changes in parts of the body other than the brain may hold some potential for measurement of workload. Electrodermal activity (EDA) measures electrical changes in the skin. As workload increases, EDA was found to increase. The measure is not very selective because many factors were found to affect EDA. Changes in hormone levels are related to extremely stressful situations. Hormone levels are usually used for long-term studies on workload [7].

#### 2.1.1.3.4 Summary Table

Table 2.2 presented below gathers all the measures that were addressed in this chapter and sums all the information related to them. Moreover, some considerations about it will be taken regarding the use of measures from different groups or types.

**Table 2.2:** Summary and analysis of Workload measures

| Measure | Result of Increased WL | Benefits | Drawbacks |
|---|---|---|---|
| **Heart Rate** | Increases | Widely accepted and studied, easy to measure | (1) May not be completely reliable; (2)Does not measure absolute levels of work |
| **Heart Rate Variability** | Decreases | Some studies indicate better accuracy than HR | (1) Not widely studied or accepted; (2) Influenced by respiration equipment |
| **Blood Pressure** | Increases | Can be used to calculate modulus | (1) Not widely studied accepted; (2) No more information than HR or HRV |
| **Respiratory** | Increases | Rate Easy, unobtrusive, sensitive, reliable | Influenced by emotion, stress, speech |
| **Electro encephalogram** | Alpha waves replaced by Beta waves | Extremely accurate, reliable, catches changes | (1) Obtrusive, requires special equipment; (2)Other measures may miss and training, may not be cost effective |
| **Electro oculogram** | Less jumps in data (Gaze and blink rate) | Accurate for visual measures | Not widely used, obtrusive, requires special equipment and training |
| **Eye Blink Measures** | (1) Rate decreases;(2) Pupil diameter increases | Most accurate for visual workload | Not as accurate as other workload measures |
| **Speech Measures** | Pitch, loudness, and rate increase | Can be used to determine influence on HRV | Not studied, speech not important for all applications |
| **Modified Cooper-Harper Scale** | Higher rating | Fast, fairly accurate | Conflicting opinions, considered hard to use |
| **Overall Workload Scale** | Higher rating | Fast, accurate for unidimensional scale; easy to administer, prepare for and analyze | Mainly used for identifying "chokepoints" |

| Measure | Result of Increased WL | Benefits | Drawbacks |
|---|---|---|---|
| **NASA Task Load Index Scale** | Higher rating | Accurate, valid | Takes a long time to administer and analyze |
| **Subjective Workload Assessment Technique** | Higher rating | May be more sensitive to difficulty increase than TLX | (1) Reports of high failure in analysis of results; (2) No agreement on accuracy or sensitivity |
| **DALI** | Higher rating | Identify the origins of driver workload | Factors are not linked to specific aspects of the driving task |
| **Primary Task** | Decreases | Accurate to changes in workload | Not accurate when low levels of workload |
| **Secondary Task** | Decreases | Finds spare mental capacity better than Primary | May interfere with task, not accepted |

As stated during this chapter regarding each measurement group and technique it is possible to discriminate groups of measures. Accordingly, two groups of techniques to measure mental workload can be differentiated [55]: (i) first group assumes that it is possible to obtain a global measure of mental workload, comparable to single-resource use (*e.g.*, self-report measures, performance measures and physiological measures that are arousal-related); (ii) the other group are diagnostic procedures that are linked to theories of multiple resources (*e.g.*, secondary-task techniques and some of the physiological measures). It is possible that single-resource theories and global workload measures are in many cases applicable, simply because task demands in one dimension predominates. In general, and particularly in most applied settings, measures from both groups are useful [7].

Another important consideration to highlight is that not all measures are sensitive to workload in the same area of performance, and dissociation between measures of different categories was already reported [7]. A dissociation well known is between self-report and performance measures, although having some authors that address another dissociation between self-report and physiological measure [56]. This dissociation between measures happens due to the lack of correlation to changes in workload, or if one measure stats a decrease in workload while the other indicates an increase. One explanation to this phenomenon is a differential sensitivity of different measures to particular sources. For example, performance

is affected by amount of resources invested, by resource efficiency and by competition for a resource, while subjective workload perception is affected by amount of resources invested and by demands on working memory [7].

#### 2.1.1.4 Related Work

The task of a Driver Workload estimation is to identify driver's workload status from the observations of driver's behavior. Some old researches on cognitive workload estimation followed a pattern that, briefly, consist of an initial analysis of the correlation between various features, that relate with human behavior, and the subsequent model's design to generate a workload index by combining features of high correlation. This approach can be referred as a manual DWE design process methodology. Obviously, it can raise some doubts regarding the robustness of the estimator due to the dependence of a strong domain knowledge in the field of human behavior and the inefficiency of a manual data analysis and modeling considering the enormous list of features related to driver's cognitive workload. One alternative approach to this paradigm is the learning-based DWE design process. In other words, alternatively to analyzing manually the meaning of each feature or a small set of features, it can be considered the complete set of features at the same time using machine-learning techniques, in order to adjust the DWE module and obtain an optimized model that will provide the correct workload index.

The effectiveness of machine-learning in discovering the subjacent structure of data and in generating precise models, supported its introduction in this field since experts could not discover effectively, from domain knowledge, such inner information and models. Lately, this is the approach that a huge part of researchers are adopting in the designing of theirs workload estimators.

##### 2.1.1.4.1 Driver Cognitive Workload Estimation: a Data-driven Perspective

Zhang Y., Owechko Y., and Zhang J. [10] proposed a machine-learning-based design process for driver cognitive workload estimation. The learning-based DWE design process conducted in this study is depicted in the Figure 2.5.

One of the most widely used method for inductive inference, the *Decision-Tree*, was used as a learning algorithm in this study, through a decision-tree learning software named *See5* (developed by Quinlan). In order to improve the perfor-

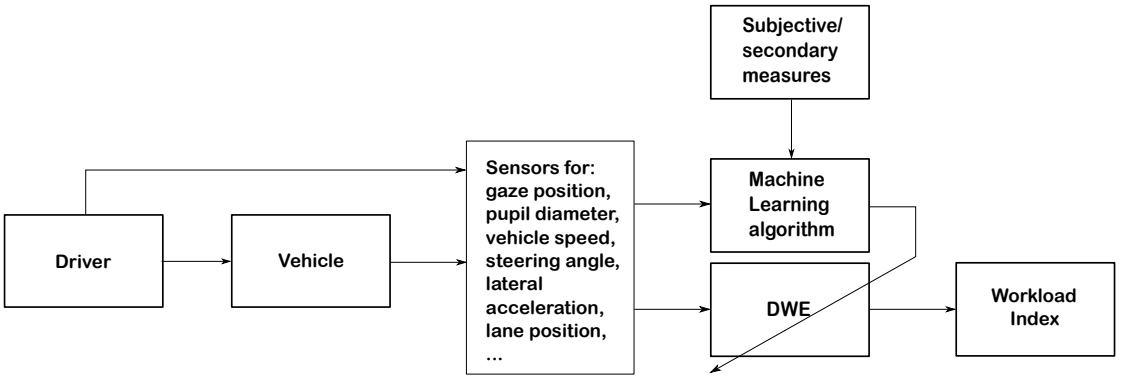



**Figure 2.5:** The learning-based DWE development process [10].

mance, a well-known used training mechanism called *Boosting* was used.

Database for training the machine learning algorithm was composed by simulated vehicle data and driver's behavior data. The simulated vehicle data was collected from a driving simulator supported by General Motors. This simulator was equipped with a non-force feedback *Microsoft Sidewinder* USB steering wheel together with the accelerator and brake pedals.

The data collected from the simulator encompasses vehicle speed and acceleration, steering angle, lateral acceleration and lane position, being transposed and used in terms of means and standard deviations. The driver behavior's monitoring was made using a gaze tracking system, installed in the subject's computer, composed by a remote monocular eye-tracker and a head tracker. This system measures the pupil diameter and the point of gaze.

During the experiment, twelve students participated in the simulation scenarios. Each participant drove the simulator in three different driving scenarios: highway, urban and rural. Each scenario, with 8 to 10 minute duration, was divided in two sessions where the participants were asked to perform secondary tasks (two verbal tasks and two spatial-imagery tasks). These tasks were performed during four different periods of thirty seconds. The verbal task consisted in the construction of words starting with a designated letter. On the other hand, in the spatial-imagery task, subjects were asked to imagine the letters from A to Z with one of the following characteristics: (i) remaining unchanged when flipped sideways, (ii) remaining unchanged when flipped upside down, (iii) containing a close part such as "A", (iv) having no enclosed part, (v) containing a horizontal line or (vi) containing a vertical line. In each session, a 2 minute period was established as a control session, where no secondary tasks were introduced.

Dataset labeling was not done by directly assess the driver's workload, through a subject workload assessment. Instead, based on the assumption that the more

tasks a driver has to handle at a time, the more resources he is consuming and the higher workload he is bearing, the study's authors labeled all the sensor inputs that concern to dual-task periods with *high workload*. Similarly, sensor inputs concerning normal control periods were labeled with *low workload*.

Training was performed with different *See5* decision trees, where each one was trained with various combinations of features. The results showed that the best performance achieved by the algorithm was 81% when using all of features in training. The Eyegaze-related features were concluded to be more predictive when comparing to driving-performance features since the removing of these features from training set reduced the correct prediction rate by only one percentage point, from 81% to 80% (Table 2.4).

Study's authors concluded that the machine learning algorithm provided a good performance when estimating the driver's workload index, considering the difficulty of cognitive workload estimation. Also, since a general machine learning package was used as an inference tool, a customized algorithm could provide a better performance than a general-purpose one.

**Table 2.4:** Correct prediction rates in the task-level training with different features combinations [10].

| Feature Combination | Correct Prediction Rate (%) |
|---|---|
| All features | 81 |
| Eyegaze-related features | 80 |
| All but pupil-diameter features | 70 |
| Pupil-diameter features only | 61 |
| Driving-performance features | 60 |

#### 2.1.1.4.2 Driver Cognitive Workload Estimation using Support Vector Machines

Son et al. [11] suggests several approaches for identifying driver's cognitive workload. The inference model implemented in this study was enforced using a machine-learning technique, based on statistical learning theory: Support Vector Machines.

For the algorithm training, driving performance, physiological response and eye movement data was considered. The driving performance data was collected using a fixed-based driving simulator with a car cab shape. The virtual roadway was displayed on a 2.5 meters by 2.5 meters wall-mounted screen and sensory feedback was provided to the driver through an auditory and kinetic channels.

Such data encompasses driving distance, speed, steering throttle and braking inputs, captured at a 30 Hz sampling rate. The physiological an eye behavior data were, respectively, collected using the MEDAC System (NeuroDyne Medical Corp., Cambridge, MA) and the FaceLAB eye tracking system (Seeing Machines Ltd., Canberra, Australia).

A preprocessing phase took place on the aforementioned data, resulting in six different features from the aforementioned three domains that were selected as SVM models' input features to detect driver's cognitive workload. Standard deviation of lane position (SDLP) and steering wheel reversal rate (SRR) are related to features specified in driving performance domain. Heart rate (HR) and skin conductance level (SCL) are features that represents the physiological domain and the standard deviation of horizontal gaze (SDHG). And lastly, standard deviation of vertical gaze (SDVG) are related to features representing the eye behavior. An experiment was conducted in the previously described simulator and followed the procedure illustrated in Figure 2.6.

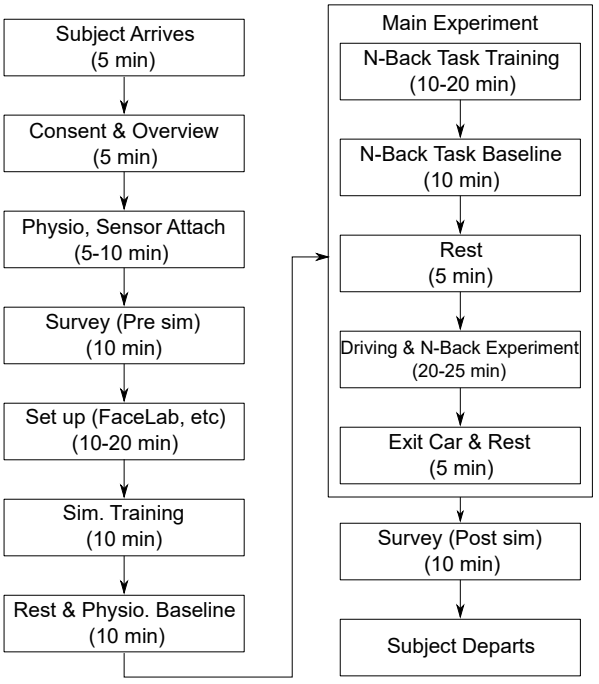



**Figure 2.6:** Experimental setup and procedure [11].

Since one of the study's premises was to consider age differences, subjects for collecting training and testing data were expanded to older drivers. Therefore, some requirements were imposed when selecting subjects: (i) age within 25 to 35 or 60 to 69, (ii) driving frequency at least twice a week, (iii) self-reported good health condition and free from major medical conditions, (iv) not taking medications for

psychiatric disorders, (v) scored 27 or greater on the mini mental status exam for grading the cognitive state [57], and (vi) having not previously participated in a simulated driving study.

After the completion of an informed consent and a pre-experimental questionnaire, subjects spent 10 minutes of driving training and adaptation time in the simulator. Afterward, participants were trained in the secondary task, that would be introduced during the main experiment as a cognitive workload instigator.

A Secondary task applied in this study as an auditory delayed digit recall task that create three workload distinct levels. It requires subjects to speak out loud the *nth* stimulus back in a sequence that is informed via audio recording. For example, the lowest level n-back task is the 0-back, meaning that the subject must repeat loudly and immediately the last item presented. At the intermediate level (1-back), the next-to last stimulus must be repeated. Lastly, the most difficult level (2-back), the second to the last stimulus must be repeated.

The n-back task was presented in four 30-second trials consisting of 20 single digit numbers (0-9) shown in a random order with a 2.1 seconds interval. Followed by 5 minutes rest, subjects drove through a 37 km straight highway in a good weather, during 20 to 25 minutes. For the driving and n-back task experiment, between minutes 5 and 7, a single task driving reference was instructed to the subject. Thirty seconds later, the secondary task (0, 1 or 2-back) was introduced, based on a 18 seconds periods of expert instructions. A two-minute recovery period was provided before presenting instructions for the next task. The order of these three levels of task difficulty was randomly assigned across subjects.

Subsequently to the experiment data collection, the labeling procedure was carried out according to the periods of experience to which the data belong. In other words, the experiment data were categorized based on the category of cognitive workload that the subject, in a specific period, is supposed to hold. Consequently, four cognitive workload categories were established based on the complexity of primary and secondary tasks.

The secondary task, according to the n-back task applied to the subject, may induce different levels of workload and, therefore, can be categorized as follow: (i) the 0-back task, that is a low-level cognitive challenge, was categorized with a low workload level on individuals; (ii) the 1-back task, that requires an additional step up in cognitive load, would have a moderate workload on individuals; and (iii) the 2-back task, that requires the highest cognitive load, was categorized with the high workload level. Although the cognitive workload was classified into four categories (*i.e.*, normal driving and driving with three levels of n-back task) the

study only use two levels of cognitive workload: *normal driving* and *high cognitive workload* condition (*i.e.*, driving while performing the most difficult cognitive task, the 2-back task).

Model training was performed using a 10-second window of experiment data, without overlapping. These data refer to the normal driving and the driving while performing the 2-back task. The segmented data were then normalized using z-score and labeled as 'not distracted' for normal driving condition or 'cognitively distracted' for high cognitive workload condition. The kernel function used to construct the SVM models was the Radial Basis Function (RBF).

In order to train and evaluate the performance of the cognitive workload detection model, a nested cross-validation[1] was adopted.

The evaluation of the performance of SVMs was made using three aspects: classification accuracy, sensitivity and specificity (Equations (2.1), (2.2), (2.3)) [58].

$$Sensitivity = \frac{TP}{TP + FN} * 100 \tag{2.1}$$

$$Accuracy = \frac{TP + TN}{TP + FN + TN + FP} * 100 \tag{2.2}$$

$$Specificity = \frac{TN}{TN + FP} * 100 \tag{2.3}$$

Where:

$TP$ = True Positive
$TN$ = True Negative
$FP$ = False Positive
$FN$ = False Negative

The study reported some promising results regarding algorithm's performance and its influential features. The SVM models' performance varied from 58,9% to 89,0%, depending on the input features combinations (*i.e.*, performance is influence by which input feature or group of input features was used for each SVM model's training).

Regarding the SVM models that used a single input feature, the one that obtained the best accuracy was the heart rate (HR). This model achieved an 80% accuracy in detecting driver's cognitive workload followed by the model with the

[1] Allows the simultaneous optimal selection of parameters of SVMs and the unbiased estimation of the performance of the final SVM model [11].

standard deviation of vertical gaze (SDVG) as a single input feature with an accuracy of 76,7%. The high accuracy rate evidenced by the HR-based model confirmed the HR effectiveness in detecting the driver's cognitive workload, as already reported in early findings.

The SVM models that reported the best performance were the ones with multiple input features (Table 2.5). Among this last models, the combination of physiologic, gaze and SDLP features achieved the best performance with an accuracy of 89%, sensitivity of 86,4% and specificity of 91,7% (the higher the specificity, the less false alarms will be triggered). Also, the combination of all features as input features of the SVM model was not optimal, regarding accuracy and specificity.

**Table 2.5:** Model performance of Single and Multiple input features [11].

| **Single Input** | | | | | | |
|---|---|---|---|---|---|---|
| **Model** | **SDLP** | **SRR** | **SDHG** | **SDVG** | **HR** | **SCL** |
| **Sensitivity** | 62.5 | 67.2 | 82.8 | 71.9 | 77.8 | 75.0 |
| **Specificity** | 55.3 | 76.1 | 64.2 | 81.4 | 82.2 | 72.5 |
| **Accuracy** | 58.9 | 71.7 | 73.5 | 76.7 | 80.0 | 73.8 |
| **Best Combination** | | | | | | |
| **Model** | **Physio. Gaze** | **Physio. Gaze SDLP** | **Physio. Gaze SRR** | **Physio. Driving SDHG** | **Physio. Driving SDVG** | **Physio. Driving Gaze** |
| **Sensitivity** | 85.0 | 86.4 | 86.7 | 83.6 | 85.0 | 87.5 |
| **Specificity** | 91.1 | 91.7 | 90.6 | 90.6 | 89.7 | 90.0 |
| **Accuracy** | 88.1 | 89.0 | 88.6 | 87.1 | 87.4 | 88.8 |

An interesting conclusion concerns the influence of age in the input features and, consequently, in the SVM models' performance. The results showed a better return with younger drivers than with older drivers in the driving performance domain. However, this fact was reversed in the physiological domain. Therefore, from these differing results, the study recommends to use cross-domain measures for improving the robustness against age factors.

Concluding, the study presents a limited data set and a limited driving use cases, being constricted to a straight highway road in a homogeneous traffic scenario. Therefore, a diverse set of scenarios is crucial to fully assess the holistic workload domain.

#### 2.1.1.4.3 Driver Workload Classification through Neural Network Modeling using Physiological Indicators

Hoogendoorn and Arem [59] proposed an neural network model approach to classify and predict driver workload through physiological indicators of driver workload, driver characteristics and characteristics of the driving conditions. Driver's reaction times were used as output of the model representing the driver workload.

Data for the analysis and the algorithm's training was collected through a driving simulator experiment with a two independent groups experimental design. Between groups all variables were kept constant, except for the traffic intensity. The independent variable was varied through the groups as it was assumed that a higher traffic intensity will cause a higher driver workload.

The driving simulator was composed by three screens placed at an 120 degree angle, a driver's seat mockup, and hardware and software to interface the simulation (simulation software was developed by StSoftware). A virtual driving environment consisting of two different segments was developed for the experiment: (i) the first segment encompasses a short test drive on the freeway, aiming to let participants familiarize to driving in the simulator and to test if they were prone to simulator sickness; and (ii) the second segment was used for the experiment. The virtual driving environment consisted of a freeway with three lanes.

In order to provide a more realistic experiment, the virtual freeway was populated with other vehicles. Their speeds were varied depending in which lane they circulated: vehicle in the left lane were programmed to have a average speed greater than vehicles in the right lanes. The only varied variable was the traffic intensity, being set to 2500 veh/hr in the first group and to 5000 veh/hr in the second group.

Physiological and performance measures were chosen as they are driver workload indicators. Heart rate (HR) and heart rate variability (HRV) were measure through a ElectroCardioGram (ECG). The participant's age was collected before each session of the experiment. Also, driving behavior was measured and registered using the driving simulator at a 10 Hz sampling rate. From this data, reaction times were extracted using a car-following model, the Helly model [60], through the estimation approach described in Hoogendoorn and Hoogendoorn [61]. The authors of the study chose this approach because reaction times are not directly observable from data.

Forty four subjects (34 male and 10 female) were recruited for the experiment, within a range of ages between 18 and 65 years old (mean age of 28,89 years). These

subjects were randomly assigned to one of two aforementioned groups, resulting in a balanced distribution between the groups regarding age, driving experience and gender.

The aforementioned neural network was used in this study to classify inputs (*e.g.*, physiological indicators of driver workload, driver characteristics and characteristics of the driving condition) into a set of target workload categories. More specifically, it is a Multiple-layer perceptron neural network(MLP), with a two layer feed-forward network using sigmoid hidden and output neurons. A relatively small number of 10 hidden neurons were set for this neural network.

The training step was performed using a scaled conjugate gradient backpropagation method, that is a variation of the normal gradient backpropagation method and provides a generally faster convergence to the minimum gradient of the error function. The dataset collected in the experiment was divided in three subsets: 70% of data for training, 15% of data for validation and 15% of data used for testing.

The neural network input data encompasses three main features types: (i) driver characteristics, composed by the age of participants, (ii) physiological indicators of driver workload represented by heart rate and heart rate variability, and (iii) traffic intensity, coded with 1 for low traffic intensity while high traffic intensity was coded with 2. The output data encompasses the reaction time, composed by three categories: $T_r \leq 1$, $1 > T_r \leq 2$ and $T_r > 2$. Each one represents low, medium and high driver workload, respectively.

Before proceeding to the network training, the study reports an analysis of the experiment collected data, regarding heart rate and its variability. This analysis showed a noticeable difference between the two groups, finding that the overall heart rate was higher and the heart rate variability was lower in the group with the high traffic intensity. This information indicated that was an increased driver workload in the group with the high traffic level, revealing an association between high traffic level and the increase of driver workload.

Afterward the training, testing and validation phases, the accuracy of the network was analyzed and it achieved a 78% correct classification rate. Also, the model produced the higher classification accuracy for its highest class, when observing the generated confusion matrices and the respective Receiver Operating Characteristic (ROC) plots.

Concluding, the model showed the capability to classify workload with a good performance. However, some limitations have to be pointed out. The limited number of types of physiologic data can, in some way, jeopardize the performance and

the model's correct classification of driver workload. Also, additional driver characteristics can be introduced, such as gender, driving experience, among others. The traffic intensity was the only independent variable that, therefore, was varied. Nevertheless, other factors can express the driving complexity conditions, like road design, secondary tasks, and others. Finally, the study's authors expressed the motivation to switch from a driving simulator experiment to a naturalistic driving observation method, in order to validate the model and obtain a more realistic feedback.

#### 2.1.1.4.4 Bayesian Network Classifiers Inferring Workload from Physiological Features

A workload classifier was proposed in [12]. This study proposes an approach based on Bayesian Networks (BN) in order to estimate the workload of operators, using physiological features and the reaction time from a secondary task as a cognitive measure of workload. Different BN structures were implemented in order to test several combinations of physiological features. The ground truth was provided as a subjective measure collected during the experiment. This classifier was implemented for the aviation domain. However, this study is still relevant due to the many similarities with the automotive domain and its workload estimation approaches.

For the experiment, ten subjects (9 males and 1 female) with a mean age of 30 years (standard deviation of 10,7 years) have participated and with normal or corrected to normal hearing and seeing.

The simulator was composed by a non-force feedback joystick and a 24" monitor, where the experiment simulation was generated by ICE software. Simulation data (*e.g.*, aircraft position) was collected from the simulator at a sampling rate of 100 Hz and physiological data was acquired at a 2048 Hz sampling rate. Stereo headphones were provided to subjects so they could hear the recorded instructions regarding the experiment and also aircraft engine noises.

Experiment script was composed by a predefined flying trajectory made of 60 rings. The subject's goal was to pass through each rings, placed in a varied vertical distance, trying to successfully pass through ring placed in the trajectory. Aircraft's speed was predefined and constant for all trajectories.

Experiment was divided in 5 sessions composed by 6 trials each. The trial's duration was approximately 90 seconds. The 5 sessions imposed to each participant was divided as follows:

- The three first sessions presented to the subject three different trajectories of increasing difficulty. The trajectories of each session for the 6 trials were maintained the same. The only variable independent, and therefore varied, was the trajectory difficulty in order to create different levels of workload upon the subject. This additional cognitive load was introduced by varying the vertical distance between two successive rings, keeping the depth distant.

- In the last two sessions, subjects had to repeat the flight of the simplest and the hardest trajectories of the first and third session, and they were proposed to beat their own scores over these trajectories.

Furthermore, for each of the five aforementioned sessions, a secondary task was introduced in the experiment. This task was composed by two geometrical shapes that were displayed on the screen for 1 second, at random positions and random times (minimum time interval of 1,5 seconds between two successive targets). When a square target appears, the subject must, as quickly as possible, press a button on the joystick and should not react to a triangle target.

Performance data was collected during the experiment realization. This data encompasses the primary task (*e.g.*, percentage of hit rings) and the secondary task (*e.g.*, reaction times and false and true detection rates). Physiological features encompasses heart rate (estimated from the ECG), root mean squares of the flexor digitorum EMG and the right trapezius descendens EMG, respiration measured trough chest expansion, and skin conductance measured using electrodes placed on the subject's fingers.

During the experiment performance, subjects evaluated their own workload level using the NASA Task Load Index questionnaire (TLX). The subjective rates on the six subscales were weighted and summed for each session in order to obtain a single task load index per session.

Before proceeding to the model construction, the collected data was analyzed in order to evaluate its representativity within the problem. Mainly, from this analysis, was perceived the increase of cognitive resources allocation when a secondary task was introduced during the primary task performance. Also, investigating the subjective data it indicates that the TLX scores increase with the session difficulty. Correlating the TLX scores and the reaction times, per session, the study achieved, in most cases, a value greater than 0,5 (Table 2.6). This correlation value indicates that the subjective evaluation of workload is consistent with the objective measure (*i.e.*, reaction time). Some values under the aforementioned one exist due to the withdrawal of some subjects on the secondary task when ex-

periencing high workload, so RT were not available and the relation with TLX scores was no more linear.

**Table 2.6:** Correlation between TLX scores and reaction times over th five sessions, per subject [12].

| **Subject** | **S1** | **S2** | **S3** | **S4** | **S5** | **S6** | **S7** | **S8** | **S9** | **S10** |
|---|---|---|---|---|---|---|---|---|---|---|
| **Correlation** | 0.67 | 0.79 | 0.83 | 0.80 | 0.68 | 0.48 | 0.32 | 0.75 | 0.90 | 0.45 |

In order to prepare data to model's learning phase, the noise in the raw signals was removed using specific filters. Moreover, the first and last seconds of each trial's signal were removed in order to avoid possible starting and ending effects. After this step, data was normalized between 0 and 1. The Shannon's method was used to calculate the entropies of the physiological data that will be inputs for the model. The entropy is a measure of disorder, thus it is expected to acquire differences in physiological data regarding workload variations. These entropies are estimated on 15 second long windows slided with 5 seconds sliding windows. Once again, the entropy values are normalized between 0 and 1, taking the maximal and minimal values that appeared on the three first sessions, per subject.

Despite the variation of the mean entropy features was consistent with the variation of the primary task, the reaction time and the TLX scores, the variation of the physiological data presented an abnormal pattern. The mean entropy values of physiological data decreased with difficulty levels for some subjects while for others it has decreased. Due to this abnormality, the model implementation was be performed for each subject separately (*i.e.*, for each subject, a unique model was trained and tested).

Since a data driven approach, and therefore a Supervised learning method, was applied it requires the data labeling with the ground truth extracted from the subjective measure applied during the experiment. Thus, a relationship between the input features and the model's output can be discovered during the algorithm's learning phase. To this end, three different BN structures are tested, differing in how many physiological features the model would have (one, two or three input physiological features). In total, 25 classifiers were trained and tested with different physiological nodes and implementing the aforementioned 3 different structures (Figure 2.7): (i) the first structure is a naive BN where the TLX is a direct child of the physiological nodes, (ii) the second structure is the same BN but TLX node is a child of the RT that is also a child of the physiological nodes, and (iii) the third structure also is a naive BN but has a more complex structure, where TLX

is a direct child of both physiological nodes and RT node. The model's objective, independently of which structure was being used, was to infer the subject's TLX score on each session.

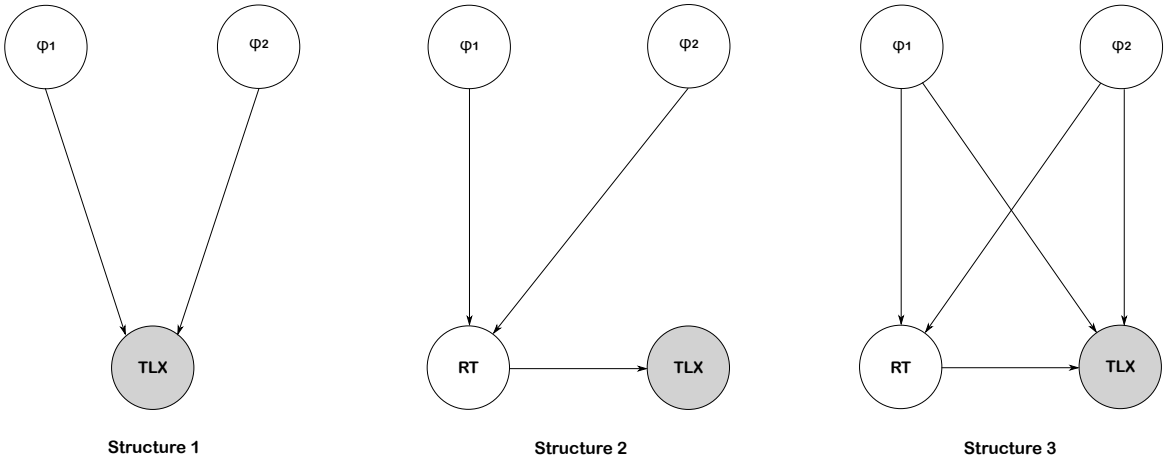



**Figure 2.7:** BN model structures for TLX score inference from physiological features $\varphi_1$, $\varphi_2$ and reaction time (RT) [12].

The algorithm's training was performed using the experiment collected data. For this phase, the training set was composed by data collected on the first three sessions and the testing data set encompasses data from the last two sessions. The learning and inference phases were performed using a Matlab toolbox: the Bayes Net Toolbox. For assess the performance of the models, the study's authors analyzed the differences between the subjects' TLX scores and the predicted TLX scores, for the three first sessions. Since a separately model was implemented for each subject, the study's goal was to achieve the maximum number of subjects to be correctly detected. Therefore, a two-variable optimization problem appears, since model diversity and accuracy were intended to be optimized. The model diversity results from teh percentage of subjects' TLX score correctly detected.

Some results were achieved regarding the two-variable optimization problem. Thus, the best classifier in terms of maximal accuracy was the single physiological (skin conductance feature) node classifier with the first structure. Regarding the maximal diversity, the best classifier was the two physiological (heart rate and skin conductance features) node classifier with second structure (Table 2.7). However, the optimal model is the one with a good performance in both criterion. The model with the best performance regarding accuracy and diversity was the three physiological (hear rate, right trapezius descendens EMG and skin conductance features) node classifier with the second structure. Its accuracy reached the 60% and the diversity a 0,66 score.

**Table 2.7:** Performance of the best model combination regarding accuracy and diversity independently [12].

| Performance criterion | Structure | Physiological nodes | $S$ *(Diversity)* | $\theta$ *(Accuracy)* |
|---|---|---|---|---|
| **Maximal Accuracy** | 1 | SC | 20 | 0.89 |
| **Maximal Diversity** | 2 | HR and SC | 80 | 0.47 |

The study concludes that the best model was the three physiological node classifier with the second structure. However, each one of the five proposed physiological features appear in the classifiers with a good performance. Moreover, the model's performance increases with the number of physiological inputs. Therefore, it can be concluded that these physiological features provide information related to workload and a model implemented with all these features would outperform the study's proposed classifier. Another conclusion suggests that the inclusion of the reaction time in the model yield to a better workload prediction, although creating a more task-dependent method due to the introduction of a secondary task. Finally, the study claims that a test on new subjects must be performed in order to check if the best classifier, that shows a good diversity performance, remains performing well.

#### 2.1.1.4.5 Classifying Driver Workload using Physiological and Driving Performance Data

The study [13] proposes a series of machine learning classification algorithms and perform a evaluation on their ability to identify elevated cognitive workload levels in drivers. Moreover, a comparison between heart rate, skin conductance and driving data is performed for classification. Models for inferring the workload level across individuals and for each person were implemented, through a large two field studies. The purpose of model's training across individuals was to demonstrate the potential of building models that can generalize and work with new drivers. The final goal with the classifier algorithm is to evaluate IVIS and technology using the cognitive workload detection. From this, moments of interest can be identified during task performance in order to apply countermeasures regarding the not intended increase of cognitive workload.

Two on-road driving experiments were conducted for each field study. In

both studies, subjects drove on an interstate highway and vehicle performance and physiological data were collected during the experiment. Besides the normal primary driving task, subjects were asked to complete secondary tasks that cause elevated cognitive workload. In the first experiment, 20 subjects participated and the goal was to classify individual driver's workload level by building exclusive models for each subject. The second experiment, 99 subjects participated aiming to build classification models across participants, reducing the need for training on each subject.

The secondary task was introduced in both field studies and was employed to impose additional mental workload while driving. It was composed by an auditory presentation with verbal response delayed digit recall task, namely n-back task. A single number between 0 and 9 is displayed, one at a time with 2,25 seconds intervals between digit displays. For each digit display, subjects must say out loud the digit two items back in the current sequence (2-back task). This task demands an attentive auditory perception and the corresponding cognitive processing involving working memory.

For the two field experiment data collection, the vehicle was equipped with embedded vehicle sensors for a synchronized data record. Vehicle data was logged at 10 Hz and physiological data at 250 Hz. Different vehicles were used for each field study. The vehicle data encompasses the driving speed, steering wheel position, and acceleration data, collected from the vehicle's CAN bus. Physiological data was composed by data from a electrocardiogram (EKG) accordingly placed on the subject, and skin conductance measures.

Before proceeding to the model construction, data collected from the experiment must be processed in order to create the model's input features. After processing the raw signal coming from the EKG, heart rate (HB) and heart rate variability (HRV). However, due to prior work [62] regarding this physiological measure, only the HR was chosen because it was found to be more robust than HRV during driving and performing a secondary task that increases driver's cognitive workload. The skin conductance was preprocessed with a filter in order to remove the high frequency noise component of the original signal. Lastly, the steering wheel reversal measures the frequency of its reversals exceeding a certain threshold angle.

For the feature generation, a sliding-window approach was used in order to preserve information about temporal dynamics and to aggregate features. Then, these features (*i.e.*, heart rate, skin conductance level and vehicle velocity) were computed in mean, standard deviation, minimum, maximum and first derivative.

Moreover, the number of small and large steering wheel reversals was computed. In order to choose which window length and overlap factor provides the best workload information and enables a higher classifier algorithm's performance, sliding windows of 10, 15, 20, 25 and 30 seconds were used with overlap percentages of 0%, 25%, 50% and 75%.

As mentioned in the introduction of this analysis, the study used five feature based machine learning algorithms: decision tree, logistic regression, 1-nearest neighbor, Multiple-layer perceptron neural network(MLP), and naïve Bayes. These algorithms were chosen because they are simple learners, their generated models can be interpretable and are incremental learners (*i.e.*, they can be retrained with new data without any model change). A supervised learning was applied and, therefore, a labeled dataset was built. Data was labeled as *elevated* workload during cognitive demand task periods, and labeled as *normal* workload if belonging to driving only periods.

The first field study looked at building individual models to account for individual differences between drivers. Twenty six subjects were recruited to participate in this first experiment. These participants had to drive more than three times a week, have a valid driver's license for at least three years and do not report any accidents for the past year, in order to be able to participate. The experiment was composed by an approximately 10 minutes of urban traffic driving, until reaching an interstate highway. Then, additional 20 minutes of interstate driving were provided to subjects in order to familiarize with the vehicle and the environment, followed by a 2 minute single task driving reference period. Subsequently, 24 task periods were presented to subjects and consisted of the 2-back cognitive demand task while maintaining the primary task of driving. Following this 30 second task period, a 90 second recovery and baseline periods were provided to subjects (Figure 2.8). During the experiment performance, heart rate, skin conductance, speed and steering wheel data were being collected.

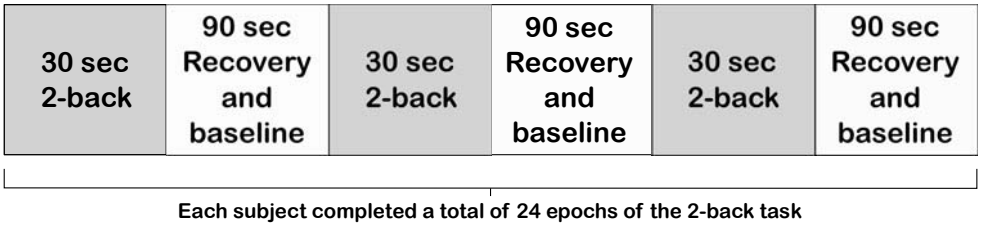



**Figure 2.8:** Experimental procedure for the first field study [13].

Since the first field study was interested in individual classification methods, 13 separated datasets were built for each subject and classification was performed

on each dataset. Each subject dataset was composed by twenty four 30-second samples of elevated cognitive workload where task periods were performed, and twenty four 30-second samples of normal cognitive load representing the middle of recovery and baseline periods.

In order to evaluate the algorithms approaches, a ten-fold cross validation was implemented. Moreover, an inner ten-fold cross validation process was applied within the training set in order to choose the best window size and overlap that yielded the highest algorithm accuracy. Each dataset was divided in training set and test set, being the training set split into training and validation set.

The mean accuracy and standard deviation for each classifier are shown in the Table 2.8.

**Table 2.8:** Mean and standard deviation for classification of elevated workload from normal driving across 13 subjects using all features and across 20 subjects using heart rate only [13].

| | **All Features** | | **Heart Rate** | |
|---|---|---|---|---|
| | **Mean** | **S.D.** | **Mean** | **S.D.** |
| **Decision Tree** | 75.0 | 10.8 | 72.8 | 12.8 |
| **Logistic Regression** | 75.5 | 10.9 | 73.9 | 11.3 |
| **Multilayer Perceptron** | 75.7 | 10.9 | 74.0 | 12.4 |
| **Naïve Bayes** | 75.0 | 12.5 | 74.1 | 11.8 |
| **Nearest Neighbor** | 69.4 | 11.6 | 71.5 | 10.3 |

Using the Trukey-Kramer test, the study concludes from the results that, for all features, the Nearest-Neighbor classifier had a significantly worse performance when compared with the other four algorithms. Concerning the heart rate only, the Trukey-Kramer test showed that the Nearest-Neighbor classifier had a significantly worse performance when compared with logistic regression, multilayer perceptron and naïve Bayes, except decision tree.

Regarding the first field study, the results showed that a reasonable good classification accuracy was achieved. Moreover, with only the heart rate, the accuracy did not decreased substantially when comparing with the all features approach, showing the high capacity of representing workload changes in drivers.

The second field study went towards the building of general classifiers that detect high cognitive workload without extensive training on individual drivers. Data was gathered from 99 participants (age between 20 and 69 years) with the objective of discovering common features and algorithms that dependably can classify cognitive load automatically crosswise over drivers. These participants had to meet some requirements, such as having a healthy condition, driving more

than three times a week, having held a valid driver's license for at least three years and a driving record free of accidents for the past year. In a first moment of the experiment, 146 subjects were recruited. However, due to heavy traffic, poor weather conditions, poor measurement quality in at least one signal domain or missing data, 47 cases were excluded from the final dataset. The age and gender between subjects were balanced in order to represent a real world scenario.

The experiment encompassed a 10 minutes driving in urban traffic, until reaching an interstate highway. Then, a 20 minutes driving familiarization period was provided prior to a 2 minute single task driving period where reference data was established. Subsequently, three task periods were presented to subjects as a series of four secondary task segments each (Figure 2.9). These three task periods were composed by 0-back, 1-back and 2-back task, respectively. However, only the 2-back task periods were classified as *elevated workload* and, therefore, contributed to the model's build. After each task period, a 2 minute recovery period was introduced to subjects.

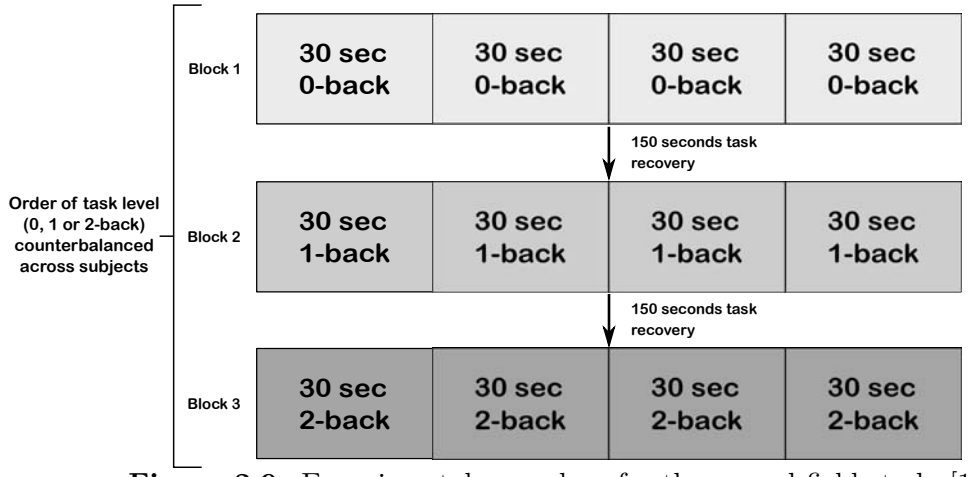



**Figure 2.9:** Experimental procedure for the second field study [13].

Considering that the second field study was involved in evaluate cognitive workload classification approaches across individuals, similar processing and classification methodologies used in the first field study were reapplied. Nonetheless, a dataset from 99 participants was used and the classification was performed across subjects. Moreover, only the set of four series of 2-back task periods and the single task driving periods were used per individual. Sliding windows and overlap factors aforementioned were also taken into account to see the effects that these parameters could have on classification accuracy.

The results of the second field study showed different features' influences on algorithm's accuracy. Training classification algorithms with features generated

from driving performance measures only led to an average classification accuracy of 64% (*i.e.*, all classification algorithms had a similar average accuracy). Investigating the influence of the heart rate features only, and ignoring skin conductance features as well as driving data, except for the 1-Nearest Neighbor, all other learning algorithms achieved an average accuracy of 80%. Applying all features from physiologic data (*i.e.*, heart rate and skin conductance level), logistic regression outperformed other classifiers obtaining the highest performance (average accuracy of 90%) using a 30 second sliding window. The multilayer perceptron neural network and naïve Bayes achieved a slightly lower performance with an average accuracy of 89%. The best classification accuracy came from the physiology and driving performance approach, being similar to the one obtain using only physiology features. Logistic regression, naïve Bayes and multilayer perceptron neural network achieved a better accuracy than all other classifiers.

The trade-off between window size and classification was analyzed in order to find a possible relation. Therefore, was found that increasing the window size improved classification in all the possible learning algorithm approaches, except for the case where the vehicle telemetry features were uniquely applied. The window size takes an important role when applied to real-time classifications where it sizes imply lag (*i.e.*, the bigger the window size, the more information will be gathered and later will be the classification operation). However, regarding the overlap factor, no significant impact was noticed on the classification accuracy.

Through the experiment some considerations were taken regarding the development of automatic cognitive workload classifier in real-world driving. The study found a trade-off between window size and classification accuracy (the bigger the size, the better the algorithm's accuracy), not finding any significant effects of window overlap. Moreover, the second field study proved that physiologic features, specifically heart rate features, can represent and classify very well the elevated cognitive workload. The study wants to examine in the future the possibility do train the algorithm with a large dataset and then verify if it would be able to classify new system interactions across a different set of drivers when they enter the vehicle. Furthermore, an hybrid approach could be implement so that, after being trained with a large dataset, when a new driver entered the car, it would spend only a short time providing additional specific individual training data in order to fine-tune the model. Lastly, since the only elevated workload periods that were classified belongs to 2-back task periods, the study could use the other 0-back and 1-back task periods in order to classify other levels of cognitive demand.

#### 2.1.1.4.6 Driver State Estimation by Convolutional Neural Network using Multimodal Sensor Data

A driver state algorithm that uses multimodal vehicular and physiological sensor data is proposed by Lim and Yang in [14]. This study estimates states of driver drowsiness, visual distraction, cognitive distraction, and high workload of multiple subjects using sensory data collected from a driving simulator. The driver state algorithm was developed based on deep learning and using physiological and vehicle data collected in the driving simulator environment. Contrary to what is normally done, multimodal data was fused into a two-dimensional matrix rather than treating types of data differently. This matrix is composed by different sensor data in one dimension and on the other is used for time. For performance evaluation, both correct detection rate (CDR) and false alarm rate (FAR) criteria were used and the algorithm's performance was compared with the performance of the previous study that implemented a driver state estimation algorithm using a dynamic Bayesian network model [63]. Although the study addresses various abnormal driver states, only the high workload state will be focused during this analysis.

Human-in-the-loop experiment was conducted to collect data for the algorithm's training. This experiment was divide into 3 sub-experiments, each one varying one of the independent variables and aiming to induce abnormal states on drivers: drowsiness, distraction and high workload. The high workload experiment encompasses interaction with other vehicles, such as frequent sudden stops of the lead car or cars driving in the same section. Therefore, the independent variable was the interaction with the lead car.

A total of 16 subjects were recruited for the high workload experiment, with ages between 25 and 29 years old (average age of 27.5 years and a standard deviation of 1.23 years). Subjects were divided in two groups, depending on either or not they carried out the driver high workload induction task. Both groups performed the same driving section and each subject consummated the experiments in both groups. The experiment procedure encompassed driving along a section of road in a downtown area at a velocity of 40 km/h without changing lanes, for approximately 450 meters. The downtown area was composed by complex elements, like billboards, high buildings, foggy weather and more. During the experiment, the only primary task given to subjects was the interaction with the lead car, while avoiding a collision with the lead car that was making frequent and sudden stops. It was assumed that the atypical states happened amid all study sections in

which a task had been assigned. Throughout the experiment, vehicle, vision, voice and physiological data were gathered, using a 30 Hz sampling rate. The vehicle data encompasses vehicle's velocity, longitudinal acceleration, lateral acceleration, steering wheel angle and gas pedal angle. Vision data included participant's audio amplitudes. The physiological data combined subject's heart rate, respiration rate, galvanic skin response and body temperature. The resulting data was splitted intro training, validation and test set, where training and validation dataset were used to train the model and the test set to analyze the its performance.

The deep learning based driver state detection algorithm implemented in the study was the Convolutional Neural Network (CNN). Its layers were implemented using rectified linear units (ReLU), max-pooling, dropout and softmax regression. CNNs are neural network designed to handle large input spaces. This network was implemented by alternatively arranging convolutional and pooling layers. The CNN was used in this study due to the intuition that some pattern in time and between multimodal data could exist. Moreover, this approach prevents the data's overfitting and reduces the computational cost.

Forty four model input features collected from sensors were structured in a 2-dimensional matrix where the other dimension represented the number of samples in 0.1 seconds of time slots. The CNN encompassed two convolution layers followed by max-pooling layers. For each convolutional layer, ReLu was used as an activation function. The output of the second layer was connected to a fully connected layer with a 0.5 dropout rate. In order to obtain the probability of the driver being in normal state or one of the abnormal states (*i.e.*, drowsiness, visual distraction, cognitive distraction and high workload), a softmax regression node was implemented (Figure 2.10).

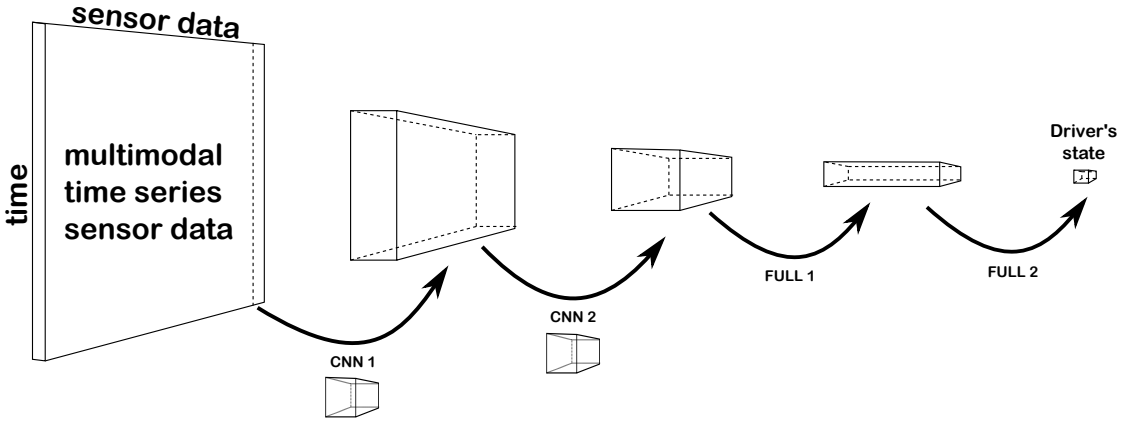



**Figure 2.10:** Convolutional Neural Network model [14].

Before proceeding to the algorithm's training, 4 hours of driving data collect from each experiment was divided into training and test sets. Each input was labelled concerning to which driver state it was gathered. In order to implement and train the CNN model, the Google's TensorFlow application interface was used [17].

The algorithm's performance was assessed applying the CDR and FAR as performance metrics. Therefore, the estimation's average CDR values were 0.966, 1.000, 0.987 and 0.841 fro drowsiness, visual distraction, cognitive distraction and high workload, respectively. Regarding average FAR values, 0.007, 0.000, 0.089 and 0.158 were respectively the values for each driver abnormal state. The Table 2.9 shows the algorithm's performance.

**Table 2.9:** Convolutional neural network model performance results [14].

| **Driver's state** | **CNN** | |
|---|---|---|
| | **CDR** | **FAR** |
| Drowsiness | 0.966 | 0.007 |
| Visual distraction | 1.000 | 0.000 |
| Cognitive distraction | 0.987 | 0.089 |
| High workload | 0.841 | 0.158 |
| **Computation time (ms)** | 0.74 | |

One algorithm characteristic that make it practical for use as an in-vehicle system is the fact that the inference process takes less than a millisecond to be computed. This is an big advantage when comparing with other algorithms with a bigger computational cost.

Concluding, the CNN model was implemented as planned using multimodal data. The accuracy of the proposed driver state detection method can be considered as high enough for the domain. Although it is not the main focus of this analysis, the study CNN model approach showed a significant improvement when comparing with the results of the previous research using dynamic Bayesian network model. As a future improvement, the study authors want to apply the deep learning approach to field data gathered in a real driving scenario.

### 2.1.2 Machine Learning Concepts

The Machine Learning concept will be a significant part of the work developed within this thesis. It will provide some important tools that will allow to design and implement the decision-making centers that will be essential for the

right performance of the sub-systems that compose the assess and manage driver's workload. Therefore, it is important to introduce some concepts regarding this thematic and to explain some available algorithms, in order to build knowledge and to harden off with these concepts.

Machine learning is a type of artificial intelligence that provides computers with the ability to learn without being explicitly programmed. It explores the study and construction of algorithms that can learn from and make predictions on data. Such algorithms provide the capability to a program to dynamically change its behavior by making data driven predictions or decisions, through building a model from sample inputs [64]. These "changes" might be either enhancements to already performing systems or initial synthesis of new systems. Machine learning is employed in a range of computing tasks where designing and programming explicit algorithms is unfeasible or the system needs adaptation to its surrounding context.

The process of machine learning is similar to that of data mining, analyzing data in order to find relevant patterns. However, instead of extracting data for human comprehension, in the case of data mining applications, machine learning uses that data to detect patterns in it and adjust program actions accordingly [65]. Machine learning algorithms are often categorized as being (i) Supervised or Unsupervised, (ii) Active or Passive Learners, and (ii) Online versus Batch Learning Protocol. Supervised algorithms can apply what has been learned in the past to new data, instead Unsupervised algorithms can draw inferences from datasets. Regarding Active and Passive Learners, an active learner interacts with the environment at training time, while a passive learner only observes the information provided by the environment [66]. Finally, the Online Learning Protocol defines that the learner has to respond online, basing its decision in the learning experience and throughout the learning process. In the other way, the Batch Learning Protocol has to engage the acquired expertise only after having a chance to process large amounts of data [66].

Some machine learning algorithms, models and concepts will be described in the next sub-sections in order to have a general view of some available machine learning techniques.

#### 2.1.2.1 Naïve Bayes Classifier Algorithm

Naïve Bayes (NB) algorithm is a classification technique based on Bayes' Theorem that relies on the independence of assumptions between predictors. Briefly, a NB

classifier infers that one particular feature in a class is unrelated to the presence of any other feature [67]. For example, a fruit may be considered to be an apple if it is red, round, and has a specific diameter. Regardless if these features have a correlation with each other or upon the existence of other features, all of these features independently contribute to the probability that this fruit is an apple, regardless of any possible correlations between the color, roundness, and diameter features.

Depending on the type of probability models, NB classifiers can be very efficiently trained when in a supervised learning category. NB model is particularly suited when the dimensionality of the inputs is high. Generally, in practical applications, parameter estimation for NB models uses the method of maximum likelihood [68]. Despite their naïve design and apparently oversimplified assumptions, NB classifiers have worked quite well in many complex real-world situations and is widely used because it often outperforms more sophisticated classification methods, such as boosted trees or random forests [69]. NB classifiers are highly scalable, requiring a linear number of parameters (*i.e.*, features/predictors) in a learning problem.

The advantages and disadvantages regarding the NB classifier algorithm encompasses the following points:

- **Advantages:**
  - It is easy and fast to train and to classify the test dataset;
  - Requires a small number of training data to estimate the parameters necessary for classification;
  - Not sensitive to irrelevant features;
- **Disadvantages:**
  - If a given class and feature never occur together during the process of training, then the frequency-based probability estimate will be zero. This will lead to the cancellation of all information in the other probabilities since these zero estimation will be multiplied with them. Moreover, the model will be unable to make a prediction. This is often known as "Zero Frequency" [70];
  - Is a bad estimator, so the probability outputs from prediction are must not be considered;

- Assumes independence of features, meaning that it can't learn interactions between features. In real life, it is almost impossible that we get a set of predictors which are completely independent.

#### 2.1.2.2 Artificial Neural Network (ANN)

Artificial Neural Networks are processing algorithms that are inspired by our present knowledge of biological nervous systems, although they do not realistically follow it in every detail (*i.e.*, much simpler structure). An Artificial Neural Network consists of an huge group of simple processing elements that take internal decisions and have a large number of weighted connections between these elements [71]. They contain a series of mathematical equations that are used to simulate biological processes such as learning and memory [72]. These elements act as brain neurons-like nodes that co-operate to perform the desired function, operating in parallel. The representation of knowledge is distributed over the element connections and the acquisition of knowledge is obtained through a learning process. The learning methods still have to be programmed, however, and for each problem, we must choose a suitable learning algorithm keeping the same general approach in mind.

As aforementioned, one of the main characteristic of ANNs is the ability to learn, figuring out how to perform their function on their own and determine what to do based only on sample inputs [73]. Another feature is the capability to generalize its decision, producing reasonable outputs for inputs that it had not been taught how to behave before.

Like other machine learnings alternatives, the Artificial Neural Network has some advantages and disadvantages that can have a weight when choosing the best suited machine learning algorithm [72]:

- **Advantages:**
  - Requires less formal statistical training to develop;
  - Models can implicitly detect complex non-linear relationships between independent and dependent variables;
  - Can be developed using multiple different training algorithms.

- **Disadvantages [74]:**
  - Are a "black box" and have limited ability to explicitly identify possible causal relationships;
  - Models may be more difficult to use in the field;
  - Modeling requires greater computational resources;
  - Are prone to overfitting;
  - Model development is empirical, and many methodological issues remain to be resolved.

#### 2.1.2.3 Convolutional Neural Network (CNN)

A Convolutional Neural Network (CNN) is a neural network which has at least one layer that does a convolution between its input and a configurable kernel that generates the layer's output. In a simplified definition, a convolution's goal is to apply a kernel (*a.k.a*, filter) to every point in a tensor and generate a filtered output by sliding the kernel over an input tensor. An example of the filtered output is edge detection in images. A special kernel is applied to each pixel of an image and the output is a new image depicting all the edges. In this case, the input tensor is an image and each point in the tensor is treated as a pixel which includes the amount of red, green and blue found at that point. The kernel is slid over every pixel in the image and the output value increases whenever there is an edge between colors.

CNNs follow a simplified process matching information similar to the structure found in the cellular layout of a human's striate cortex. As signals are passed through a human's striate cortex, certain layers signal when a visual pattern is highlighted. For example, one layer of cells activate (*i.e.*, increase its output signal) when a horizontal line passes through it. A CNN will exhibit a similar behavior where clusters of neurons will activate based on patterns learned from training (*e.g.*, after training, a CNN will have certain layers that activate when a horizontal line passes through it). In the context of CNNs, these patterns are known as filters or kernels and the goal is to adjust these kernel weights until they accurately match the training data. Training these filters is often accomplished by combining multiple different layers and learning weights using gradient descent.

A simple CNN architecture may combine a convolutional layer, a non-linearity layer (*e.g.*, Rectified Linear Unit layer), a pooling layer (*e.g.*, Max pooling layer)

and a fully connected layer. Without these layers, it is difficult to match complex patterns because the network will be filled with too much information. A well designed CNN architecture highlights important information while ignoring noise.

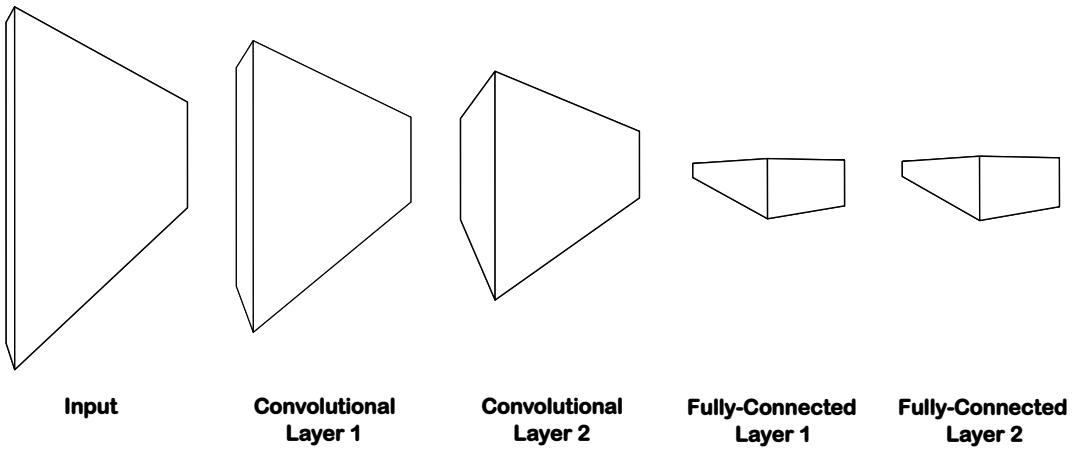



**Figure 2.11:** Convolutional Neural Network base structure example.

Convolutional neural networks, as other machine learning structures, have some advantages and disadvantages associated and these can be found in the following points:

- **Advantages:**
  - Fewer memory requirements - same coefficients are used across different locations in the space, so the memory required is drastically reduced;
  - Easier and better training - using the standard neural network that would be equivalent to a CNN, because the number of parameters would be much higher, the training time would also increase proportionately. In a CNN, since the number of parameters is drastically reduced, training time is proportionately reduced;
  - Parameter reduction - CNNs use the same shared weights through the same layer of hidden neurons.
- **Disadvantages:**
  - High computational cost;
  - For complex tasks, can be quite slow to train without a good GPU;
  - Need of a considerable amount of training data.

#### 2.1.2.4 Bayesian Networks

Bayesian networks (BNs) (or a belief networks) is a probabilistic graphical model that represents a set of variables and their probabilistic dependencies. Formally, Bayesian networks are directed acyclic graphs whose each node in the graph represents a random variable, while the edges between the nodes represent probabilistic dependencies among the corresponding random variables [70]. These conditional dependencies in the graph are often estimated by using known statistical and computational methods. Therefore, BNs combine principles from multiple fields: graph theory, probability theory, computer science, and statistics. BNs belong to the family of probabilistic graphical models (GMs), known as be used to represent knowledge about an uncertain domain [75].

There are efficient algorithms that perform inference and learning in Bayesian networks. BNs have two well known types regarding its purpose and computation methodology: (i) networks that model sequences of variables (*e.g.*, speech signals or protein sequences) are called Dynamic Bayesian networks and (ii) generalizations of BNs that can represent and solve decision problems under uncertainty are called Influence Diagrams [70].

Some advantages and disadvantages regarding the BNs encompasses the following points [76]:

- **Advantages:**
  - Is intuitively easier for a human to understand direct dependencies and local distributions than complete joint distribution;
  - Efficient and principled approach for avoiding the overfitting of data;
  - Handling of incomplete data sets (due to variable encoding);
  - Predicts result of an intervention before intervening.
- **Disadvantages:**
  - Acyclic nature of BNs. The acyclic property is required to carry out probability calculus, but implies that feedback effects cannot be included in the network;
  - It is computationally intensive, specially for models involving many variables. Regarding a large dataset with many variables being estimated, it may very well be prohibitively computationally intensive;

- Despite Bayesian networks can deal with continuous variables, it is done in a limited manner. Therefore, the usual solution is to discretize the variables and build the model over the discrete domain. However, discretization can only capture rough characteristics of the original distribution and statistical power can be loosed.

#### 2.1.2.5 Deep Learning

Deep learning is a branch of Machine Learning, which was introduced with the goal to approximate Machine Learning to one of its original goals: the Artificial Intelligence [77]. It is based on a set of algorithms that attempt to model high level abstractions concerning data by using a deep graph with multiple processing layers, composed of multiple linear and non-linear transformations (*i.e.*, deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction) [78]. In other words, allows the computer to build complex concepts out of simpler concepts. Therefore, it introduces representations that are expressed in terms of other simpler representations.

Deep learning discovers complex structures in large data sets by using the backpropagation algorithm to indicate how a machine should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layer. [79]

Some representations are better than others at simplifying the learning task (e.g., face recognition or facial expression recognition). One of deep learning's assumption is to replace handcrafted features with efficient algorithms for unsupervised or semi-supervised feature learning and hierarchical feature extraction.

Many deep learning algorithms have been designed to tackle unsupervised learning problems, but none have truly solved the problem in the same way that deep learning has largely solved the supervised learning problem for a wide variety of tasks [80].

The Deep Learning complex architecture and its lifetime confers the attribution of some advantages and disadvantages. These are the followings:

- **Advantages:**
  - Good classification performance on problems that significantly outperforms other solutions in multiple domains;
  - Feature engineering process reduced (one of the most time-consuming

task in machine learning);

- Ease of adaptability to different problems (e.g. Vision, time series, etc).

- **Disadvantages:**
  - Requires a large amount of data, which is unlikely to outperform other approaches;
  - Extremely computationally expensive to train;
  - The learned output is not easy to comprehend when comparing to other classifiers (e.g. decision trees).

#### 2.1.2.6 Machine Learning Frameworks

In this section, an overview of the most used and known machine learning frameworks will be presented. Moreover, rather than just present its features and how user can interface with the frameworks, its advantages and disadvantages will be discriminated for comparison purposes.

##### 2.1.2.6.1 TensorFlow

TensorFlow [17, 22] is an open source software library for machine learning and is the result of years of lessons learned from creating and using its predecessor, *DistBelief*. Developed by *Google* to meet their needs for systems, TensorFlow is capable of building and training neural networks to detect and decipher patterns and correlations, analogous to the learning and reasoning processes which humans use. It was made to be flexible, efficient, extensible, and portable. Computers of any shape and size can run it, from smartphones all the way up to huge computing clusters. It comes with lightweight software that can instantly transform a trained model into a product, effectively eliminating the need to re-implement models. *TensorFlow* embraces the innovation and engagement of the open source community, but has the support, guidance, and stability of a large corporation. It is currently used for both research and production at *Google* products, often replacing the role of its closed-source predecessor, *DistBelief*. *TensorFlow* was originally developed by the *Google Brain* team for internal *Google* use before being released under the Apache 2.0 open-source license on November 9, 2015. Due to the popularity *TensorFlow* has gained, there are improvements being made to the library on a daily basis- created by both *Google* and third-party developers.

Its popularity increased at a fast pace, being the most forked *Github* project in just two months since its release. As can be seen in Figure 2.12, it emerged brutally comparing with other frameworks. Its increasing popularity lead world-wide companies like ARM, DeepMind, Airbus, Ebay or Xiaomi to use *TensorFlow* in their products and platforms.

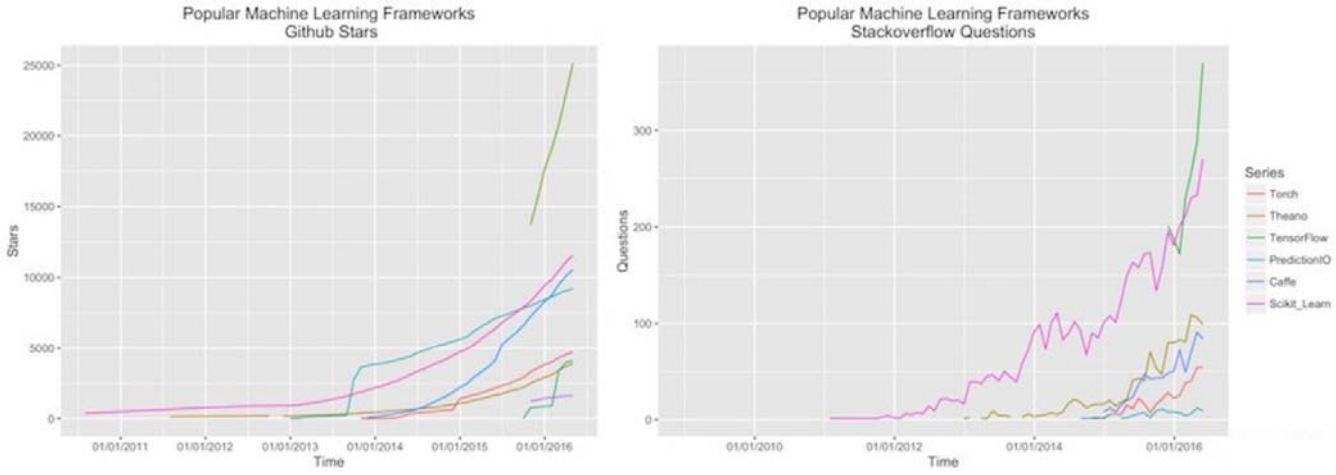



**Figure 2.12:** Graphical statistics regarding the popularity and the questions addressed to TensorFlow [15].

Instead of calling itself a "library for machine learning", it uses the broader term "numerical computation". *TensorFlow* computations are expressed as stateful dataflow graphs. While *TensorFlow* does contain a package, "learn" (a.k.a. "*Scikit* Flow"), that emulates the one-line modeling functionality of *Scikit*-Learn, it is important to note that *TensorFlow*'s primary purpose is not to provide out-of-the-box machine learning solutions. Instead, it provides an extensive suite of functions and classes that allow users to define models from scratch mathematically. This allows users with the appropriate technical background to create customized, flexible models quickly and intuitively. Additionally, while *TensorFlow* does have extensive support for ML-specific functionality, it is just as well suited to performing complex mathematical computations.

While the reference implementation runs on single devices, TensorFlow can run on multiple CPUs and GPUs (with optional CUDA extensions for general-purpose computing on graphics processing units). TensorFlow is available on 64-bit Linux, macOS, Windows, and mobile computing platforms including Android and iOS. Moreover, the core of *TensorFlow* is written in C++, which introduces a very low overhead, and encompasses different front-ends for specifying the computation, as Python, C++ or Java (however, the Python front-end is the best

optimized one). Figure 2.13 provides an overview of the holistic *TensorFlow* architecture as well as how the execution platforms are placed regarding it.

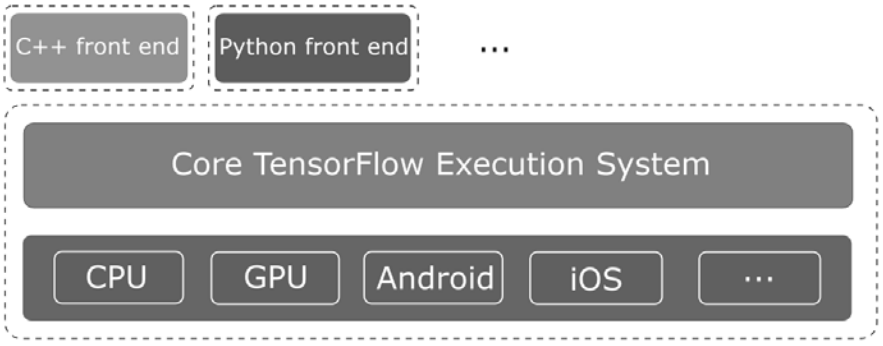



**Figure 2.13:** *TensorFlow* modular architecture with multiple front-ends and execution platforms [16].

*TensorFlow* also provides different types of additional software that can help to debug the model and to deploy the trained model, TensorBoard and TensorFlow Serving respectively.

Some of the pros and cons regarding *TensorFlow* are discriminated in the following points:

- **Pros:**
  - Diverse language options, with an easy Python based interface;
  - Auto-differentiation, by enabling the user to define the computational architecture of predictive models combined with objective functions, and handle complex computations;
  - Faster compile times than *Theano*;
  - Has extra software, *TensorBoard*, for model visualization and debugging;
  - Highly flexible, by enabling users to write their own higher-level libraries on top of it by using C++ and Python, and express the neural network computation as a data flow graph;
  - Portable, giving the option to run on varied CPUs or GPUs, and even on mobile computing platforms. It also supports Docker and running via the cloud.

- **Cons:**
  - Slower than other frameworks;
  - Much "fatter" than *Torch*;
  - Not many pre-trained models;
  - Computational graph is pure Python, therefore slow;
  - No commercial support;
  - Drops out to Python to load each new training batch;
  - Not very toolable;
  - Dynamic typing is error-prone on large software projects.

### TensorBoard

TensorBoard [22] is a web-based graph visualization software that is included with any standard TensorFlow installation and gives the possibility to inspect and understand TensorFlow runs and graphs. When a user includes certain TensorBoard-specific operations in TensorFlow, TensorBoard is able to read the files exported by a TensorFlow graph and can give insight into a model's behavior. The computations used with TensorFlow - like training a deep neural network - can be complex and confusing. This suite of visualization tools make it easier to understand, debug, and optimize TensorFlow programs (Figure 2.14).

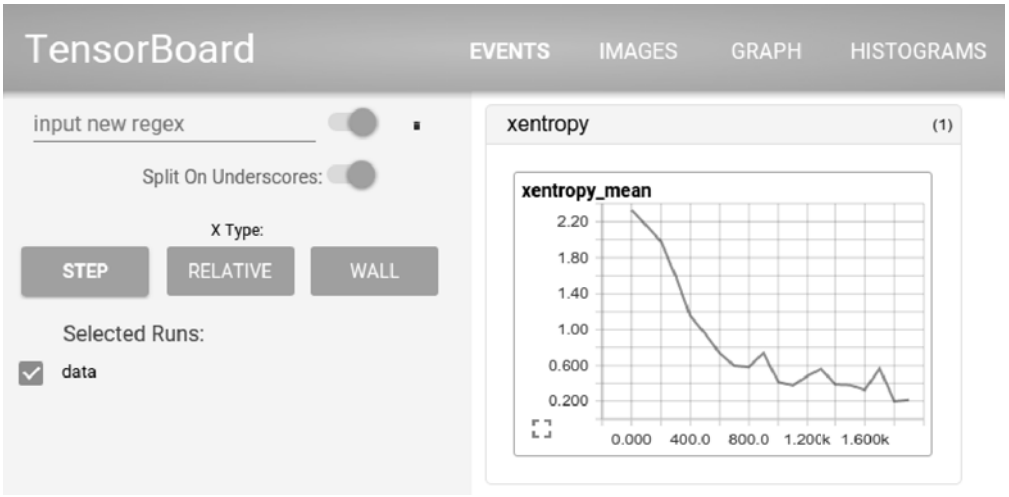



**Figure 2.14:** TensorBoard web application interface [17]

**TensorFlow Serving**

TensorFlow Serving (TFS) [81] is a software that facilitates easy deployment of pre-trained TensorFlow models. Using built-in TensorFlow functions, a user can export their model to a file which can then be read natively by TensorFlow Serving. It is then able to start a simple, high-performance server that can take input data, pass it to the trained model, and return the output from the model. Additionally, TensorFlow Serving is capable of seamlessly switching out old models with new ones, without any downtime for end-users. While Serving is possibly the least recognized portion of the TensorFlow ecosystem, it may be what sets TensorFlow apart from its competition. Incorporating Serving into a production environment enables users to avoid reimplementing their model, who can instead just pass along their TensorFlow export. TensorFlow Serving is written entirely in C++ , and its API is only accessible through C++.

TensorFlow Serving makes it easy to deploy new algorithms and experiments, while keeping the same server architecture and APIs.

#### 2.1.2.6.2 Caffe

Caffe [82] is a deep learning framework, originally developed by Yangqing Jia as part of his PhD at U.C. Berkeley and now maintained by U.C. Berkeley's Vision and Learning Center (BVLC). It is open-source, under a BSD license. Its core is written in C++ and provides a Python interface. Caffe is a well-known and widely used machine-vision library that ported *Matlab*'s implementation of fast convolutional networks to C and C++. Caffe is not intended for other deep-learning applications such as text, sound or time series data.

Caffe is both "cleaner" and more extensible, where models and optimization are defined by configuration without hard-coding. Switch between CPU and GPU by setting a single flag to train on a GPU machine and then deploy to commodity clusters or mobile devices. Moreover, this framework is recommended for research experiments and industry deployment due to its speed. Caffe can process over 60M images per day with a single NVIDIA K40 GPU. That is 1 ms/image for inference and 4 ms/image for learning and in more recent library versions and hardware are faster. Another marking feature about Caffe is the community. It already powers academic research projects, startup prototypes, and even large-scale industrial applications in vision, speech, and multimedia.

A summary of pros and cons regarding Caffe is presented in the following

points:

- **Pros:**
  - Good for feed-forward networks and image processing;
  - Good for fine-tuning existing networks;
  - Capability to train models without writing any code;
  - Python interface.
- **Cons:**
  - Need to write C++ / CUDA for new GPU layers;
  - Not good for recurrent networks;
  - Cumbersome for big networks (GoogLeNet, ResNet);
  - Not extensible;
  - No commercial support;
  - Decreasing use due to slow development.

Looking further to the future of Artificial Intelligence, and encompassing a more light-weight and scalable structure is the new version of Caffe that was already released, *Caffe2* [83], on April, 2017.

#### 2.1.2.6.3 Theano

Many academic researchers in the field of deep learning rely on Theano [84], one of the most known deep-learning framework, which is written in Python. Theano is a library that handles multidimensional arrays, like Numpy. It is an open source project released under the BSD license and was developed by the LISA group at the University of Montreal, Canada. Theano uses C++ as native code and efficent native libraries in order to run all structures and models created by the user, in CPUs or GPUs. Used with other libs, it is well suited to data exploration and intended for research.

The actual syntax of Theano expressions is symbolic, which can be hard to beginners that are used to normal software development. Specifically, expressions are defined in the abstract sense, compiled and later actually used to make calculations.

It was specifically designed to handle the types of computation required for large neural network algorithms used in Deep Learning. It was one of the first libraries of its kind (development started in 2007) and is considered an industry standard for Deep Learning research and development.

Numerous open-source deep-libraries have been built on top of Theano, including *Keras* [85], *Lasagne* [86] and *Blocks* [87]. These libraries attempt to layer an easier to use API on top of Theano's occasionally non-intuitive interface.

The following points described some of the pros and cons regarding the Theano framework:

- **Pros:**
  - Interface in Python.
  - Tight integration with NumPy (use of NumPy arrays in Theano-compiled functions).
  - Transparent use of a GPU
  - Computational graph has a nice abstraction.
  - RNNs fit nicely in computational graph.
  - Extensive unit-testing and self-verification.
  - High level wrappers (*e.g.*, *Keras*, *Lasagne* and *Blocks*) help the development in Theano.
- **Cons:**
  - Raw Theano is low-level.
  - Error messages can be unhelpful.
  - Large models can have long compile times.
  - Much heavy than *Torch*.
  - Patchy support for pre-trained models
  - Problems in deployment to Amazon Web Services.
  - Single GPU deployment.

#### 2.1.2.6.4 Other Frameworks

In addition to all the aforementioned machine learning frameworks, there are many others that are also known in the community of machine learning enthusiasts, such as *Keras* [85], *CNTK* [88], *Torch* [89], *MXNet* [90] and *SciKit Learn* [91]. In order to better understand which capabilities each one of the aforementioned frameworks provide, a brief explanation will be presented in the following paragraphs.

**Microsoft Cognitive Toolkit**

The Microsoft Cognitive Toolkit, previously known as *CNTK* [88], is a unified deep learning framework developed by Microsoft Research. It describes neural networks as a series of computational steps via a directed graph. The Microsoft Cognitive Toolkit can be used to speed-up progresses in areas such as speech and image recognition and search relevance on CPUs and NVIDIA GPUs, and is available on *GitHub* via an open-source license. Its main characteristic is the ability to scale efficiently across multiple GPUs and multiple machines on massive data sets. The Microsoft Cognitive Toolkit can be included as a library in Python, C++ programs and, recently in the 2.0 version, Java or used as a standalone machine learning tool through its own model description language (*BrainScript*). It is used by companies like *Skype*, in its translator, and implemented the *Cortana* digital assistant.

**Keras**

Keras [85] is an open source high-level neural networks API, written in Python and capable of running on top of either TensorFlow or Theano machine learning frameworks. It was developed with a focus on enabling fast experimentation with deep neural networks, and it focus on being minimal, modular and extensible. Keras supports both CNNs and RNNs, as well as combinations of the two. Moreover, it runs seamleassly on both CPU and GPU. Keras was developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System), and its primary author and maintainer is François Chollet, a *Google* engineer.
In 2017, Google's TensorFlow team decided to support Keras in TensorFlow's

core library. The Keras' primary author explained that it was conceived to be an interface rather than an end-to-end machine-learning framework. It presents a higher-level, more intuitive set of abstractions that make it easy to configure neural networks regardless of the backend scientific computing library. Microsoft just implemented, in 2017, a CNTK backend to Keras.

### Torch

Torch [89] is a scientific computing framework with wide support for machine learning algorithms that puts GPUs first. It provides a wide range of algorithms for deep machine learning and is easy to use and efficient, thanks to an easy and fast scripting language, LuaJIT, and an underlying C/CUDA implementation. Its main features relies on the popular neural network and optimization libraries which are simple to use, while having maximum flexibility in implementing complex neural network topologies. It provides tools to build arbitrary graphs of neural networks, and parallelize them over CPUs and GPUs efficiently. Another important feature is the portability, being embeddable with ports to iOS, Android and FPGA backends. Torch comes with multiple community-driven packages in machine learning, computer vision, signal processing, parallel processing, image, video, audio and networking among others, and builds on top of the Lua community.

### MXNet

MXNet [90] is an open-source deep learning framework used to train, and deploy deep neural networks. It is scalable, allowing for fast model training, and supports a flexible programming model (both imperative and symbolic programming). MXNet is characterized for the performance based on the optimized C++ backend engine that parallelizes both I/O and computation. Moreover, it provides multiple languages as front-ends with the same performance (C++, Python, Julia, Matlab, JavaScript, Go, R, Scala, Perl, Wolfram Language). The MXNet library is portable and can scale to multiple GPUs, multiple CPUs, servers, desktops or mobile phones. It is supported by major public cloud providers including Azure and AWS.
Currently, MXNet is supported by Intel, Dato, Baidu, Microsoft, Wolfram Research, and research institutions such as Carnegie Mellon, MIT, the University of Washington, and the Hong Kong University of Science and Technology. It was

developed and is maintained by collaborators from multiple universities and companies.

#### Scikit Learn

Scikit-learn [91] is an open source machine learning library for the Python programming language. It provides various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to cooperate with the Python numerical and scientific libraries NumPy and SciPy, respectively. Briefly, it presents simple and efficient tools for data mining and data analysis.
The scikit-learn project was created as scikits.learn, a Google Summer of Code project by David Cournapeau. The original code base was later rewritten by other developers. Of the various scikits, scikit-learn as well as scikit-image were described as "well-maintained and popular" in November 2012. Until now, 2017, scikit-learn is under active development.

#### 2.1.2.6.5 Summary

During the present section, machine learning frameworks were detailed and its pros and cons presented. Therefore, the Table 2.10 summarizes the frameworks and their features in order to be possible to have a comparison source and to easily understand their differences. Moreover, this table can be used as a reference to justify the application of one of the aforementioned frameworks during the implementation phase.

**Table 2.10:** Machine learning frameworks summary table. [23, 24]

| Framework | License | Core Language | Front-end Languages | GPU support | Graph Optimization | Graph construction |
|---|---|---|---|---|---|---|
| ***Caffe*** | BSD | C++ | Python, C++, Matlab | Yes | - | Static |
| ***CNTK*** | MIT | C++ | Python, C++, C#/.NET, Java | Yes | - | Static |
| ***MXNet*** | Apache 2.0 | C++ | Python, C++, Julia, Matlab, JavaScript, Go, R, Scala, Perl, Wolfram | Yes | - | Static |
| ***TensorFlow*** | Apache 2.0 | C++ | Python, C++, Java | Yes | Supported | Static |
| ***Theano*** | BSD | Python | Python | Yes | Supported | Static |
| ***Torch*** | BSD | Lua, C | Lua, LuaJIT, C, C++, CUDA | Yes | - | Static |

## 2.2 Industry Concepts and Solutions

In this section, different industry solutions and concepts already developed are mentioned and described. This allows a better understanding of what was done or is being doing regarding this thesis field of study.

### 2.2.1 Motorola Driver Advocate

The Driver Advocate is a driver assistance interface developed by Motorola and installed in a Chrysler Town & Country minivan, for testing purposes. Its main vision is to orchestrate the driver's attention to the most important task (*i.e.*, driving) in a useful and acceptable way (Figure 2.15).

To achieve this, the Driver Advocate monitors all driver-related controls taking into account the information originated from telematics, navigation and chassis control systems [18]. Its purpose is to design and develop an intelligent controller system capable to integrate, prioritize, and process the information from different sensors and devices using artificial intelligence technologies (*i.e.*, Machine Learning), and output the result through a multimodal user-interface [92]. The artificial intelligence is capable to learn driver models and learn how to give advises in order to get the best reaction form the driver, without distracting the driver.

Driver Advocate's goals are, obviously, correlated with its vision and aims the following assumptions: (i) improve driving safety (*i.e.*, enhancing driver situational awareness), (ii) reduce driver's distraction (*i.e.*, directing driver's attention

to critical tasks), and (iii) alert the driver to potential road hazards.


**Figure 2.15:** Representation of the Driver's task demand interfaces [18].

Driver Advocate system has a Workload Management Center that processes the information about the driver's attention and stress levels, measured by driver actions and vehicle motion information, and determines if a message should be displayed to the driver or if non-safety-compromising information should be intercepted in order to not disturb the driver from its driving task.

The interface between the driver and the HMI system is possible through a user-friendly three-button installed on the steering wheel, allowing to access to incoming cell phone calls, navigation system and vehicle diagnostic information [93]. If the Workload center decides that the context does not requires a message suppression, the information is normally displayed. Moreover, the driver can determine when is the appropriate time to display the respective message/information (*e.g.*, the driver may want to dismiss the navigation system until reaching a specific part of a route).

### 2.2.2 SAVE-IT - SAfety VEhicle using adaptive Interface Technology

SAVE-IT project is a project funded by the United States Department of Transportation, having Delphi as the prime contractor together with the University of Michigan and Iowa, Ford, General Motors and Seeing Machines as subcontractors. Its goal is to conduct research and develop a new technology that reduce driver distraction, and resulting crashes, induced by telematics devices (*e.g.*, IVIS

and ADAS). The project's main goal is to assess the potential safety benefits and driver acceptance of a system encompassed with adaptive safety warnings and distraction mitigation strategies [30].

The SAVE-IT system is composed by two main subsystems: Adaptive Warnings system and Distraction Mitigation system (Figure 2.16). Physically, this system integrates several interconnected subsystems that monitor the driver, the driving environment, and the in-vehicle systems in order to minimize driver distraction, driving demand, workload, and safety during the driving task. The integration of these three components induce the capability to the system to direct and redirect the driver attention to the roadway when the situation demands it (*i.e.*, imminent crash situations) and accordingly warn the driver when needed [19].

Adaptive Warning system main goals [19] regard the annoyance reduction, crash reduction potential improvement or minimization degradation, and affordability of the technology.

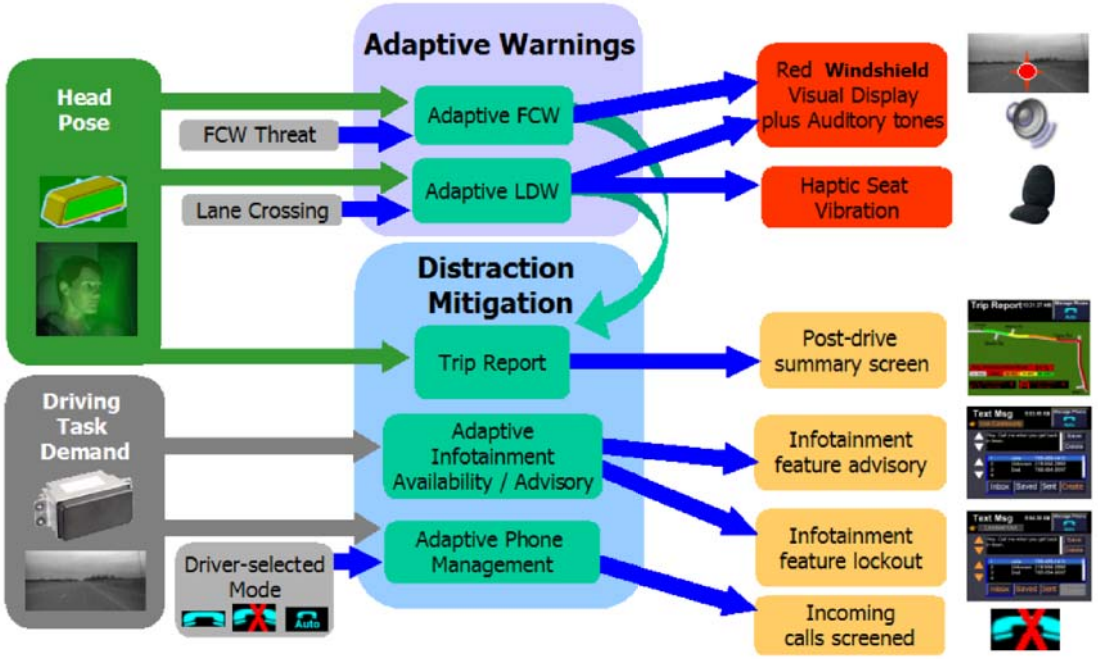



**Figure 2.16:** Representation of the SAVE-IT system [19].

Distraction Mitigation system [19] is responsible for the Infotainment Availability/Advisory adaptation, the Phone Management adaptation and the Trip Report management. These multitude of systems are responsible for the infotainment system adaptation and the phone communications management, based on the actual Driving Demand (*i.e.*, driver workload estimation coming from radar, yaw, path, wipers, etc), providing advertisements in the IVIS and blocking the number

of features that are available for the driver to interact. For example, if the driving demand is medium many IVIS features are available and many others are advised against its use. Finally, the Trip Report Management is responsible to provide to the driver a concise review after each drive.

### 2.2.3 AIDE - Adaptive Integrated Driver-vehicle interface

The Adaptive Integrated Driver-vehicle interfacE (AIDE) is an integrated project funded by the European Commission in the 6th Framework Programme. It was a 50-month project involving 31 partners from the European automotive industry, and a range of leading research institutes and universities [2]. It addresses behavioral and technical issues related to automotive Human-Machine Interface (HMI) design, with a particular focus on integration and adaptation. The project's basis involves an integrated empirical research approach, driver-behavior modeling, and methodological and technological development.

The AIDE's goals somehow correlates with the issues that the P689 project where this thesis is incorporated. Its main goal is to generate knowledge and develop the methodologies and Human-Machine Interface technologies mandatory for a safe and adequate integration of multiple ADAS and IVIS functions into the in-vehicle environment [94]. Precisely, its goal is to design, develop and validate an Adaptive Integrated Driver-vehicle Interface that assures the following assumptions:

- Maximize the efficiency of individual and combined advanced driver assistance systems, preventing negative behavioral effects (e.g. under-load, over-reliance and safety margin compensation) and enhancing the safety benefits of these systems;
- Reduce the workload level and distraction related to the interaction with in-vehicle information and nomadic devices;
- Enable the potential benefits of new in-vehicle technologies and nomadic devices in terms of mobility and comfort, without compromising safety.

AIDE project's features are very similar to the ones presented by the P689 project and, in some points (*e.g.*, forth item of the list of items presented below), with this thesis proposal. These features are the following [2] (Figure 2.17):

- Multimodal HMI interface devices shared by different ADAS and IVIS (*e.g.*,

head-up displays (HUD) and speech input/output);

- Centralized intelligence for decision-making (*e.g.*, by means of information prioritization, scheduling and HMI personalization);
- Integration of nomadic devices into the on-board driving environment;
- Adaptivity of the HMI system to the current driver/driving context (*e.g.*, traffic environment or driver workload level);
- Reconfigurability of the adaptive interface regarding the different drivers' characteristics and preferences.

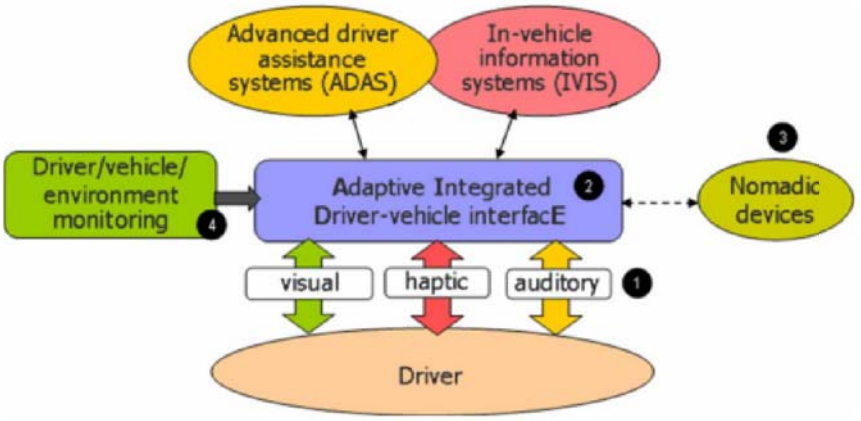



**Figure 2.17:** Illustration of the AIDE concept [20].

# Chapter 3

# System Specification

The past chapter presented and exposed the key concepts and tools that are related to this master's thesis thematic. Thus, this chapter will firstly outline the development environment which supports the implementation of the systems that are specified also within this chapter. Moreover, Chapter 3 describes the experimental setup designed to collected data in order to build the training dataset. Lastly, it specifies the preprocessing phase responsible for the dataset balancing and normalization, preparing it to the training phase.

## 3.1 System's Architecture

The thesis' proposed system is part of a bigger architecture that composes the P689 project as mentioned in earlier chapters. This project's architecture encompasses multiple modules that, holistically, provide and manage HMI features (*e.g.*, navigation, weather information, music, etc) through different modalities (*e.g.*, touch, auditory, visual, haptic, aromatic, etc). Different architecture's layers were outlined in order to give meaning to system modules that represent different hierarchical positions comparatively to the main goal of providing HMI features for the driver (Figure 3.1):

- **Data Layer:** Includes the modules responsible for providing raw data to the information layer modules.
- **Information Layer:** This layer includes modules that provide the Driver Vehicle Environment (DVE) variables.

- **Context Layer:** Includes Information that characterizes the context of each DVE entity, and employs mechanisms for relevant scenario identification.
- **Decision Layer:** The Decision layer includes modules responsible for managing and coordinating the HMI system and the vehicle.
- **HMI Layer:** This layer represents the modalities and channels that allows the user to interact with (*e.g.*, visual, touch, voice, gestures, etc).

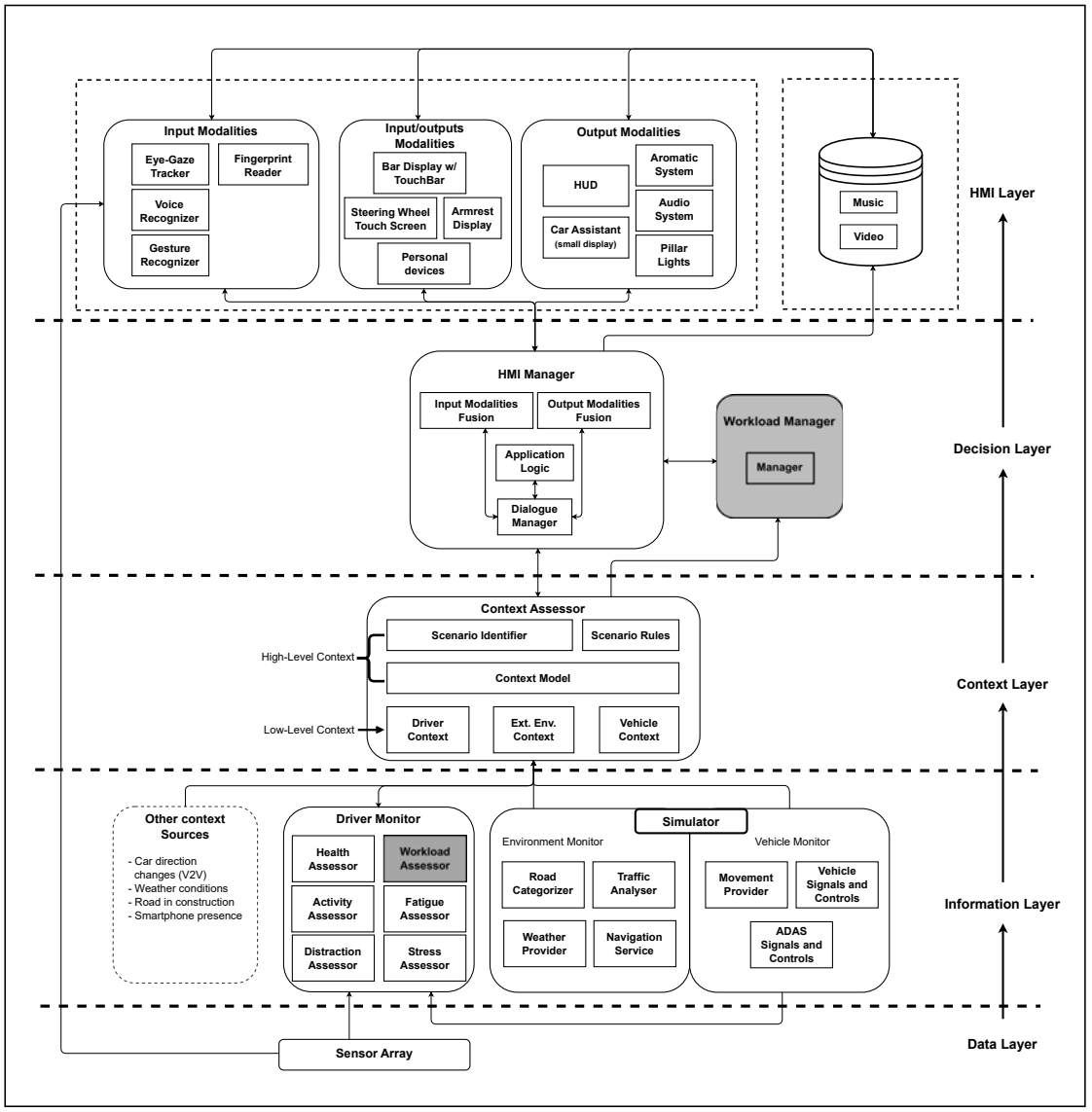



**Figure 3.1:** Project P689 "The Cockpit of the Future" overall architecture.

The thesis' proposed system is composed by two main modules: the Workload Assessor and the Workload Manager. These two modules are highlighted in the architecture Figure 3.1 with a gray background.

The Workload Assessor is located in the Information layer and is the module

responsible to assess the driver's workload level through a machine learning algorithm. The assessed level is inferred based on the features generated by the other sub-modules of the Driver Monitor module and by the Simulator module.

The Workload Manager is located in the Decision layer and is the one that, from the previously assessed workload level, instructs the HMI manager to act on the HMI system in order to counteract abnormal levels of workload and to restore its optimal level. Their instructions are based on information from the Context Assessor in which the driver and the vehicle meet. The Context Assessor is responsible for the identification of scenarios/situations to which applications should adapt their behavior to. Therefore, the actions to be applied in the HMI system are selected to respond in the best way to the situation in which the driver is. Moreover, these actions have driver-based meaning where each applied action is the more effective for the actual driving context and for the driver in the abnormal state.

Further details regarding the two aforementioned modules will be given in the following sub-chapters.

The project P689 and the thesis' proposed system architecture were designed taking into account an autonomous driving environment with 5 levels of automation. Each level represents incremental differences on the driver's task control degree (Table 3.1). As far as this thesis is concerned, these automation levels and their differences are largely relevant for the Workload Manager methodology and, more specifically, to the process of instructing the actions since these levels can influence their specificity.

**Table 3.1:** Summary of SAE International's Levels of Driving Automation for On-Road Vehicle [25].

| SAE Level | Name | Execution of Steering and Velocity control | Monitoring of Driving Environment | Fallback Performance of *Dynamic Driving Task* | System Capability (Driving Modes) |
|---|---|---|---|---|---|
| **Human driver monitors the driving environment** | | | | | |
| **0** | **No Automation** | Human driver | Human driver | Human | n/a |
| **1** | **Driver Assistance** | Human driver and system | Human driver | Human | Some driving modes |
| **2** | **Partial Automation** | System | Human driver | Human | Some driving modes |
| **Automated driving system monitors the driving environment** | | | | | |
| **3** | **Conditional Automation** | System | System | Human | Some driving modes |
| **4** | **High Automation** | System | System | System | Some driving modes |
| **5** | **Full Automation** | System | System | System | All diving modes |

The taxonomy and definitions for the automated driving, and its levels, were based on the SAE (Society of Automotive Engineers) international's J3016, issued January 2014 [25].

### 3.1.1 Workload Assessor architecture

The Workload Assessor is an holistic HMI concept for automotive applications that estimates the driver's workload level. This module infers the driver's workload level from multimodal data. This data encompasses physiological data, vehicle data and environment data collected from the DSM (Driver Simulator Mockup) (*e.g.*, simulated vehicle and environment data) and from physiological sensors (*i.e.*, extracted from the Driver Monitor module). A data-driven approach using a machine learning algorithm was applied in order to estimate the driver's workload level. The system's output encompasses the workload level as a scale of *low*, *normal*, and *high* levels (Figure 3.2).

The Workload Assessor receives information regarding the external Environment Context, the Driver Context and the Vehicle Context, estimates the workload in one of the three aforementioned levels and sends this information to the Context assessor.

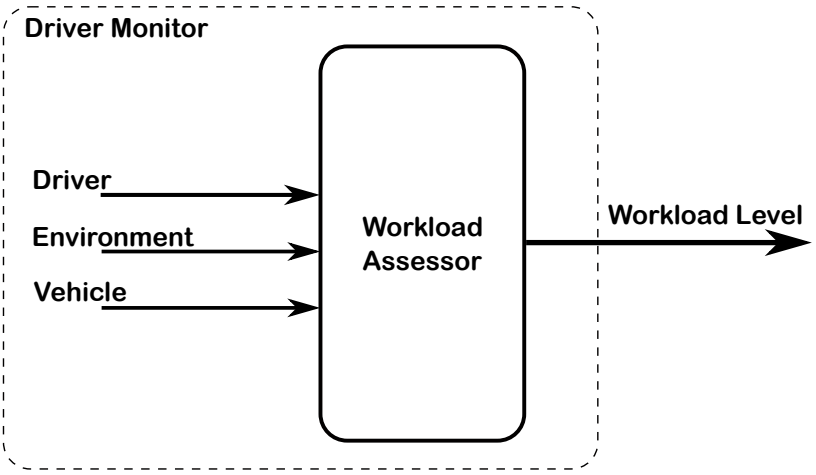



**Figure 3.2:** Workload Assessor I/O diagram.

Deep learning is applied to the fused multimodal data. This type of learning has been successfully applied in many areas, including computer vision, speech recognition, and multimodal data fusion [23]. The approach encompasses the use of a deep learning algorithm, which has rarely been used in this domain to address the driver state detection problem. Instead of treating types of data differently, the

multimodal data time sequence was fused using a two-dimensional matrix, where one dimension is used for the different sensor data and the other is used for time.

A Convolutional Neural Network (CNN) model composes the Workload Assessor deep learning algorithm. Although this algorithm is not widely used by the community that develops machine learning-based workload detection algorithms, the promising results that were analyzed in the literature review led to its choice. Combining this with the fact that a deep learning algorithm prevents data overfitting and reduces the cost of computing when compared to other types o neural networks, it makes its choice much more reinforced and valid. Moreover, since the basic structure of the algorithm is a neural network, it becomes easier to understand its dynamics and structure, which is an important factor due to the lack of previous experience on the part of those who studied and will developed it. Its implementation is performed using the machine learning framework recently unveiled by Google: the TensorFlow [17]. As stated in the Chapter 2, Section 2.1.2.6, this framework provides a high-level API that facilitates the process of developing machine learning algorithms as well as the process of training and deployment.

The CNN model structure (Figure 3.3) to be developed encompasses two layers of convolution (Convolution Layer 1 and 2), which can be stated as a standard in this type of algorithm. Max-pooling layers follow and interleave each convolution layer, making a total of two. After establishing the first four layers aforementioned, two fully-connected layers, respectively Fully-connected Layer 1 and 2, are then added at the bottom of the structure. The penultimate (Fully-connected Layer 2) one implements a dropout layer with a dropout rate of 75%. The dropout is a smoothing technique that allows the reduction of the training data overfitting in a neural network through the drop of random neurons from it, at each iteration. The rate means a probability for a neuron not to be dropped out. Regarding the activation function applied to all layers (except to the last fully-connected layer), the Rectified Linear Unit (ReLu) is chosen instead of the Sigmoid function. The Sigmoid activation function is actually quite problematic in deep networks. It squashes all values between 0 and 1 and when you do so repeatedly, neuron outputs and their gradients can vanish entirely. Moreover, modern networks use the ReLu activation layer. To the last fully-connect layer, a Softmax regression activation function is selected because it is best suited for classification problems. This layer will give the probability of the driver being with a certain workload level: *low*, *normal* or *high*.

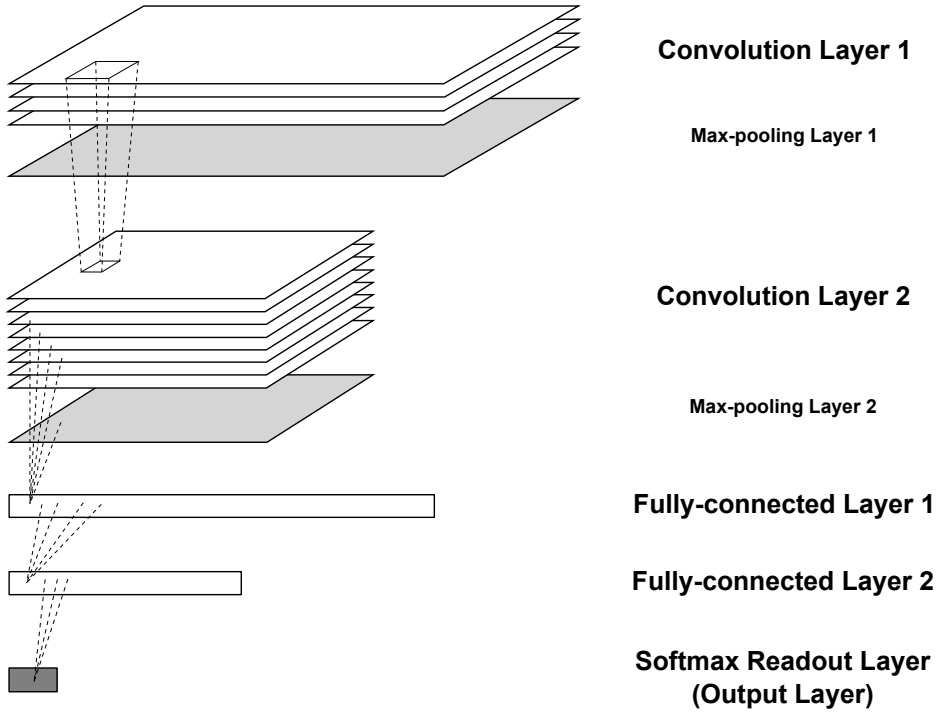



**Figure 3.3:** Convolutional Neural Network model structure from the Workload Assessor system.

#### 3.1.1.1 TensorFlow integration

As stated in the Chapter 2, Section 2.1.2.6.1, Tensorflow provides multiple complementary platforms that helps the process of model training and its deployment. Therefore, these additional tools are necessary in order to carry out the process of training and deployment of the trained model throughout the system development stage. The model training pipeline and the architecture that will support the clients serving is illustrated in the Figure 3.4.

The Workload Assessor model training and validation will use the collected experiment data splitted in two different datasets: 70% for the training dataset and 30% for the test dataset [10, 11, 12, 13, 14, 59, 95]. During the training phase, the model is adjusted according to its error, while in the validation phase the data is used to measure the network generalization and to halt training when generalization stopped improving. The testing phase will produce no effect on the training phase, but provides an independent measure of network performance during and after the training phase.

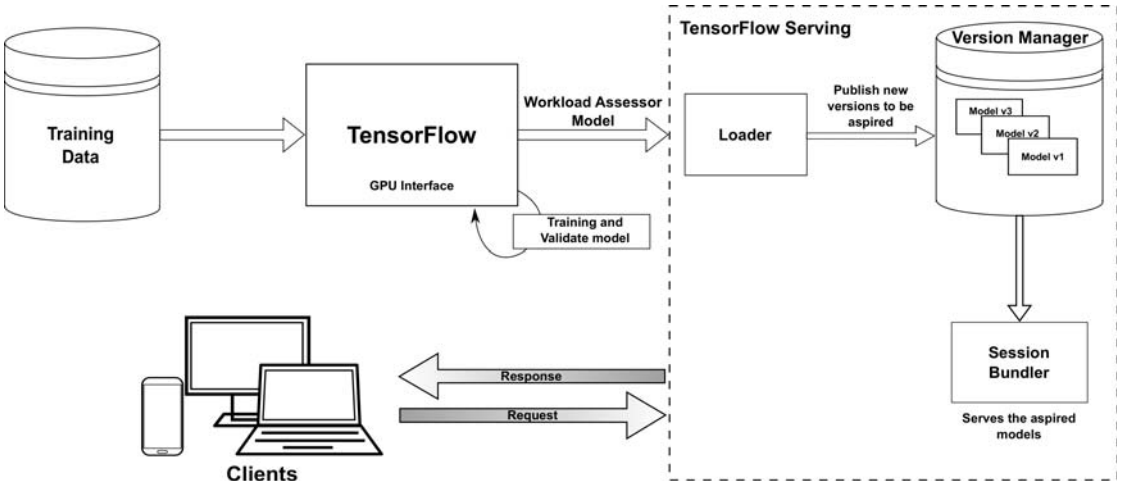



**Figure 3.4:** Model training and production pipelines architecture.

The training procedure will encompass the use of a Cross-Entropy loss function and an Adam optimizer. The loss function will compute a value that represents how badly the system predicts and the optimizer will try to minimize the loss function error by adjusting the neuron weights and biases with such values that improve the overall model accuracy. The Cross-Entropy is selected as a loss function because, for classification problems, is more efficient (*i.e.*, faster convergence) than the Euclidian distance or Mean Square Error (MSE), for example, and can be considered as ubiquitous in modern deep neural networks [96]. Regarding the optimizer, the Adam algorithm is selected instead of the classical Gradient Descent due to the deepness characteristic that the CNN model will present. When dealing with high dimensional spaces, saddle points are frequent. Although these points are not local minima, the gradient is, however, zero and the gradient descent optimizer remains stuck there. In this way, the Adam optimizer was adopted, which does not present the above mentioned problem and is able to "navigate" easily throughout these saddle points [97].

The TensorFlow API [17] provides, programmatically, all the aforementioned necessary methods during the training loop. Moreover, the training, validation and test phases will be computed through a GPU interface, in order to obtain a faster processing pipeline.

After the training, validation, and testing phases are concluded, the training pipeline will produce a trained and validated Workload Assessor model. This model has all conditions to proceed to the production deployment in order to assess the driving world, through the TensorFlow Serving platform.

The production deployment will be performed using the TensorFlow Serving

(TFS) [81] tool that enables the deploy of new algorithms and experiments, while keeping the same server architecture and APIs. Recapitulating, the TS provides a direct integration with TensorFlow models, with ease extending to serve other types of models and data. This is possible due to the flexible, high-performance serving system that it offers.

A TFS server will be developed to serve model estimations for all clients that connect to it. This server should include, among others, the submodules that are in Figure 3.4, delimited by the border of TFS. In TFS, each model is treated as a servable object. Periodically, the server scans the local file system and loads or unloads the models (*i.e.*, Loader), according to the local file system itself and the version policy of the model (*i.e.*, Version Manager). After the appropriate model version is found, the server is ready to respond to customer requests for workload level estimation (*i.e.*, Session Bundler). This architecture enables trained models to be easily deployed by copying exported models to a specific file path during TFS continuous execution (*i.e.*, while the TS server is running, different versions of the trained model can be deployed to an anteriorly specified file path and will be automatically loaded during its execution if, for example, a newer model version arrives).

In order for clients to perform prediction invocation to the model's server, they must respect the communication protocol exposed by TFS. This interface is the gRPC [98], an open-source, high performance remote procedure call (RPC) framework that runs on HTTP/2. The gRPC uses Protocol Buffers (*i.e.*, protocol buffers are Google's open-source mechanism for serializing structured data in efficient binary format), also known as Protobuf, as the format of messages exchanged between server and client. Therefore, as illustrated in Figure 3.4, the communication will be based on gRPC requests and gRPC responses. The client's request encompasses the two-dimension structured matrix with all features that are collected from the driver's physiological sensors and the simulator data (*i.e.*, vehicle and environment data). It sends this information through a gRPC request, the TFS server receives it and makes it available to the model for inference. The result will be then sent back to the client through a gRPC response message with the inferred workload level.

This TFS architecture (Figure 3.4) allows an horizontal scalability since the prediction with the TensorFlow models is inherently a stateless operation. Therefore, more instances of different types of clients (*e.g.*, remote or local Computers, smartphones, etc) can be added, as long as they meet the communication protocol requirement.

### 3.1.2 Workload Manager architecture

The Workload Manager (WM) supervises the HMI system and provides actions that are able to adapt it accordingly to the driver's overload or unverload level. Based on workload level estimated by the Workload Assessor, this system provides countermeasures to a specific driver and context of driving, in order to be possible to adjust and normalize the driver's cognitive workload. These countermeasures are action-based and act directly on the HMI system (through the HMI Manager), influencing indirectly the driver.

Afterward the driver's workload level assessment, this information is stored in the Context Assessor. Then, the Workload Manager gathers the workload level and other variables that define the current driving context (*e.g.*, the trigger that influenced the driver state of underload or overload and the automation level engaged by the car). The trigger is considered the source that originated or influenced an abnormal level of workload on the driver.

The fact that the trigger that originated the driver's abnormal workload level is taken into account helps in the selection of a possible action or countermeasure that proves to be more appropriated for the current driving context. Therefore, the only part of the HMI system that will be affected is the one that is considered most appropriated to influence the level of the driver's workload, leaving the remaining HMI experience unchanged. These triggers encompass driver stress, weather conditions, traffic conditions, driver activity (*e.g.*, HMI interaction), secondary/non-driving related tasks performance and driver drowsiness.

The WM module receives also as input the current vehicle's automation level. This input becomes important in defining the strategy of applying the action to the HMI system, just like the previously mentioned trigger. The automation level may indicate whether there is no active ACC type (i.e., automation level 0) or whether there is a full automation level (*i.e.*, automation level 5). In this way, you can make a selection of the actions that are most appropriate for each level. At higher automation levels (*i.e.*, between automation level 4 and 5) if the driver is in an underload state, a possible audible warning may not be issued since the driver is only in a supervised state and may want to continue in this state (*e.g.*, sleeping). On the other hand, at lower levels of automation, this type of warning may now be considered. Thus, the level of automation becomes a very important variable in selecting the best action to be applied in the HMI system and, indirectly, influencing the driver.

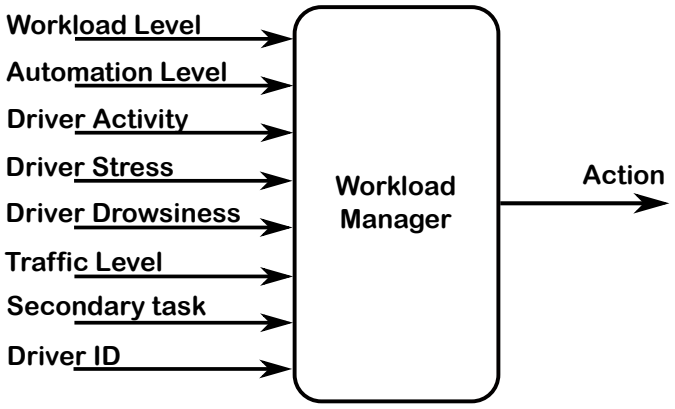



**Figure 3.5:** Workload Manager I/O diagram.

Lastly, the driver identification is included as in input of the Workload Manager in order to delegate personalization to this module. Further details regarding this personalization feature will be presented over this subsection.

The Workload Manager encloses an Artificial Intelligence (AI) agent that is responsible for the choice of an action, regarding the workload level, driver identification, driver activity, stress, drowsiness, traffic level and secondary task. The agent takes the inputs and, regarding their value, it returns all possible actions that match the input requirements. The AI agent is based on SWI-Prolog [99], a free implementation of the programming language Prolog. This mechanism follows a rule-based logic and is simple to apply. Moreover, an approach using reasoners with a defined ontology could also be applied, both due to the high performance that could be harnessed and to the potentiality that it presents. However, project constraints in terms of available resources and a driving context domain with an average complexity are reasons that lead to this approach to be dropped.

However, the Prolog agent only returns the set of actions that corresponds to the specified rules in the Prolog file. Thus, no driver-specific action based on the actual context is indicated. Therefore, the WM has an inner action-reward driver-specific algorithm that handles with this personalization. Relatively to the driver that is driving in a specific moment, within the set of actions that is returned by the Prolog agent, the algorithm manages the best rewarded actions based on the driver's past. This is, if an action was previously applied to a driver and resulted in a driver workload normalization (*i.e.*, driver's workload changed from an abnormal level to a normal level), a positive reward must be attributed to that action. On the contrary, if the application of an action did not result in driver's workload normalization, then a negative reward will be attributed to it. These informations are stored in a local driver profile in order to be loaded at any time.

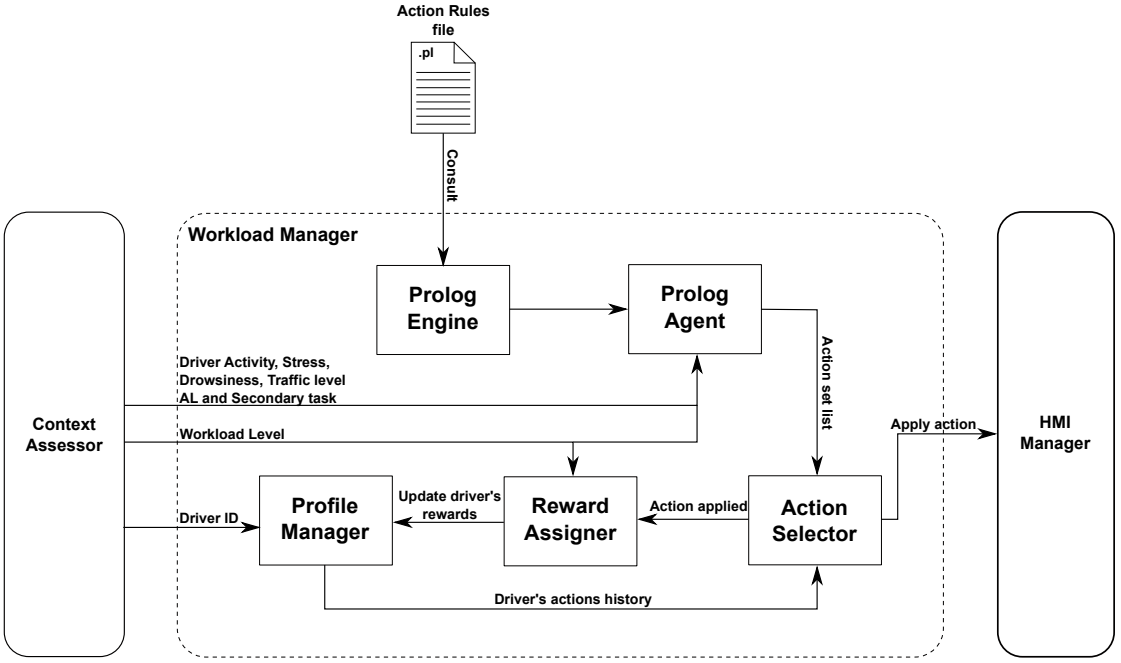



**Figure 3.6:** Workload Manager blocks diagram.

A system block diagram is depicted in the Figure 3.6 in order to better understand its dynamics. The Prolog Engine starts by consulting the file that gathers all Action Rules, generating after a Prolog Agent that will work as an interface when a query to the action rules is required. In this module, the Prolog Agent is queried with the input features (*e.g.*, driver activity, driver stress, driver drowsiness, traffic level and secondary task) and a set of actions that complies with the action rules is returned. Furthermore, an action selection is made based on the aforementioned returned set and the driver's actions history, in order to apply the best rewarded action from the returned actions set. Afterwards, the selected action is sent for the HMI Manager in order to be applied to the HMI system. Based on the success of the action employment, the Reward Assigner analyzes the current workload level in order to find out if a normalization level was obtained. The result of this analysis is then sent to the Profile Manager in order to update the driver's profile with the current action rewards.

#### 3.1.2.1 Action rules specification

The countermeasures used by the Workload Manager to normalize the abnormal workload states are the action rules. These actions give specific information to the HMI system to manage any of the modalities that it controls in order to make the driver's workload level return to a normal state. The majority of the

actions were specified with empirical knowledge and with the notion that they are plausible to be applied to the HMI system to combat the induced abnormal driver workload states. The literature does not report palpable results or even solutions, duly validated, regarding computer algorithms that allow to manage the driver workload level.

**Table 3.2:** List of available actions used by the Workload Manager system.

| State | Trigger | Actions | Automation Level |
|---|---|---|---|
| Overload | Weather | - Prioritize HMI displayed information<br>- Propose turn on ADAS<br>- Limit use of Secondary tasks | 0 - 1 |
| | Traffic | - Prioritize HMI displayed information<br>- Propose turn on ADAS<br>- Limit use of Secondary tasks | |
| | Physiological | - Prioritize HMI displayed information<br>- Propose turn on ADAS<br>- Play relaxing music | |
| | Secondary task | - Limit/Turn off secondary task<br>- Limit/Prioritize displayed information | |
| Underload | Drowsiness | - Visual and auditory warning<br>- Play music<br>- Turn on cockpit ambience lights<br>- Decrease cockpit temperature | |
| | | - Visual and auditory warning<br>- Visual and auditory proposition to switch back to lower automation levels | 2 - 3 |
| | | - Visual warning<br>- Visual proposition to switch back to lower automation levels | 4 - 5 |

Table 3.2 specifies the actions according to the external conditions where they were intended to be more suitable and have a greater positive impact. Therefore, they were divided concerning the workload level, the main trigger that induced the abnormal workload state and the automation level owned by the vehicle.

## 3.2 Experimental setup

The experiment to be developed will allow the data collection from a driving simulator, in order to create a database to train the previously specified model.

Therefore, a human-in-the-loop experiment will be conducted in a Driving Simulator Mockup (DSM). The DSM (Figure 3.7) delivers a very realistic driving experience, simulating different weather conditions (*e.g.*, day, night, fog, rain) in various types of roads (*e.g.*, urban roads, highway, etc) with all kinds of environment elements (*e.g.*, buildings, lampposts, moving and parked cars, trees, pedestrians, etc). This simulator is used with the goal of validating the HMI system being developed under the P689 Project, mentioned in the Chapter 1.

The DSM encompasses an off-board projection (three projectors) with a 210° viewing angle screen in front of a stationary car cockpit mockup and incorporates the Silab Driving Simulator software[1] [100]. The car cockpit is instrumented with a fully interactive multimodal HMI system, that is composed by (i) one cluster or dashboard (where the vehicle velocity, engine revolutions and all the standard telltales are displayed), (ii) one lower stack (where the driver can interact with some minor functions of the HMI system and the car, like the air conditioning) and (iii) one central stack (where the driver can interact with the holistic HMI system, such as play music, news visualization or personalize the cockpit environment, and even receive notifications regarding system's or vehicle's significant events). Moreover, sensors are installed in the cockpit in order to obtain data regarding the driver's physiology, activity and response (*e.g.*, eye tracker system by SmartEye, BioPack system, etc).

**Figure 3.7:** Driving Simulator Mockup [21].

For the experiment itself, 21 subjects will be recruited with between the ages of 20 and 30, and each must comply with the following conditions:

- Hold a driver's license for more than 1 year;
- Aged between the 20 and 30 years old;
- Drive frequently;

[1]Developed by Würzburg Institute for Traffic Sciences (WIVW)

- No major medical illness;
- No recent surgery;

Participants will be subjected to two different experiments: the first experiment will be composed by the execution of scenarios in one group (*i.e.*, manipulating only one independent variable) and aim to provoke a *high workload* state as well as a *normal workload* state (in order to have a driver's reference state), and the second experiment will be composed by one scenarios aiming to induce a *low workload* state. Each experiment will be described later in this section.

Each one of the experiences, and its different scenarios, will adopt a established procedure to which the subjects will have to follow. The experimental procedure flowchart is depicted in the Figure 3.8 and will be applied for both experiments.

The experiment procedure encompasses an initial familiarization with the DSM where all the commands and interaction points concerning to the cockpit will be shown to the subject, due to the possibility of its activation during the experiment. At the same time, information about the need to use some cockpit interaction points during the performance of the experiment and a brief summary about its unfolding will be presented to the participant. Then, a driving training segment aims to provide familiarization time to the subject with both the cockpit and the simulated environment, and to check for possible nausea symptoms that are known in this type of experiment. After the familiarization procedure, the experiment execution will be assigned to the subject. When the experiment period ends, if in a city scenario, a workload subjective assessment will be conducted, using the DALI subjective measure. If a highway scenario is being performed, the subjective questionnaire to be applied is the Karolinska Sleepiness Scale (KSS) [101]. It was chosen as a subjective measure due to the fact that the project in which this dissertation is inserted already have all the support to assess with it. Therefore, the assessment procedure will require the subject to select one of the nine factors that the measure provides through an Android application that will pop up every 5 minutes, during the scenario execution. Thus, the KSS application goal will be to give insight regarding periods where drivers may feel less involved in the driving task and the low workload level reached, by means of sleepiness levels.

When all scenarios regarding the experiment were already performed, the next experiment must be introduced if there is still another one not performed. Otherwise, the experiments session will be terminated.

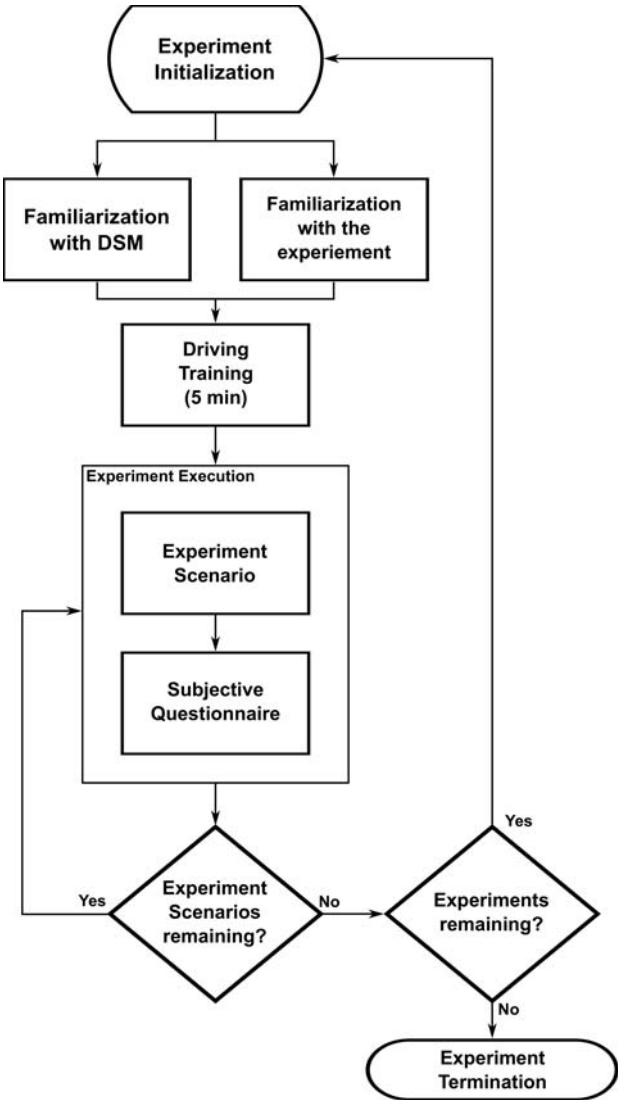



**Figure 3.8:** Experiment procedure flowchart.

The next two subsections will present the various scenarios within each experiment and how they will be introduced to the participant.

### 3.2.1 Experiment 1

The Experiment 1 will be the first to be presented to subjects. It encompasses one group of scenarios where the traffic level will be varied. In this group, the scenarios presentation order for participants will be done randomly, not allowing it to adapt to the independent variables variation. A possible use-case scenarios presentation with a gradual increase in traffic level could lead to the adaptation to the difficulty of the scenario by the subject.

The experiment will consist of a defined structure regarding the simulated scenario. This structure consists in four types of road elements: an adaptation road, a straight road, a roundabout and an intersection. In all scenarios of this experiment, their disposition throughout the simulated course will be static. However, from use-case in use-case, the visual aspect (*e.g.*, the landscape, the house types, etc.) and morphological (*e.g.*, roundabout size) will be changed, in order to not allow the driver to become familiarized to the scenario characteristics. The flowchart that translates the Experiment 1 simulation is shown in Figure 3.9.

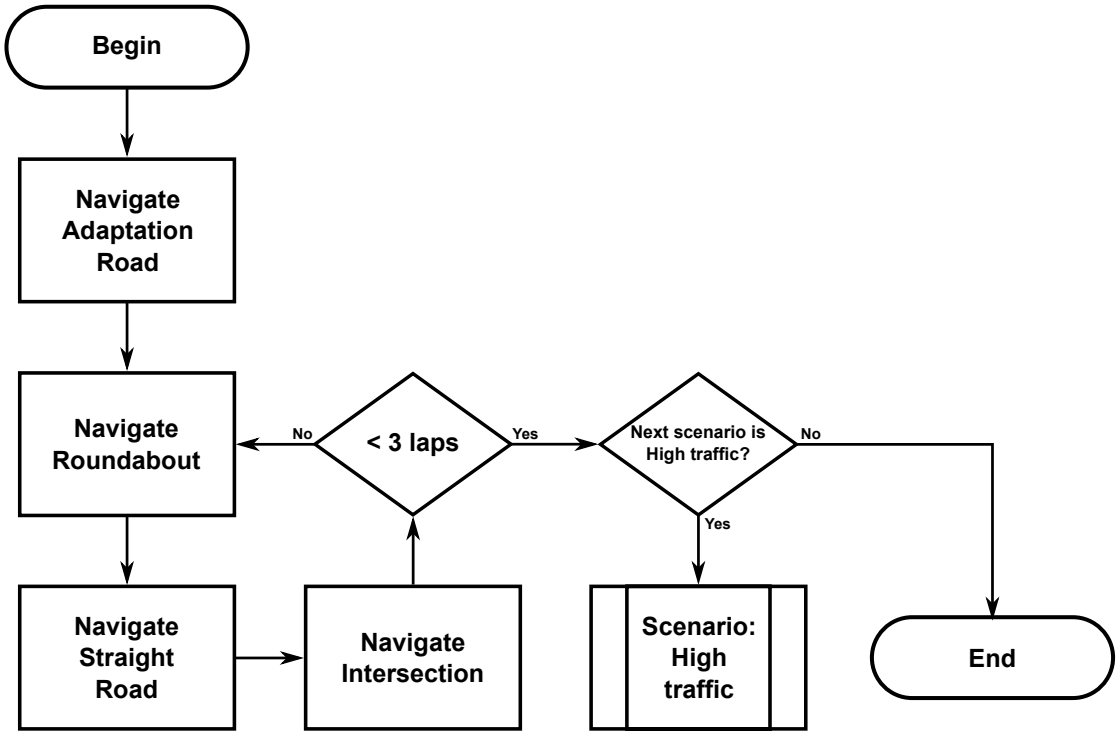



**Figure 3.9:** Experiment 1 simulation flowchart.

The subjective assessment questionnaire regarding the completed use-case will be done at the beginning of the next use-case adaptation road.

In order to increase the chances of having high workload level periods on subjects during the experiment execution, a *N-back* test will be applied as a non-driving related task (*i.e.*, a Secondary task). As mentioned in early sections (Section 2.1.1.4.2, Chapter 2), this task encompasses a sequence of stimuli (in this case, number between 0 and 9) presented one at a time and asked to compare the current stimulus to one presented $n$ items prior in the sequence [102]. For example, if performing a 2-back test, the current number is a target if it matches the number presented two numbers ago. Therefore, a 2-back test will be applied to subjects during the high traffic scenario execution, in complex sections (*e.g.*, roundabouts

and intersections).

#### 3.2.1.1 Traffic scenarios group

As aforementioned, the traffic group will cover different use-case scenarios, where the traffic level in the simulation will vary. Details regarding these use-case scenarios are depicted in the Figure 3.10.

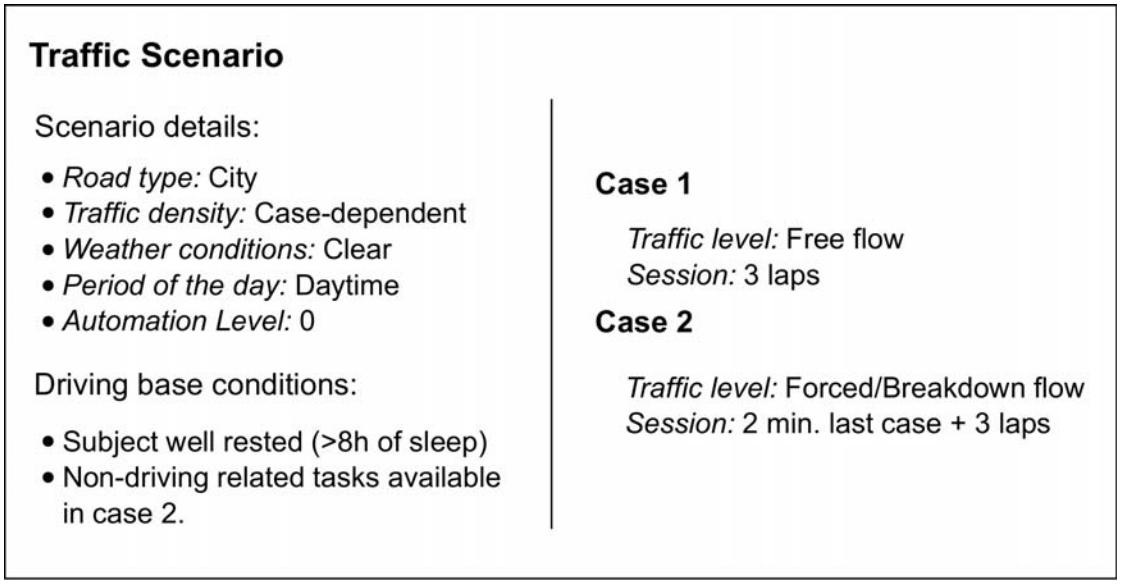



**Figure 3.10:** Traffic scenarios group details.

Traffic scenarios have their baseline well established. For all use-cases, the experiment will be in a simulated city environment during the day and with a variable traffic density between use-cases. Moreover, in all these cases, the level of automation that the car handles is fixed to zero. Furthermore, it is assumed that the driver presents the ideal conditions for the execution of the experiment, which means having a well rested condition (*i.e.*, 8 or more hours of sleep), thus ensuring that no fatigue issues arise and influence this type of experience.

Participants will have to complete three laps in the driving course that is outlined for each of the cases. However, cases that are executed in second place, will have, as a complement, a period of 2 minutes of driving with the conditions applied to the preceded case, at the beginning of the current case. The reason is that, since the subjective measure survey is presented to the driver at the end of each scenario case, this period allows that a real transition between scenarios takes place. Moreover, as aforementioned, among these cases, traffic will be varied

in three different levels: Free flow, Stable flow and Forced or Breakdown flow[2]. These different cases will be applied in a random order, as already explained.

### 3.2.2 Experiment 2

The Experiment 2 will be the second and last to be presented to subjects. It will be composed by only one use-case scenario. The experiment simulated scenario will essentially encompass a very uncomplicated highway environment, abstaining from the presence of billboards, the abundance of nature elements or road signs. Moreover, stimulant substances or products cannot be used by participants, during or before the experiment, in order to enable a fatigue state to affect the subject during the experiment.

The main goal of this experiment is to induce a low workload level in the participant. Apart from the visual elements that make up the experiment scenario, the most important points for the induction of the underload state is the monotony characteristic that a highway presents and the duration that the experience itself will take. The use-case duration will be about 40 minutes. Furthermore, one variable to take into account in the experiment is the level of automation, which will range from zero to two in the use-case scenario. Thus, the ease in driving by the subject, allowed by the ADAS, will also influence the driving demand and performance (*i.e.*, the commitment that the driver presents to the driving task), allowing to drag the subject to lower workload levels.

The experiment simulation will encompass a highway scenario, as aforementioned. During the scenario execution, a tablet will will increase the screen brightness and a KSS scale will appear (see Appendix C.1). The participant will have to select the value of the KSS scale that best suits the drowsiness level that he presents.

Globally, it will be possible to allow a low level of workload to be achieved by the subject. The details about the Highway scenario are presented in the Figure 3.11.

[2]The definition of each traffic level is explained in the glossary chapter.

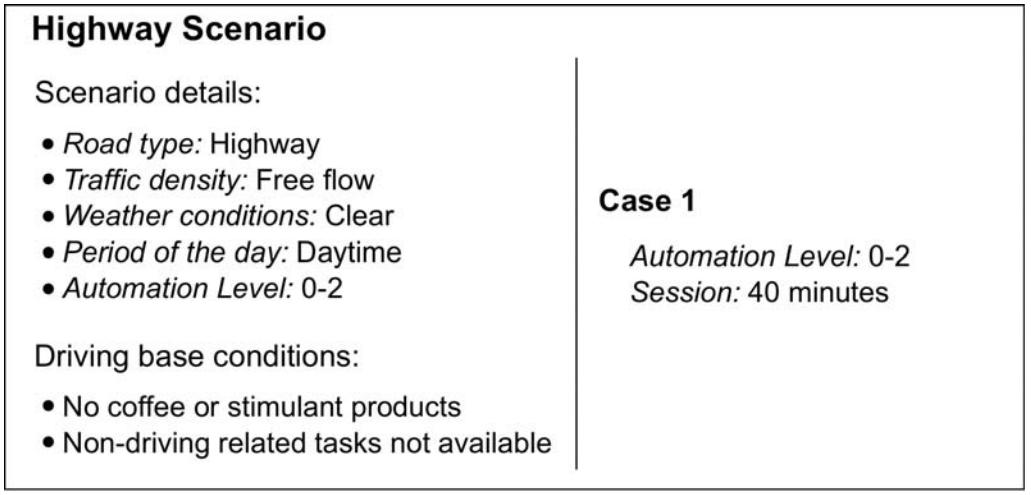



**Figure 3.11:** Highway use-case scenarios details.

### 3.2.3 Experiment data collection

The experiment data collection to be performed during the experiments will allow the dataset creation in order to be used during the WA model training. The experiment data collection will allow the database construction for training the Workload Assessor model. As mentioned in earlier sections, the model input features are derived from three different categories: driver physiological data, vehicle data and environment context data. These features are organized by category and source system in the Table 3.3, and also which values from each feature will be considered as inputs for the algorithm.

Data will be gathered at a sampling frequency of 30 Hz, due to project constraints and also because it is largely used by the literature regarding this domain. Although the entire Experiment 1 will be recorded in a database, the gathered data holding the most important information regarding workload are those that are collected during the most demanding road sections (*i.e.*, roundabouts and intersections). Therefore, points of interest were placed in the simulation environment five meters before and fifty meters after these demanding sections, so that these points are easily identified during the post-experiment analysis. Regarding the Experiment 2, the analysis strategy will go through processing the data that is further away in time from the beginning of the experiment and that presents the higher KSS scale value. The close to the final part of the experiment is where the level of workload is believed to be low because of the long driving time in a monotonous environment.

During the experiment, all types of data coming from different sensory sources

will be continuously collected (*e.g.*, eye-related data from the eye tracking system, environment and vehicle context data from the simulator, and heart rate data from the smartwatch). Therefore, a central data management system will be built, aiming the synchronous join of data from different sources so that a training database is built and presents all the synchronized components in a single repository. In this way, the construction of the training database will be less error prone than using a manual process of joining these large amounts of data.

As mentioned in earlier sections, the subjective assessment questionnaire will be taken after the completion of each use-case scenario, in the beginning of the adaptation road of the next scenario. The DALI subjective questionnaire will be presented to the participant and, with the help of a collaborator, it will be fully answered (see Appendix B). In the post-experiment analysis, scores given by the participant to the 6 DALI factors regarding each scenario will be taken in consideration and a global workload score will be generated. Based on this value, the data collected during the experiment (*i.e.*, physiological, vehicle and environment data) will be properly labeled. Later, this labeled dataset will be used to train the workload model, following a supervised learning approach as aforementioned.

## 3.3 Data Preprocessing

In order to enhance the best knowledge discovery during the training phase, a pre-processing step will be performed. From this step it will be possible to obtain the final dataset for training the model. A technique for balancing the dataset will be applied so that the distribution of the minority classes is adjusted (i.e., SMOTE). Further, before processing the dataset itself, an analysis of the features that compose it will be performed (i.e., P-value). This step will allow to perform a feature selection based on the correlation between features and workload levels (i.e., the ground-truth). Allied to this feature parsing is the Error Incremental analysis that will be adopted during the training phase of the model.

**Table 3.3:** Experiment features collection.

| Category | Parameter | Values | Source System |
|---|---|---|---|
| **Driver** | Heart Rate | Bpm | BioPack system, Smartwatch and Radar |
| | Respiration Rate | Breaths per minute | Radar |
| | Blinking Rate | Blinks per second | Eye Tracker |
| | PERCLOS | Percentage of eyelid closure | Eye Tracker |
| | | | |
| **Vehicle** | Velocity | Km/h | SciLab Simulator |
| | Longitudinal acceleration (Ax) | $m/s^2$ | SciLab Simulator |
| | Lateral deviation (Ay) | Meters | SciLab Simulator |
| | Lateral acceleration | $m/s^2$ | SciLab Simulator |
| | Steering wheel | Angle (Degrees) | SciLab Simulator |
| | Brake Pedal | Angle (Degrees) | SciLab Simulator |
| | | | |
| **Environment** | Traffic Density | Low, normal and high | SciLab Simulator |
| | Road type | City and highway | SciLab Simulator |
| | Road Characteristics | Straight, roundabout and intersection | SciLab Simulator |
| | Time of Day | Darkness/Nocturnal, daytime and twilight | SciLab Simulator |
| | Weather conditions | Clear, rain, fog, fog and rain, and snow/glaze | SciLab Simulator |

### 3.3.1 Dataset Balancing

Due to the fact that the dataset is unbalanced, a balancing process must be applied. The dataset unbalancing comes from the fact that it is composed by

more samples representing certain classes when compared with the other classes. Therefore, the algorithm will have difficulties in learning and distinguishing data relative to a certain class.

To address this problem, an oversampling technique will be applied. The selected approach is called SMOTE (Synthetic Minority Over-sampling Technique) [103]. It is one of the most adopted approaches due to its simplicity and effectiveness. The SMOTE oversampling approach generates new minority class data instances via an algorithm, instead of replicating minority class samples (*i.e.*, copy and paste existing samples in order to fill the gap). The problem of just copying existing sample is that it causes overfitting because repeated instances causes the decision boundary to tighten. Alternatively, this algorithm creates synthetic samples that are similar to the minority class existing instances. Briefly, the oversampling algorithm takes an instance from the dataset, and look at its $k$ nearest neighbors (regarding the feature space). Therefore, to generate a synthetic instance, it takes the vector between one of those $k$ neighbors, and the current instance, multiplying this vector by a random number ranging between 0, and 1. Finally, it adds this value to the current instance in order to create the new synthetic data point.

The SMOTE algorithm implementation will not be performed in this dissertation due to the fact that there are already implementations of it. Instead, the Imbalanced-learn API will be used since it is highly recommended by the community as a good implementation and it is built as a Python package [104].

### 3.3.2 Dataset Normalization

Due to the fact that the dataset collected from the experiment encompasses multimodal data, the existence of different units and scales between independent variables is quite normal. Furthermore, it is necessary to normalize the dataset so that all data have only one scale (i.e., Feature Scaling). Due to the nature of almost classifiers, the range of all features should be normalized in order to each feature to contribute proportionally to the algorithm learning process.

This type of normalization prevents the neurons of a neural network from saturating due to variant scales of data. Moreover, in specific optimizers, it helps in a much faster convergence with feature scaling.

The normalization method to apply is the min-max algorithm. It simply rescales the range of features to scale a specific range (commonly scaled to [0, 1] or [-1, 1]). The normalization range selected for this preprocessing step is between 0 and 1. The formula for the dataset normalized calculation is as follows:

$$x' = \frac{x - min(x)}{max(x) - min(x)}, \tag{3.1}$$

where $x$ is the unnormalized feature vector and $x'$ is the normalized feature vector.

### 3.3.3 Feature Selection

The objective of the feature selection process is to assess which of the collected variables are statistically significant, contributing to a best classification performance. The ranking of the statistically significant variables provides an insight on the discriminative power of each variable. By selecting the optimal number of top-ranked features, a reduction of the dimensionality problem will be possible. Therefore, the best classification performance and efficiency could be achieved.

The statistical significance of each feature will be obtained through the compute of the P-value [105]. This measure indicates how likely a given outcome is under the null hypothesis. That said, the *p-value* represents the probability that the coefficient between the feature and the ground-truth (in this application scenario, could be between any other features) is actually zero. Briefly, *p-value* helps in the process of decision if there is a relationship between two variables or not. Therefore, the smaller the *p-value*, the more positive is the existence of a relationship between the two variables. The *p-values* origin comes from hypothesis testing in statistics. In this field, two hypothesis can be discriminated: (i) H0, the null hypothesis, where there is no relationship between two variables; and (ii) H1, the alternative hypothesis, where exists in fact a relationship between two variables. If the *p-value* is less than a specific threshold (often used 0,05 or 0,10), then the null hypothesis, H0, can be rejected, meaning that can be concluded that there is a relationship between the two variables (as the alternative hypothesis, H1, dictates).

The threshold specified and that differentiate features that are statistically significant for the workload level is 0,05 (*i.e.*, $p$<0,05).

Based on the most significant variables, a set of top-ranked features will be obtained. In order to selected the optimal number of top-ranked features, an error incremental analysis will be performed. It will start from the inclusion of the top-ranked features in the training dataset, and incrementally adding the next best ranked feature until all significant variables are included.

## 3.4 Development Environment

The development environment will allow the implementation of the previously specified systems. This environment encompasses both hardware and software platforms that will be specified throughout this section.

### 3.4.1 TensorFlow

The TensorFlow was selected as a machine learning framework, firstly, due to its support to beginners in the machine learning domain and the support coming from the community. Its support is based in well structured tutorials which introduce its API and how to use its various abstraction layers, and also helps on the development of some machine learning topologies. Moreover, the popularity and adherence by the community that TensorFlow is gaining since its release in November 2015, was an additional point that strengthened its choice for the implementation of the driver workload model. This is important in order to obtain the support directly from the community when problems with recent TensorFlow versions occur. Another crucial detail of TensorFlow is its performance that it presents regarding the computational graph construction and training. Therefore, a fast and efficient performance in the overall computation was an important feature for its selection. Lastly, the add-ons that integrate TensorFlow are considered as very useful tools: the TensorBoard and TensorFlow Serving. With these add-ons, the debugging and the algorithm health monitoring, and the deployment for production of the developed model can be easily and efficiently performed. Both tools were already mentioned and explained in earlier sections (Section 2.1.2.6.1).

### 3.4.2 NXP i.MX6 series platform

The NXP i.MX6 series platform is the required hardware platform by the P689 Project. Therefore, in order to not spend resources and as it was considered suitable platform to include the modules developed during this thesis. Moreover, the platform possesses high CPU clock speeds (up to 1.2 GHz in quad-core architectures) and GPU components to succumb to the most demanding performance, which is important for a fast processing of the input features coming from the aforementioned external sources, and a consequent efficient model inference. This

platform as contains a set of consumer- and automotive-used connectivity protocols (*e.g.*, Ethernet, FlexCAN, MOST), reveling an important feature for its use in an automotive domain. As security, nowadays, became more and more a concern, advanced security support (*e.g.*, high assurance boot, cryptographic cipher engines, random number generators) is implemented by the platform. Finally, the i.MX6 platform is supported by a Linux BSP as well as multiple third party operating systems and reference implementations that allow fast time to market and rapid prototyping. Since the thesis system development will be entirely performed using the Linux operating system, the aforementioned feature has a solid point regarding the choice of this platform.

# Chapter 4

# System Development

After the system's specification in the previous chapter, this chapter describes the implementation and the work involved in the development of the same system. Chapter 4 is divided into two major subsections, contrasting with the two stages of development that were required to implement the final systems.

Firstly, the Workload Assessor implementation is presented. A description of the model construction and the implementation of each layer is given. Moreover, the model training procedure is explained in order to understand which were the steps followed during this process. The deployment procedure is also described during the chapter, encompassing the model export, the model loading from the server, the client and server setup, and the communication between these two entities, through a queries exchange mechanism.

The second phase encompasses the Workload Manager implementation process description. In this phase, the mechanism built for the action rules loading based on a Prolog approach is described, as well as how the action rules file was implemented. Furthermore, the Profile Manager module that takes care of the load and save of the driver profile is properly presented. Lastly, both Action Selector and Reward Assigner modules are described in implementation terms.

The symbiotic interaction between the modules described during this chapter is also depicted in order to better understand the workflow that each overall system (*i.e.*, Workload Assessor and Manager) presents during their execution in the deployment environment.

## 4.1 Workload Assessor

The implementation process of the Workload Assessor system was the core stage regarding this dissertation and its focus. The literature review performed in the beginning of this dissertation brought the necessary knowledge to design this system and to do it the most stable and robust possible way. Understanding how to build a model using machine learning in the TensorFlow (Subsections 4.1.1.1 and 4.1.1.2) development environment as well as how to deploy the trained model to a production environment using TensorFlow serving and TensorBoard (Subsections 4.1.1.3 and 4.1.2), helped substantially the implementation phase and allowed the design of a scalable architecture. Also, the interaction with Workload Manager module was simplified alongside the adopted approach (Subsection 4.1.2.2).

### 4.1.1 Workload Assessor model

In order to recall the previous chapter with regard to this module, some aspects about it are presented here. Therefore, the model structure was built based on a Convolutional Neural Network (CNN). It was implemented and trained using the Google's framework TensorFlow [17], and it follows the structure depicted in Figure 3.3, presented in the Section 3.1.1.

#### 4.1.1.1 Model implementation

The Workload Assessor implementation begins with the development of its model that estimates the driver's workload level, between *low, normal* and *high*. As aforementioned, the Figure 3.3 depicts faithfully the structure of the implemented model and its layers as well as in terms of quantity and representativity within the model's architecture.

The first layer that encompasses the model is the Input layer (Listing 4.1). Its responsibility is to hold the input matrix that has the input features and, therefore, it will be used for the training and classification. The multidimensional matrix allows the application of a mini-batch approach during the training phase. It will enable to have a commitment between the computational resources needed to calculate the model parameters and the speed at which the model can learn and generalize.

**Listing 4.1:** Input holder definition.

```
def __init__(self, input_data_batch, label_batch):
 #(...)
 self.X_input = tf.placeholder(tf.float32, [None, 48, 6, 1],
                                                  name='Input')
 #(...)
```

After the Input layer processes its data, the layer that follows is the Convolution layer 1 (ConvLayer 1). This layer takes as input the result from the previous layer (*i.e.*, Input layer) and performs convolution on the input layer matrix. In this layer, each "neuron" does a weighted sum of the cells just above it, across a small region of the matrix. Therefore, each neuron reuses the same matrix of weights throughout the input matrix, sliding over it (Figure 4.1).

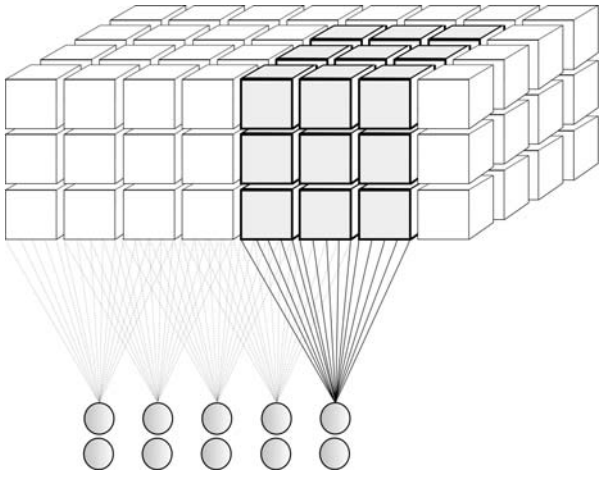


**Figure 4.1:** Convolution's process illustration.

Then, it adds a bias and feeds the result through its activation function. The Listing 4.2 represents the ConvLayer 1 definition.

**Listing 4.2:** Definition of both weights and bias used for the Convolution layer 1 and the activation function, followed by a Max-pooling layer.

```
def __init__(self, input_data_batch, label_batch):
 #(...)
 with tf.name_scope("ConvLayer1"):
  W1 = tf.Variable(tf.truncated_normal(shape=[6, 6, 1, 32],
                                stddev=0.1, name='Kernel_Conv1')
  tf.Variable(tf.constant(0.1, tf.float32, shape=[32],
                                          name='Bias_Conv1')
  Conv1 = tf.nn.conv2d(self.X, W1, strides=[1, 1, 1, 1],
```

```
9                           padding='SAME', name='Conv1')
10   Y1 = tf.nn.relu(Conv1 + B1)
11   Y1_pool = tf.nn.max_pool(Y1, ksize=[1, 2, 2, 1],
12                            strides=[1, 2, 2, 1],
13                            padding='SAME', name='Y_Conv1_pool')
14  #(...)
```

The output from the first Convolution layer is provided by a Max-pooling operation. This layer is responsible to down-sample the ConvLayer 1 output matrix, reducing its dimensionality and also the computational cost by decreasing the number of parameters to learn. The `strides` value in the `Y1_pool` definition (Listing 4.2, Line 11) indicates a reduction in half of the output matrix from the ConvLayer 1. Using the `padding` value as `SAME` in both layers means that the output size is the same than the input size. Therefore, it requires the weight window to slip outside the input map, hence the need to pad it evenly left and right (*i.e.*, add extra columns in order to obtain a odd columns number).

Following the previous layer is the Convolution layer 2 (ConvLayer 2). This layer has the same operational purpose than the previous one. However, it has a higher depth but receives inputs with smaller dimensions. Moreover, this ConvLayer will generate outputs with higher information relevance due to the stage where it appears. The ConvLayer 2 implementation is described on Listing 4.3.

**Listing 4.3:** Definition of both weights and bias used for the Convolution layer 2 and the activation function, followed by a Max-pooling layer.

```
1  def __init__(self, input_data_batch, label_batch):
2   #(...)
3   with tf.name_scope("ConvLayer2"):
4    W2 = tf.Variable(tf.truncated_normal(shape=[5, 5, 32, 64],
5                                         stddev=0.1, name='Kernel_Conv2')
6    B2 = tf.Variable(tf.constant(0.1, tf.float32, shape=[64],
7                                                  name='Bias_Conv2')
8    Conv2 = tf.nn.conv2d(Y1_pool, W2, strides=[1, 1, 1, 1],
9                         padding='SAME', name='Conv2')
10   Y2 = tf.nn.relu(Conv2 + B2)
11   Y2_pool = tf.nn.max_pool(Y2, ksize=[1, 2, 2, 1],
12                            strides=[1, 2, 2, 1],
13                            padding='SAME', name='Y_Conv2_pool')
```

The layers that follow the aforementioned Convolution layers, are the Fully-Connected (Listings 4.4 and 4.5). They take as input volume, an output coming from the previous layer, and maps it to an N dimensional output vector, where N

is the number of classes that the model has to choose from. However, these layers compute weighted multiplications in a two-dimensional form. Therefore, assuming that the output from the Convolution layers are matrices with a four-dimensional shape, the Fully-Connected layer implementation definition starts with a reshape of its input to a two-dimensional output. Then, the weighted multiplication is added to a bias that later feeds a ReLu activation function.

**Listing 4.4:** Fully-Connected layer 1 and activation function definition, followed by a Dropout operation.

```
1  def __init__(self, input_data_batch, label_batch):
2   #(...)
3   with tf.name_scope("FullyConn1"):
4    Y2_pool_flat = tf.reshape(Y2_pool, shape=[-1, 12 * 2 * 64],
5                                          name='Y_Conv2_reshaped')
6    W3 = tf.Variable(tf.truncated_normal(shape=[12 * 2 * 64,512],
7                                      stddev=0.1, name='Kernel_FC1')
8    B3 = tf.Variable(tf.constant(0.1, tf.float32, shape=[512],
9                                                  name='Bias_FC1')
10   Y3 = tf.nn.relu(tf.matmul(Y2_pool_flat, W3) + B3)
11   Y3_drop = tf.nn.dropout(Y3, self.pkeep)
12  #(...)
```

Before the output is sent to the next layer, it must be processed by a Dropout layer. It is a regularization technique that prevents the so-called phenomena of overfitting. This technique randomly drops some neurons that compose each Fully-Connected layer based on a `pkeep` value that represents the probability to one neuron to be kept or not. Then, at each iteration of the training loop, neurons are randomly removed with all their weights and bias. This probability value varies between 0.5 and 0.75, as a widespread acquired fact when using the Dropout technique. When proceeding to the model testing stage, the `pkeep` value is restored to 1 in order to put all neurons back to the network.

Recalling the previously presented consideration, this type of layer outputs a N dimensional vector, where N represents the number of classes that the model has to choose from. The second Fully-Connected layer definition (Listing 4.5, Lines 4 and 6) demonstrates the output of three classes. These classes, as specified in previous chapters, encompass the *low, normal* and *high* workload levels.

**Listing 4.5:** Fully-Connected layer 2 definition.

```
def __init__(self, input_data_batch, label_batch):
 #(...)
 with tf.name_scope("FullyConn2"):
  W4 = tf.Variable(tf.truncated_normal(shape=[512, 3],
                                       stddev=0.1, name='Kernel_FC2')
  B4 = tf.Variable(tf.constant(0.1, tf.float32, shape=[3],
                                                  name='Bias_FC2')
  Y_logits = tf.matmul(Y3_drop, W4) + B4
 #(...)
```

The two previously described Fully-Connected layers can be compared as a feed-forward neural network.

After the last Fully-Connected layer outputted a vector with information regarding the three aforementioned classes *low*, *normal* and *high*, as previously mentioned during the Fully-Connected layer 2 implementation description, the Softmax activation function is fed by this output vector. This function will allow to define the last step of the algorithm classification, indicating which of the classes presents the highest probability of representing the input data of the model, hence which workload level better represents the driver state (a brief explanation regarding how the Softmax function works is presented in Glossary, on page 155). The Softmax Readout's implementation can be consulted in the Listing 4.6.

**Listing 4.6:** Softmax Readout definition.

```
def __init__(self, input_data_batch, label_batch):
 #(...)
 with tf.name_scope("SoftmaxReadout"):
  self.Y_final = tf.nn.softmax(Y_logits, name='Y_final')
 #(...)
```

Despite the symbiotic relationship, the second Fully-Connected layer and the Softmax Readout implementation definitions were separated for representation purposes. Therefore, the classification moment could be clearly identified in the model's computational graph.

The TensorBoard platform allowed to visualize the previously described implemented model. As mentioned in the beginning of this section, methods like `name_scope()` or method's parameters as `name` (*e.g.*, `tf.name_scope("SoftmaxReadout")` and `tf.nn.softmax(Y_logits, name='Y_final')`), aim to identify layers, modules or layer properties (*i.e.*, weights and bias) in the graph generated by TensorBoard and that, in some way, represents the structure of the model implemented.

Thus, the resulting model graph representation generated by TensorBoard is depicted in Figure 4.2.

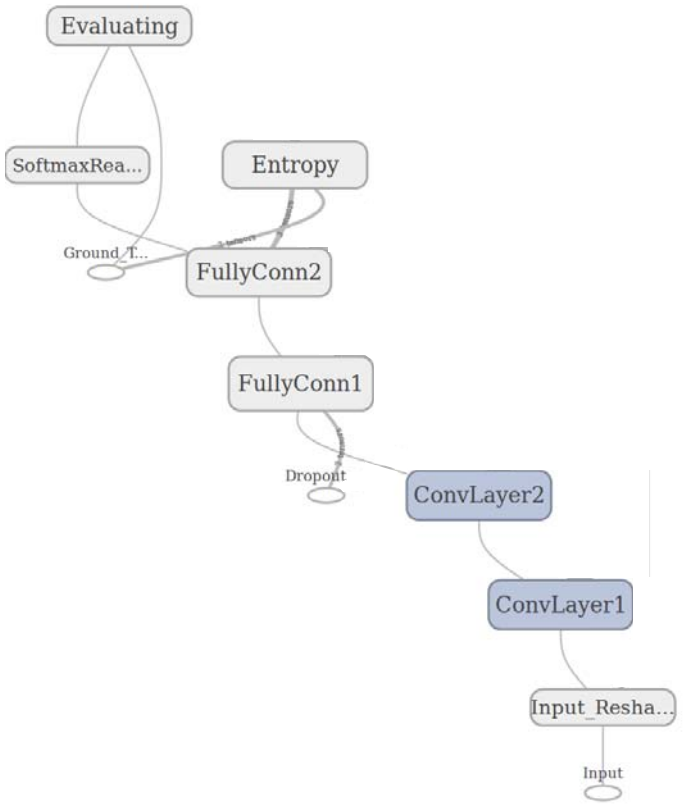



**Figure 4.2:** Model graph representation generated by TensorBoard, in a bottom-up fashion.

#### 4.1.1.2 Model training

Before proceeding to the model export and, consequently, its deployment in the final system, a training phase was introduced in order to allow the model to understand the experimental collected data, learn with it and, posteriorly, estimate driver workload level in an after-experimental phase. TensorFlow was also used during this phase. This section describes the implementation of all algorithms and functions necessary for the training phase and the loop that allowed the model to learn from the experimental collected data.

The first function implemented was the Cross-Entropy error. This loss function was already described in the previous chapter (Chapter 3, Section 3.1.1.1)

and also why it was selected. Therefore, recalling the previous explanation, this function measures how good the model estimations are good, *i.e.*, the distance between what the network estimates and what is known to be the truth. The ground-truth is provided through the Supervised learning approach applied to the model implementation, as already exposed in earlier chapters. Labels introduced in the experimental collected data (*i.e.*, the labeling of data process) allows to know the real truth regarding each data chunk. Briefly, this loss function compares the model prediction with the truth presented by labels on the data used for prediction. The Listing 4.7 presents the Cross-Entropy error function implementation.

**Listing 4.7:** Cross entropy definition.

```
1  def __init__(self, input_data_batch, label_batch):
2   #(...)
3   with tf.name_scope("Entropy"):
4    self.cross_entropy = tf.nn.softmax_cross_entropy_with_logits(
5                         logits=Y_logits, labels=self.Y_groundTruth)

6    self.cross_entropy = tf.reduce_mean(self.cross_entropy) * 20
7    ce_summ = tf.summary.scalar("cross entropy",
8                                      self.cross_entropy,
9                                      collections=['train', 'test'])
10  #(...)
```

As aforementioned, the error calculation is well noticeable in Line 5 (Listing 4.7). The `Y_logits` represents the model estimated class (*i.e.*, *low, normal* or *high* workload level) and the `Y_groundTruth` embodies the true classification class taken from labels on the labelled experimental collected data. Then, the Line 7 allows to bring the test and training cross-entropy error value to the same scale for their posterior display, using the `summary.scalar()` method. It allows the visualization of the cross-entropy value evolution throughout the training and test phases over TensorBoard (Figure 4.3).

After knowing how good the estimation was, a algorithm must be introduced in order to understand how weights and bias must be changed and then obtain correct estimations. This algorithm is the optimizer and, in this case, its name is Adam optimizer. Its role is to, based on partial derivatives of the error provided by the previously calculated cross-entropy, thus obtaining the gradient, update weights and biases by a fraction of the gradient, specified by the learning rate, and do the same thing again using the next batch of training images. The optimizer

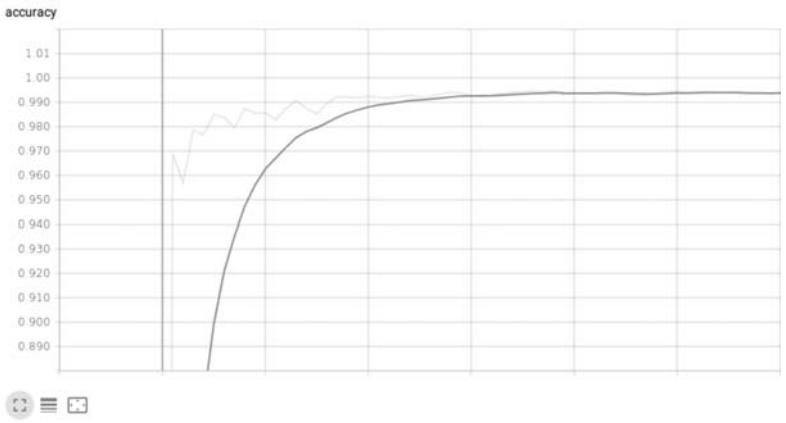



**Figure 4.3:** TensorBoard cross-entropy visualization example [22].

goal is to enable the cross-entropy to reach its minimal value while updating model weights and bias values. The loss function definition is presented in the Listing 4.8.

**Listing 4.8:** Optimizer definition.

```
1 def __init__(self, input_data_batch, label_batch):
2  #(...)
3  with tf.name_scope("Training"):
4   self.train_step = tf.train.AdamOptimizer(learning_rate=
5                                self.lr).minimize(self.cross_entropy)
6  #(...)
```

Precedent considerations are visible in the optimizer definition. The Line 5 in Listing 4.8 shows that it takes as input the learning rate, which indicates how long the training phase will take (*i.e.*, smaller learning rates lead to a slower training), and the cross entropy error, that represents how good the model is doing estimations. Based on these values, the optimizer updates accordingly all model's weights and bias.

With regard to keep trace of model's accuracy, a evaluation segment was implemented. It starts with the computation of the prediction correctness by comparing the estimated workload level with the label on each data chunk, that represents the true classification (Line 4 in Listing 4.9). The model accuracy is then calculated by performing a continuous mean on all `correct_prediction` values (Line 6). As defined in the cross-entropy implementation, the accuracy mean value is recorded during the training phase and a graphic illustration with its evolution can be accessed using TensorBoard. This visualization is possible

with the `tf.summary.scalar` method definition (Line 8).

**Listing 4.9:** Evaluation definition.

```
1 def __init__(self, input_data_batch, label_batch):
2  #(...)
3  with tf.name_scope("Evaluating"):
4   correct_prediction = tf.equal(tf.argmax(self.Y_final, 1),
5                                 tf.argmax(self.Y_groundTruth, 1))
6   self.accuracy = tf.reduce_mean(tf.cast(correct_prediction,
7                                          tf.float32))
8   accuracy_summary = tf.summary.scalar("accuracy",
9                                        self.accuracy)
10  #(...)
```

Posteriorly to the learner methods definition, the training loop must be addressed (Listing 4.10). It encompasses a `training_step` method that has auxiliary methods that manage how input data is used during training and at which pace the model learns.

**Listing 4.10:** Evaluation definition.

```
1 def launchSession(self, export, model_version):
2  #(...)
3  for step in range(1000+1):
4   self.training_step(step, test_data=(step%100==0),
5                      training_data=(step%10==0))
6  #(...)
```

In order to tell the optimizer which learning rate it should adopt during training, a learning rate decay is computed (Listing 4.11). This methodology is applied in order to prevent noisy results, with accuracy values going up and down. Therefore, as a prior adjustment of this variable to lower values which can cause the training to take a excessive time, a good solution encompasses a fast start (*i.e.*, higher learning rate) and a exponentially learning rate decay to 0.0001. The formula for the exponential learning rate decay is presented in the listing below (Line 7).

**Listing 4.11:** Learning rate decay definition.

```
1 def training_step(self, step, test_data, training_data):
2  batch_size = 20
3  batch_X, batch_Y = self.next_batch(batch_size)
```

```
4  max_learning_rate = 0.003
5  min_learning_rate = 0.0001
6  decay_speed = 1000.0
7  learning_rate = min_learning_rate + (max_learning_rate -
8              min_learning_rate) * math.exp(-step / decay_speed)
9  #(...)
```

As showed in training loop listing (Listing 4.10), a test is performed at each 100 training steps. Therefore, it will be possible to understand how well the model is learning and how accuracy and cross-entropy error are performing. In this test phase, data not used for training is applied in order to obtain results that are not biased. Listing 4.12 shows the test step definition.

**Listing 4.12:** Test step definition.

```
1 def training_step(self, step, test_data, training_data):
2  #(...)
3  if test_data:
4   result = sess.run([self.accuracy, self.cross_entropy,
5                      self.s_test, self.prediction, self.label],
6                      feed_dict={self.X_input: testData,
7                                 self.Y_groundTruth: testLabels,
8                                 self.pkeep: 1.0})
9   writer.add_summary(summary_test, step)
10  writer.flush()
11 #(...)
```

The method that deserves the attention from the step function in this loop is `run` (Line 7 in Listing 4.13). This method belongs to the TensorFlow API (more precisely from class `Session`) and runs over a `session` object. Since the session has the implemented model graph, launched in it, the `run` method will interact and influence the Workload Assessor model. It takes two arguments: `fetches` and the optional `feed_dict`. The `fetches` argument is a graph element (in this case `train_step`) that tells to TensorFlow which is the wanted output node and to run all necessary operations to calculate that node. On the other hand, the optional argument `feed_dict` is a dictionary that indicates to the caller which are the variables that are intended to override the value of tensors in the graph. In this implementation, `X_input` (*i.e.*, the model input), `Y_groundTruth` (*i.e.*, the labelled classification), `lr`(*i.e.*, the learning rate) and `pkeep` (*i.e.*, the probability that one neuron has to be kept on the network) are tensors owned by the imple-

mented model that are supposed to be overridden by the values held by the input tensors `batch_X`, `batch_Y`, `learning_rate` and `dropout_keep`, respectively. The method `run` performs the backpropagation training and is the main method that enables the model to learn. The `training_step` method executes only one step of TensorFlow computation, meaning that it will execute as many times as the loop runs.

**Listing 4.13:** Training step definition.

```
1 def training_step(self, step, test_data, training_data):
2  #(...)
3  if step % 500 == 0:
4   self.save_checkpoint(sess, checkpoint_prefix, step)
5
6  sess.run(self.train_step, feed_dict={self.X_input: batch_X,
7           self.Y_groundTruth: batch_Y, self.lr: learning_rate,
8           self.pkeep: 0.75})
```

At the end of each test iteration, values of the assigned variables with the method `summary.scalar` are updated. Therefore, at the end of training phase, a graphic of each variable evolution, regarding each test iteration, is generated by TensorBoard and can be analyzed on the platform provided for this purpose. The method `add_summary` (Line 9) is responsible for this feature (Listing 4.12).

In order to prevent some kind of disaster that could interrupt abruptly the training cycle, during the loop, checkpoints are saved with model characteristics and current state (Line 3, Listing 4.13). Therefore, when some kind of phenomena break the training loop, the model's state, characteristics and property values when it was stopped can be restored, and the training loop can be resumed and continued. This preventive measure is performed at each half epoch.

#### 4.1.1.3 Model export

Recalling the Figure 3.4 from Chapter 3, the model and deployment pipelines are divided from the end of the model training. This border encompasses the model export to a TensorFlow Serving repository to be loaded by a specific server that handles all queries to the model. Accordingly, the process of export was implemented and it comprehended some steps.

The export step definition is presented in the code snippet below (Listing

4.14). Its responsibility is to export the model for a specific repository path. The `SavedModelBuilder` (Line 11) creates the export destination directory if it does not exist. This directory is composed by the directory base path, `export_path_base`, and also by the model version, `FLAGS.model_version`, that determines the current model version. As new model versions are exported, sub-folders are created for each one.

**Listing 4.14:** Export final stage definition.

```
1  def export_model(self, dropout_keep, export):
2   #(...)
3   legacy_init_op = tf.group(tf.tables_initializer(),
4                             name='legacy_init_op')
5   export_path_base = "/tmp/workloadAssessor"
6
7   export_path = os.path.join(
8    tf.compat.as_bytes(export_path_base),
9    tf.compat.as_bytes(str(FLAGS.model_version))
10  )
11  builder = saved_model_builder.SavedModelBuilder(export_path)
12  builder.add_meta_graph_and_variables(sess,
13                         [tf.saved_model.tag_constants.SERVING],
14                         signature_def_map={
15                          'prediction':
16                              prediction_signature,
17                          tf.saved_model.signature_constants.
                                DEFAULT_SERVING_SIGNATURE_DEF_KEY:
18                              classification_signature,
19                         },
20                         legacy_init_op = legacy_init_op)
21  builder.save()
22  #(...)
```

Line 12 introduces the `builder` implementation. This module saves a replica of the trained model to a predefined repository so that it can be loaded later for inference. The model snapshot is defined by calling the `add_meta_graph_and_variables` method. This method takes as parameter the TensorFlow session `sess` that holds the trained model that is going to be exported. Since it was intended to use the graph in serving , the `SERVING` tag from `tag_constants` is used as a parameter to prepare the exported model to be loaded by a TensorFlow server. The `signature_def_map` parameter specifies the map of user-supplied key for

a signature to a default signature in order to add to the meta graph. Signature specifies what type of model is being exported, and the input/output to bind to when running inference. This parameter is composed by two fields: the `prediction_signature` and `classification_signature`. The `prediction_signature` defines which are the inputs and outputs that must be also defined in the client side, in order to be possible to perform a query to the server. Listing 4.15 shows the result of this signature implementation.

**Listing 4.15:** Predict signature definition.

```
def export_model(self, dropout_keep, export):
 #(...)
 tensor_info_x = tf.saved_model.utils.build_tensor_info
                                          (self.X_input)
 tensor_info_y = tf.saved_model.utils.build_tensor_info
                                          (self.Y_final)
 tensor_info_dropout = tf.saved_model.utils.build_tensor_info
                                                (self.pkeep)
 prediction_signature = (
  tf.saved_model.signature_def_utils.build_signature_def(
   inputs={'input': tensor_info_x ,
           'dropout': tensor_info_dropout},
   outputs={'level': tensor_info_y},
   method_name=tf.saved_model.signature_constants.
                               PREDICT_METHOD_NAME)
 )
 #(...)
```

The `inputs` argument specifies the input tensor info, `outputs` argument specifies the workload level classification info and the `method_name` argument is the method used for the inference, and since the intention is to make predict requests, it was assigned to the key `PREDICT_METHOD_NAME`. It is important to note that `input` and `level` are tensor alias names. From this point on, these names are logical names of tensor `X_input` and `Y_final` for tensor binding when sending prediction requests later on the process.

The TensorFlow Serving uses lookup tables for embedding or vocabulary lookups. Therefore, the tables initialization needs to be performed as a separate operation (in TensorFlow versions prior to 1.2). This operation is performed by the `legacy_init_op` argument.

Finally, a `save` method is defined in order to write the `SavedModel` protocol buffer to the export directory in serialized format.

### 4.1.2 Deployment

Preceding implementation phases where responsible for the model implementation, training and export to a specific repository. What follows in this implementation chain is the deployment of the exported model to its final environment from where it can be accessed.

The frameworks and their configuration that made possible the model deployment is described during this section. Therefore, it was divided in two sub-sections, concerning the TensorFlow model server and the client.

#### 4.1.2.1 TensorFlow model server

The deployment to production dynamics owns much to the TensorFlow server that serves the aspired models. For this purpose, TensorFlow provides a standard server that can serve any type of model: the TensorFlow model server.

The server's role is to dynamically discover and serve new versions of a trained TensorFlow model. One of its main features regards to the model versioning management. When two different model versions are dynamically generated at runtime, due to the implementation of a new algorithm or to the model training with a new dataset, it is intended that the newer model version is the one in the process of loading by the server while the older still serves before being unloaded. This paradigm is fully supported by the TensorFlow server. It dynamically discovers, loads and monitors newer versions of trained models and reverts older versions without breaking the serving service. The module that materialize this mechanism is the `ServerCore` (Listing 4.16).

**Listing 4.16:** ServerCore module definition.

```
1  int main(int argc, char** argv) {
2   #(...)
3   ServerCore::Options options;
4
5   if (model_config_file.empty()) {
6    options.model_server_config =
7     BuildSingleModelConfig(model_name, model_base_path);
8   }
9   else {
10   options.model_server_config =
11    ReadProtoFromFile<ModelServerConfig>(model_config_file);
12  }
```

```
13  #(...)
14  options.custom_model_config_loader = &LoadCustomModelConfig;
15  options.aspired_version_policy =
16    std::unique_ptr<AspiredVersionPolicy>(new
          AvailabilityPreservingPolicy);
17  options.file_system_poll_wait_seconds =
        file_system_poll_wait_seconds;
18
19  ::google::protobuf::Any source_adapter_config;
20  SavedModelBundleSourceAdapterConfig
21        saved_model_bundle_source_adapter_config;
22  source_adapter_config.PackFrom(
        saved_model_bundle_source_adapter_config);
23  (*(*options.platform_config_map.mutable_platform_configs())
24        [kTensorFlowModelPlatform].mutable_source_adapter_config
              ()) =
25    source_adapter_config;
26
27  std::unique_ptr<ServerCore> core;
28  TF_CHECK_OK(ServerCore::Create(std::move(options), &core));
29  RunServer(port, std::move(core), use_saved_model);
30
31  return 0;
32 }
```

The method `Create()` creates the TensorFlow server and takes a `ServerCore::Options` parameter that enables its configuration (Lines 3 and 28). It enables the specification of models that are intended to be loaded and maps them with their base paths (Line 6). The `SavedModelBundle` is a key component of TensorFlow Serving. It composes a TensorFlow model that was already loaded from a provided path and supplies a `Session::Run` interface to run inference, like in TensorFlow. The export path is then provided to the `Loader` component that enables its lifetime management by the `Manager` module. The `ServerCore` instantiates a specific implementation of `Manager`, the `AspiredVersionsManager` module. Therefore, whenever a new model version is discovered under the export path, this module loads the new version and unloads the old one.

Another feature implemented by the model server is the batching (Listing A.1). This mechanism allows the batching of multiple requests into a single request, reducing the cost of performing inference. It is an optional feature that can be turned on whenever desired. Therefore, providing a `SessionBundleConfig` when creating the `SavedModelBundleSourceAdapter` turns on this functionality

(Line 9). Batching has some parameters, `BatchingParameters`, that enables the customization of its mechanism, such as timeout, batch size, etc (Line 4).

**Listing 4.17:** ServerCore module definition.

```
1  SessionBundleConfig session_bundle_config;
2
3  if (enable_batching) {
4    BatchingParameters* batching_parameters =
5        session_bundle_config.mutable_batching_parameters();
6    batching_parameters->mutable_thread_pool_name()->set_value(
7        "model_server_batch_threads");
8  }
9  *saved_model_bundle_source_adapter_config.
10     mutable_legacy_config() = session_bundle_config;
```

When the batch is fully completed, the inference requests are internally embedded into a single large request, and a `tensorflow::Session::Run()` is invoked (where the GPU efficiency gain comes from).

After the model input batch arrives and it was already handled by the TensorFlow model server, a estimation request runs. The estimation process uses a predict implementation, `TensorFlowPredictImpl::Predict`. This function requests the `SavedModelBundle`, that has all information regarding the loaded model, to the `Manager`, through the `ServerCore` module. Then, it uses the generic signatures to map logical tensor names in `PredictRequest` to real tensor names and bind values to tensors, as already explained in Section 4.1.1.3. Lastly, it runs inference in order to classify the model input data.

Afterwards the model server inference process ends, the inference result is replied to the right client. A huge part of the server behavior can be customized according to user's needs, such as the model discover method, the model management and so on.

#### 4.1.2.2 TensorFlow Serving client

The entity that performs the inference requests and is interested in receiving the model classification result is the client. It is fully built and customized by the user. However, the client's communication structure must comply with the gRPC framework in order to establish a request-reply connection with the TensorFlow model server.

The client implementation begins with the connection definition that targets

the TensorFlow model server (Listing 4.18). After the channel conception, a prediction service is created through where all prediction requests are sent (Line 5). This service complies with the gRPC framework in terms of how the request is serialized to a *protobuf* structure and how it is communicated with the correspondent server.

**Listing 4.18:** Connection structure implementation.

```
1 def do_inference(hostport, num_batch):
2  #(...)
3     host, port = hostport.split(':')
4     channel = implementations.insecure_channel(host, int(port))
5     stub = prediction_service_pb2.
6            beta_create_PredictionService_stub(channel)
7  #(...)
```

Since the client is now connected to the TensorFlow model server, the prediction request must be created and prepared (Listing 4.19). It starts with the creation of the predict request object that defines the request details regarding the model name and the signature identification that was defined during the model export implementation (Section 4.1.1.3). These details must follow the model export definition so that the client request sent to the server can be accepted. Then, the input data is reshaped in order to meet the model input requirements and communicated to the request object. The request message is now completed and ready to be sent to the model server.

**Listing 4.19:** Request creation and preparation.

```
1 def do_inference(hostport, num_batch):
2  #(...)
3     request = predict_pb2.PredictRequest()
4     request.model_spec.name = 'workloadAssessor'
5     request.model_spec.signature_name = 'prediction'
6
7     data = test_data_set[int(num_batch)].reshape(1, 544)
8     request.inputs['input'].dtype = types_pb2.DT_FLOAT
9     request.inputs['input'].CopyFrom(tf.contrib.util.
10                    make_tensor_proto(data, dtype=tf.float32))
11 #(...)
```

The prediction result is given by the `Predict.future()` method (Line 5 in Listing 4.20). It takes as parameters the request message, previously built, and a number that specifies the maximum time that the inference response can

take until a timeout is reached. The inference result is a structured message with information about the model's output tensor, namely the result of the prediction obtained by the Workload Assessor model. Furthermore, the prediction request-response temporal duration was computed in order to have an estimative measure regarding the time that the request from the client, the model inference and the arrive of the predict result take (Lines 3 and 7).

**Listing 4.20:** Prediction request establishment.

```
1 def do_inference(hostport, num_batch):
2  #(...)
3     start_time = datetime.datetime.now()
4
5     result_future = stub.Predict.future(request, 5.0)
6
7     duration = datetime.datetime.now() - start_time
8  #(...)
```

Since the response from the server is a complex message, it must be decomposed (Listing 4.21). For this purpose, the message is processed and returned only the part related to the prediction probabilistic result (Line 3). It is composed by an array with three columns that represents the probability attributed to each workload level. Since the last layer is composed by a Softmax activation function, the model output values are very close to 1, or even 1, if one output class (*i.e.*, *low, normal* or *high*) is chosen by the model as being the most correct, and very close to 0 if the other output classes are not the model prediction choice. Therefore, the decomposed message is processed in order to find which is the most scored class, and, consequently, which workload level the driver holds.

**Listing 4.21:** Prediction result and computation implementation.

```
1 def do_inference(hostport, num_batch):
2  #(...)
3     prediction = result_future.result().
4                                 outputs['level'].float_val
5     prediction_result = numpy.argmax(prediction)
6     return prediction_result
```

Knowing which workload level the driver currently holds, the next step is to transmit this information to the Workload Manager so that if it is at abnormal levels it can be normalized.

## 4.2 Workload Manager

The Workload Manager system represents one of the most challenging component due to the lack of information and literature report concerning workload management developed systems. Therefore, both design and implementation approaches were performed from scratch and without literature support. However, it turns out that this system represents an important role in managing the driver workload level and in providing solutions (*i.e.*, actions) to the HMI systems in order to normalize possible abnormal workload levels.

As mentioned in 3.1.2, the system role is to propose actions to the HMI system in order to normalize levels of workload considered not normal (*i.e.*, overload or underload). Based on driving context, driver workload level, driver activity, and driver profile, the Workload Manager system selects a set of actions and chooses one to send to the HMI system for its application, taking into account the driver's history and the current driving context.

In order to better understand the workflow and the interface between all components that will be addressed during this section, the Figure 4.4 recalls the previously specified Workload Manager diagram.

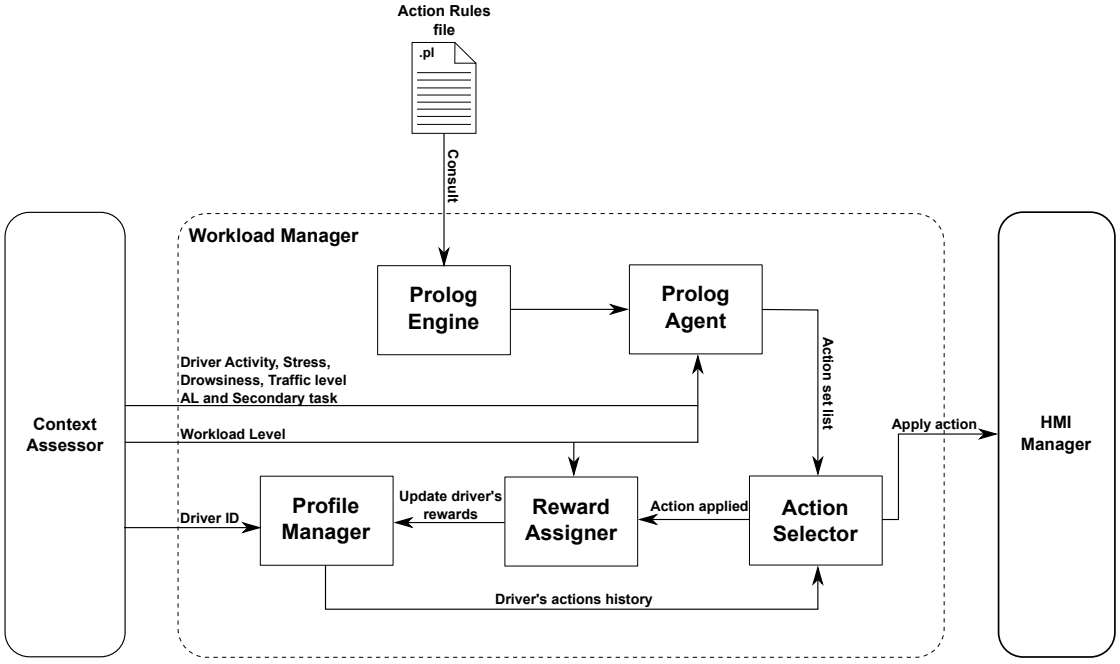



**Figure 4.4:** Workload Manager blocks diagram.

This system implements a group of components that provides all aforementioned features. The Prolog mechanism implemented will be described in Subsec-

tion 4.2.1. The Driver's Profile management and the action historic construction are explained in detail in 4.2.2. Moreover, the action selection behavior and criteria will be presented in 4.2.3. Lastly, Rewards assignment structure implementation and what influenced the choice of this approach will be described in Subsection 4.2.4.

### 4.2.1 Action Rules access

For the purpose of assigning a directive to the HMI system for its application, actions were specified and rules to access them were also described. The first step was to specify all concepts regarding the domain in which this systems works. Prolog programs describe relations mainly defined by means of two clauses: facts and rules. Facts are the basic unit in Prolog and represent a predicate, which is postulated true. This unit is composed by a head and a set of arguments. For example:

$$father(daniel)$$

$$father \rightarrow head$$

$$daniel \rightarrow argument$$

Therefore, the example above represents knowledge that can be accessed when desired. This fact informs to the knowledge base that 'daniel' is a 'father', being 'father' the fact's head and 'daniel' its argument.

The implemented action rules file is composed by a first part where a set of facts are specified as a knowledge base addressed to its domain of use. A snippet of this file is presented in Listing 4.22.

**Listing 4.22:** Knowledge base file snippet where facts are specified.

```
1  wlLevel(overload).
2  wlLevel(underload).
3
4  trigger(weather).
5  %(...)
6
7  cause(weather, overload).
8  %(...)
9
10 automationLevel(zeroOne).
```

```
11 %(...)
12
13 action(prioHMIinfo, weather, zeroOne).
14 action(switchForAS, weather, zeroOne).
15 action(switchForAS, traffic, zeroOne).
16 action(turnONmusic, physiologic, zeroOne).
17 action(turnONmusic, drowsiness, zeroOne).
18 action(prioHMItasks, traffic, zeroOne).
19 %(...)
```

The set of facts that can be noticed from the presented file encompass `wlLevel`, `trigger`, `cause`, `automationLevel` and `action`. Although most of facts are self-explanatory, the meaning of each one is described in the following points (see Section 3.1.2.1 and consult Table 3.2):

- `wlLevel` - represents the knowledge regarding driver workload levels that are considered abnormal states (*e.g.*, overload or high workload, and underload or low workload).
- `trigger` - represents the set of abnormal workload levels triggers (*e.g.*, weather, traffic, physiologic, drowsiness, and non-driving related tasks).
- `cause` - specifies for each trigger which workload level it can induce (*e.g.*, bad weather can be induce an overload state in the driver).
- `automationLevel` - represents the automation levels gap that the car can present (*e.g.*, zero to one, two to three and four to five).
- `action` - represents the set of available actions that can be attributed regarding a specific trigger and automation level.

Afterwards the definition of all domain concepts (*i.e.*, facts), rules had to be assigned in order to specify the Prolog agent behavior. In Prolog, the second predicate type is the rule or clause. It encompasses a head and a body. A rule example is:

$$Head :- Body$$

The rule is read as "Head is true if Body is true". Both the rule's body can be composed by facts.

In fact, in this case, rules were aimed to be used as a validation technique, verifying if some sets of facts could be used together to get the list of actions to

send to the HMI system. For example, in this domain and in the created knowledge base file, it can not be possible to a driver to hold an overload state and the trigger is drowsiness. These types of anomalies are barred by this mechanism. The rules specification is part of the knowledge base file were all the facts are described and is presented in Listing 4.23.

**Listing 4.23:** Validation rules.

```
1 %(...)
2 simple_validation(X,Y,Z) :- wlLevel(X), trigger(Y),
3                             automationLevel(Z).
4
5 cause_validation(X,Y) :- cause(X,Y).
6 %(...)
```

In the first statement (Line 2), a simple validation is performed in order to check if the arguments from the facts `wlLevel`, `trigger`, and `automationLevel` were previously specified as part of the knowledge. Lastly, Line 5 represents the cause validation step, which verifies if the trigger and workload state are a valid pair.

For the purpose of getting a list of actions, a simple rule was also specified in this knowledge base file. Listing 4.24 represents the rule implementation. It simply takes three arguments: `X` is the argument that will return the possible action based on the arguments `Y` and `Z`, `Y` is the argument that holds the received driver's workload level trigger, and `Z` represents the automation level range where the vehicle is operating.

**Listing 4.24:** Get action rule.

```
1 %(...)
2 get_action(X,Y,Z) :- action(X,Y,Z).
3 %(...)
```

A Prolog engine, the PySWIP - a Python-SWI-Prolog bridge enabling to query SWI-Prolog in Python programs, was used in order to implement the access to the Prolog file that has all actions specified and represents a knowledge database. The interface implementation between the engine and the Prolog file is quite straightforward (Listing 4.25).

**Listing 4.25:** Prolog interface implementation.

```
1 def main(_):
2  #(...)
3     prolog = Prolog()
4     prolog.consult("dwmKnowledge.pl")
5  #(...)
```

Initially, a Prolog agent is created and the Prolog knowledge file is associated to it. Thus, when queries are sent to this agent, it will aim the associated knowledge database.

Afterwards, all queries are built with proper values and ready to be sent to the Prolog agent (Lines 3, 5 and 7). Their implementation is described in Listing 4.26.

**Listing 4.26:** Prolog validation process.

```
1 def main(_):
2  #(...)
3  validation_query = "simple_validation(" + wlLevel + "," +
4                       trigger + "," + automationLevel + ")."
5  cause_query = "cause_validation(" + trigger + "," +
6                  wlLevel + ")."
7  getAction_query = "get_action(X," + trigger + "," +
8                      automationLevel + ")."
9
10 actions_list = []
11
12 if bool(list(prolog.query(validation_query))) & bool(list(
      prolog.query(cause_query))):
13  for soln in prolog.query(getAction_query):
14   actions_list.append(soln["X"])
15  print actions_list
16 else:
17  try:
18   raise ValueError('----->ERROR!!!! Prolog validation was
        unsuccessfully taken.<-----', wlLevel, trigger,
        automationLevel)
19  except ValueError as err:
20   print (err.args)
21   error_flag = True
22  break
23 #(...)
```

As mentioned earlier in this section, the Prolog agent is queried in order

to evaluate if the received arguments are valid (Line 12). If the validation is performed without any problem, the list of actions is returned to an array that holds all possible actions regarding the driving context (Line 14). If the validation do not pass, an error is raised in order to terminate the actions search and to report it (Line 18).

Since the list of actions is already known, it will be used to later discriminate the best suited action to apply to a specific driver base on the current context.

### 4.2.2 Driver Profile Manager

The driver profile manager component role is responsible to address all issues related to the driver profile (*e.g.*, creating a driver entry, update the driver profile, etc). Programmatically, the component is represented by a class, `DriverProfile Manager`, that encompasses methods for the driver's profile management. The driver's name is used to create a `DriverProfileManager` object (Listing 4.27, Line 5). Then, the driver's profile is restored in order to be used during the action choice and the workload management (Line 7).

**Listing 4.27:** Driver profile management.

```
1 def main(_):
2  #(...)
3  while 1:
4  #(...)
5   driverProfileManager = DriverProfileManager(driver_name)
6
7   driverProfileManager.restoreProfile()
8  #(...)
```

The process starts by verifying the profiles file tree in order to check if a driver has already a created profile and, in case it does not have, a new one is created (Listing 4.29, Line 4). In case the driver profile already exists, a JSON file is loaded. This file has registered all action decisions sent to the HMI system convoluted with the context variables captured in that moment (*i.e.*, when an action is selected and sent to the HMI system, the result from its application in the driver workload level is saved as form of rewards). Its structure is described in Listing 4.28. All keys and values used in the JSON file structure are defined in the Prolog knowledge database file, excepting the `"rewards"` and `"profile"` keys.

**Listing 4.28:** JSON file example representing driver rewards in a specific driving context.

```
1  {
2   "profile":
3   {
4    "wLevel":
5    [
6     "name": "overload"
7     "trigger":
8     [
9        "name": "weather"
10       "automationLevel":
11       [
12         "name": "oneTwo"
13         "rewards": [0,1,2,2] ...
14       ]
15    ],
16      (...)
17   ],
18   (...)
19  }
20 }
```

After the JSON file loading, the data contained by it is parsed in order to restore all values from keys into a personalized structure. This structure is then appended to an HashTable and mapped to the reward vector (Line 18). This vector represents the reward score that each action from the result Prolog query list has attributed, and regarding a specific driving context. After the execution of the driver profile restore, `restoreProfile()`, the driver's history is known and can be used in further action selections.

**Listing 4.29:** Driver's profile restore.

```
1  class DriverProfileManager:
2   #(...)
3   def restoreProfile(self):
4    restore_profile = self.checkRestore()
5
6    if restore_profile:
7     file = open(self.driver_profile_path + "/profile.json", 'r')
8     profile_data = json.load(file)
9
10    for i in profile_data['profile_actions']:
11     for j in profile_data['profile_actions'][i]:
```

```
12        #(...)
13        rewards = np.array(tmp['rewards'])
14        wLevel_name = tmp_wl['name']
15        #(...)
16        trigger_name = tmp_tr['name']
17        automationLevel = tmp_al['name']
18        self.hashTable[struct(wLevel_name, trigger_name,
19                              automationLevel)] = tmp_rewards
20    #(...)
```

The current driving context information (*i.e.*, driver workload level, trigger and automation level) are stored in a personalized data structure and used during the driver's session until new context information arrives (Listing 4.30, Line 5). This structure is then used as an argument in the `getRewards()` method in order to get the action rewards vector for the current context (Line 7), from the HashTable. If no action rewards vector is returned, it means that there is no driver's history stored in the profiles file tree and, therefore, will have to be later saved in it.

**Listing 4.30:** Driver's profile restore.

```
1  def main(_):
2   #(...)
3   while 1:
4    #(...)
5    driver_struct = struct(wlLevel, trigger, automationLevel)
6
7    rewards = driverProfileManager.getRewards(driver_struct,
8                                              len(actions_list))
9    #(...)
```

### 4.2.3 Action Selector

The Action Selector module works has a moderator, deciding which action is selected to be sent to the HMI manager. It uses the rewards attributed to each action from the returned list of actions obtained from the driver profile history. Then it returns the index of the action that was selected as the best suited to be applied by the HMI system (Listing 4.31). The method that deals with the action selection is the `getAction()`.

**Listing 4.31:** Driver's profile restore.

```
1 def main(_):
2  #(...)
3  while 1:
4   #(...)
5   action_index = driverProfileManager.getAction(rewards,
6                                                  len(actions_list))
7   chosen_action = actions_list[action_index]
8   #(...)
```

As aforementioned, the `getAction()` method takes as argument the action rewards array taken from the driver profile history (Listing 4.32). Therefore, if actions were already applied during a driver session, its rewards history will present different values for each action. Applying a Reinforcement learning principle, an exploration process is performed in order to propose actions that maybe not be frequently chosen due to its low reward rank. This process is performed in a established frequency (Line 4), and a random action index is chosen (Line 6) and returned later. On the other hand, when the exploration method is not used, the chosen action is the one that has the higher reward score, meaning that is the action that had the best results in the normalization process (Line 9).

**Listing 4.32:** Driver's profile restore.

```
1 class DriverProfileManager:
2  #(...)
3     def getAction(self, rewards, actions_len):
4         e = 0.1
5         if np.random.rand(1) < e:
6             action_index = np.random.randint(actions_len - 1)
7             print "Doing some exploration."
8         else:
9             action_index = np.argmax(rewards)
10            print action_index
11
12        return action_index
13  #(...)
```

After the action discover, it must be sent back to the HMI Manager in order to be posteriorly applied to the HMI system. The decision to select which modality (*i.e.*, auditory, haptic, visual) to apply or when to apply is entirely the responsibility of the HMI Manager to decide. Therefore, it is possible to keep an independent relation between components and, therefore, achieve an holistic

modular architecture. Recalling the earlier mentioned in the specification chapter, the Workload Manager only proposes actions or directives to the HMI Manager.

It is expected that after the application of the proposed action in the HMI system, the driver's workload level changes to a normal state. Therefore, the resulting workload level must be assessed in order to evaluate if the applied action caused impact in the driver's workload. The next sub-component is responsible for this process.

### 4.2.4 Reward Assigner

Reinforcement learning concepts also influenced the mechanism that deals with driver's workload level change after the HMI Manager applies it to the system. Furthermore, a reward approach was implemented in order to classify how good the driver workload level change occurred. The reward conceptualization was implemented through a flexible structure that allows out-of-the-box improvements. When the new driver's workload level is received, it is processed by a function, `givereward()` (Listing 4.33), that verifies if the current level is 'normal', meaning that the abnormal level was normalized, or remains in an not desirable workload state. If the normalization process was successfully performed, a positive reward is attributed to the applied action. When the contrary is verified, negative rewards are also attributed to the previously proposed action. Lastly, the rewards action array is updated with the action reward returned by the aforementioned function.

**Listing 4.33:** Driver's profile restore.

```
1 def main(_):
2  #(...)
3  while 1:
4   #(...)
5   rewards[action_index] += driverProfileManager.giveReward(
6                                                    result_state)
7   #(...)
```

Regarding errors reported by the HMI manager after the proposed action application, if this system find some kind of problems regarding the action itself and the impossibility to its application, this anomaly is sent back to the Workload Manager system in order to propose a new action to the HMI Manager. This mechanism is performed until no errors are reported by the injured system. When the driver exits the car, its session is closed and the profile saved with the new

information generated during the driving time. The driver's profile is mapped using its identification method (*e.g.*, driver's name, generated ID, etc) and, when necessary, the consequent load is performed using this same method.

# Chapter 5

# Experimental Results

In the previous chapter, the system's development process was explained.

Regarding the present chapter, the results obtained after the implementation phase will be presented. Regarding the Workload Assessor system, results focus mainly on the model performance, using classification performance metrics for the purpose, and also on the model serving (*i.e.*, Model Server). Concerning the Workload Manager, qualitative results will be presented regarding the effectiveness of the Actions Provider mechanism after the driver workload assessment.

Therefore, the first section (5.1) starts by presenting (5.1.1) the significance of the set of features used in the model training step. In the second section (5.1.2) the Workload Assessor model performance is evaluated through specific metrics, following an error incremental analysis approach. The last section (5.1.3) presents the server loading and serving the Workload Assessor model to possible clients.

In the second part, the Workload Manager system will be addressed (5.2). The first and second sections (5.2.1 and 5.2.2, respectively) describes the selection of a countermeasure for abnormal workload levels normalization and the reward assignment based on how effective the action was.

Lastly, the third part (5.3) briefly presents some results and considerations regarding the subjective workload assessment questionnaire, DALI. A statistical approach is addressed to all subjects answers in order to verify the presence of relevant patterns among them.

## 5.1 Workload Assessor

The Workload Assessor component has a key role in the overall system. As such, an analysis regarding the model assessment quality was performed and specific machine learning metrics used in order to evaluate its performance. These metrics encompass *accuracy*, *precision*, and *recall*. In machine learning, *accuracy* can be seen as the fraction of correct predictions in relation to the total predictions made. Moreover, *precision* is the fraction of relevant retrieved instances (*i.e.*, true positives) among the retrieved instances (*i.e.*, true positives and false positives), also known as positive prediction rate, while *recall*, also known as sensitivity, is the fraction of the retrieved relevant instances (*i.e.*, true positives) over the total amount of relevant instances (*i.e.*, the total of true positives and false negatives for a given class).

The model accuracy across all training steps was computed using a method provided by TensorFlow API. Regarding precision and recall, they were computed manually using the confusion matrix obtained in each training step, using the following equations:

$$Precision = \frac{tp}{tp + fp} \tag{5.1}$$

$$Recall = \frac{tp}{tp + fn}, \tag{5.2}$$

where $tp$ is the number of true positives, $fp$ is the number of false positives and $fn$ is the number of false negatives.

All training trials were based on a partial dataset division, meaning that the collected dataset during the experiment was divided in two sets: training (70%) and test (30%) [10, 11, 12, 13, 14, 59, 95].

### 5.1.1 Rank of Variables

Based on the methodology described in the Chapter 3 (Section 3.3.3), the *p-value* was computed to each of the 15 features and their significance level was found. Therefore, all features that had a significance level above 0.05 (*i.e.*, $> 0.05$) were considered as not significantly influence in the driver's workload level. The group of features with significance level below 0.05 are ranked as shown in Table

5.1.

After the model training with the most significant features, an accuracy in a range of 98 to 99% was obtained. This result was analyzed post-training and the level of significance between the features duly compared. It was noticed that the *p-value* of the 5 most significant features (Table 5.1) was very close to, if practically equal, zero. Thus, due to the fact that the experiment was carried out with very little diversity of scenarios and situations, there were eventually features that in a way indicated too explicitly the conditions where the few scenarios occurred and that indirectly revealed the conditions for the creation or imposition of a certain level of workload on the subject. For example, the high workload experiment was performed in a high traffic level environment. Therefore, the feature that translates the traffic level will erroneously represent the driver's high level of workload and significantly influence the model's training.

**Table 5.1:** Ranking of the top 12 features with significance levels less than 0.05, from the training dataset.

| **Feature** | **Rank** |
|---|---|
| Road Characteristics | 1 |
| Traffic Level | 2 |
| Road Type | 3 |
| Weather Conditions | 4 |
| Time of Day | 5 |
| Velocity | 6 |
| BioPack Heart Rate | 7 |
| PERCLOS | 8 |
| Blink Rate | 9 |
| Brake Pedal | 10 |
| Longitudinal Acceleration | 11 |
| Radar Respiration Rate | 12 |

In order to avoid this issue, the first 5 most significant features were removed. Moreover, although the Velocity feature does not have a *p-value* equal to zero, its value is also very low ($< 0.0001$). The influence that this feature presents can be equally comparable to the 5 most significant due to the scenarios and conditions where these were collected. For example, the low workload scenario occurred in a highway environment where the car velocity is usually much higher than in the other two scenarios. Thus, this feature presents a greater magnitude in this scenario wrongly influencing the model during its training and should be removed from the training dataset.

The final set of features that compose the ranking and that was used to produce the dissertation's results is ordered by level of significance in Table 5.2.

**Table 5.2:** Ranking of the top 6 features with significance levels less than 0.05, from the training dataset.

| Feature | Rank |
|---|---|
| Biopack Heart Rate | 1 |
| PERCLOS | 2 |
| Blink Rate | 3 |
| Brake Pedal | 4 |
| Longitudinal Acceleration | 5 |
| Radar Respiration Rate | 6 |

### 5.1.2 Error incremental analysis

As stated in the previous chapter, an error incremental analysis was performed in order to verify and evaluate which group of ranked features (Table 5.2) produces the best classification results. The objective of the incremental error analysis is to determine the number of top-ranked features that should be used in order to obtain the best classification results. The analysis procedure had as its starting point the top-ranked feature, adding to the training dataset, iteratively, the next best-ranked feature until all significant features were included.

Therefore, for each group of top ranked features, six training trials were performed and the aforementioned metrics were computed for each one. The average of all six trials concerning to each group of top ranked features included in the dataset is depicted in Figure 5.1. In this step, the test dataset was used to compute the accuracy scores.

The graph shown in the Figure 5.1 represents the average accuracy, in percentage, of the training performed with different numbers of top ranked features included in the dataset. Moreover, at each vertex in the graph line, an error bar is appended and represent the standard deviation regarding the obtained training accuracy for a specific dataset.

After the model accuracy computation, precision and recall were obtained in order to examine how training occurred and how well the model classifies. In the end of each training trial, the confusion matrix was generated and the aforementioned metrics calculated from it. The Figure 5.2 represents an example of a confusion matrix over a training trial with the best overall statistics when comparing with others. The matrix confronts the model's predictions and the ground

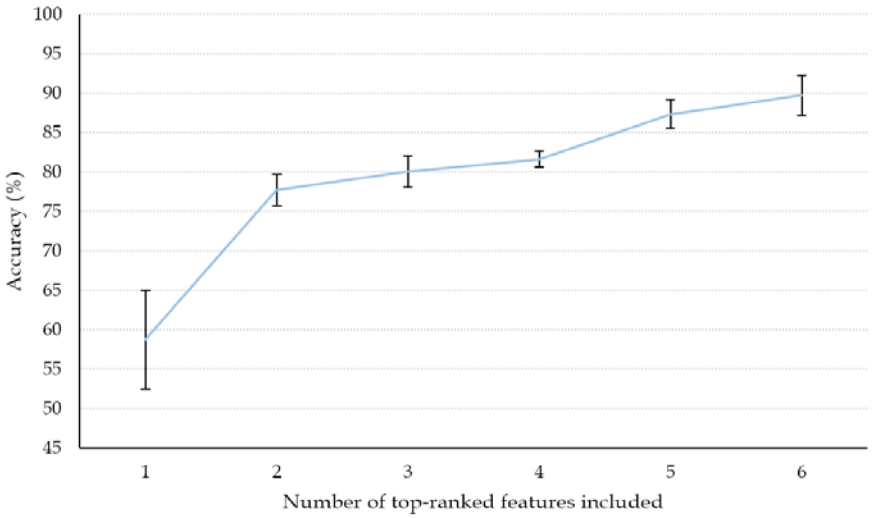



**Figure 5.1:** Incremental error analysis performance of accuracy with standard deviation indicated as error bar.

truth present in the training dataset. It represents in its diagonal the true positives in regard to each class (*low*, *normal* and *high* workload).

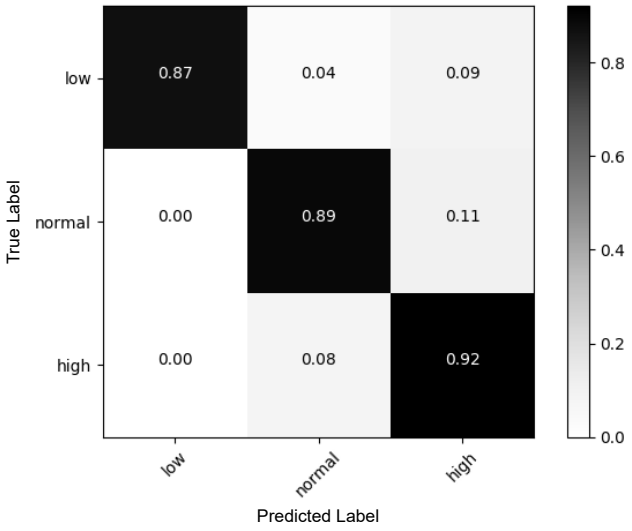



**Figure 5.2:** Normalized confusion matrix obtained from the best training of all trials, with 6 top-ranked features.

In this training trial (Figure 5.2), the model predicted correctly as *low* workload 87% of the time, according to the ground truth present in the training dataset. Regarding *normal* and *high* workload, the model had an accuracy ratio of about 89% and 92% respectively. The model classification was well performed since the

confusion matrix diagonal presents the most contrasting color when comparing to its remaining blocks (*i.e.*, the training trial generated a higher rate of true positives for each class, than false negatives or positives).

As aforementioned, based on the confusion matrix, the classification precision and recall metrics were calculated. These values and a summary of the classification accuracy is presented in Table 5.3, for each training performed with different groups of features.

**Table 5.3:** Results obtained from datasets with different number of features (error incremental analysis). Between parentheses are the minimum and maximum values obtained.

| **Nº of Features** | **Accuracy (%)** | **Precision (%)** | | | **Recall (%)** | | |
|---|---|---|---|---|---|---|---|
| | | **Low** | **Normal** | **High** | **Low** | **Normal** | **High** |
| 1 Feature | 59.0 (54.4-70.8) | 57.5 | 46.5 | 65.8 | 49.9 | 50.8 | 72.5 |
| 2 Features | 78.0 (75.7-80.9) | 87.2 | 69.6 | 64.3 | 86.7 | 68.1 | 68.5 |
| 3 Features | 80.1 (76.5-81.3) | 85.0 | 79.8 | 63.6 | 85.6 | 69.7 | 76.6 |
| 4 Features | 82.0 (79.7-82.5) | 82.6 | 74.5 | 74.8 | 93.6 | 73.1 | 70.8 |
| 5 Features | 87.3 (84.6-89.9) | 87.7 | 84.6 | 74.5 | 89.3 | 75.8 | 85.1 |
| 6 Features | 90.0 (88.2-94.5) | 87.7 | 81.5 | 82.1 | 90.8 | 81.1 | 81.9 |

Analyzing the figure and table above it is possible to verify the influence that the increment of top-ranked features in the training dataset has on the model performance and consequently in its classification assertiveness. As much features are included, the model accuracy increases reaching a maximum of 90%, in average, when all 6 top-ranked features were included. However, it takes only two top-ranked features to be included in the training dataset in order to obtain a reasonable average accuracy percentage (*i.e.*, the training dataset with 2 top-ranked features). Regarding precision and recall, from the 4 top-ranked features included interesting values were generated, achieving good results with 6 top-ranked features included, in what these two metrics relate to (*i.e.*, precision and recall with values above the 80%).

### 5.1.3 Model Server

As aforementioned in the previous Chapter 3, Section 3.1.1, the Model Server holds the exported trained model and works as an intermediate when one or various components of the overall architecture want to classify a specific batch of data.

After the training step, the model with trained parameters was exported in

order to be loaded by the server. The model loading process can be visualized in Figure 5.3.


**Figure 5.3:** Model Server searching in the repository path for a model exported and loading it. Blurred image section concerns to extremely verbose information resulting from the model configuration setup and the reservation of resources for the model loading process.

The figure above shows the searching mechanism that the model server implements, verifying a repository (in this example is pointing to the `/tmp/workload Assessor` path) and loading the model trained. A continuous scan in the pointed path is performed in order to check if a new model version was exported and, if so, load it (Figure 5.4).

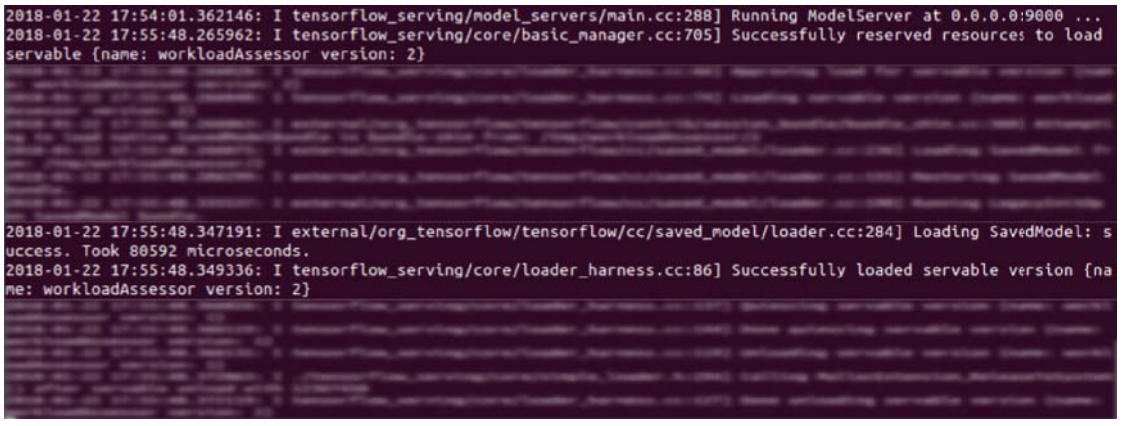



**Figure 5.4:** Model Server successfully detects a new model version and starts the allocation of new resources for the new loading. The blurred sections represent excessively verbose informations and concerns to the creation of a new model configuration and the unloading of the previous model version.

Afterward the model loading, Model Server is ready to receive queries from clients. Therefore, a batch of data was isolated and queried by a custom created client. The query sent by the client and received by the Model Server is depicted in the figure below (Figure 5.5).

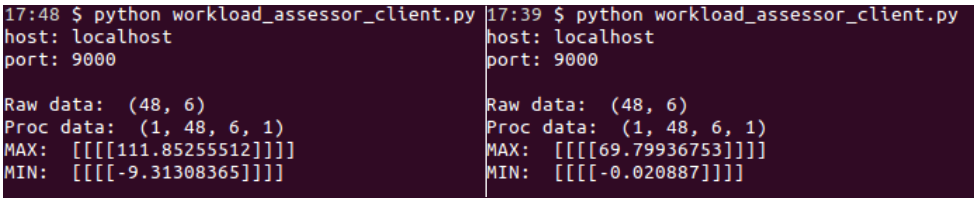



**Figure 5.5:** Representation of two different client requests encompassing two distinct data batches: high and low workload data, respectively.

When the Model Server receives the query with the data batch, the trained model receives it and classifies that data. As a response, the server sends back to the client an array of probabilities regarding each model class (*i.e.*, *low*, *normal* and *high*). The class with the highest probability represents the most likely class to represent the data batch previously sent. This behavior is proven by the below illustration (Figure 5.6).

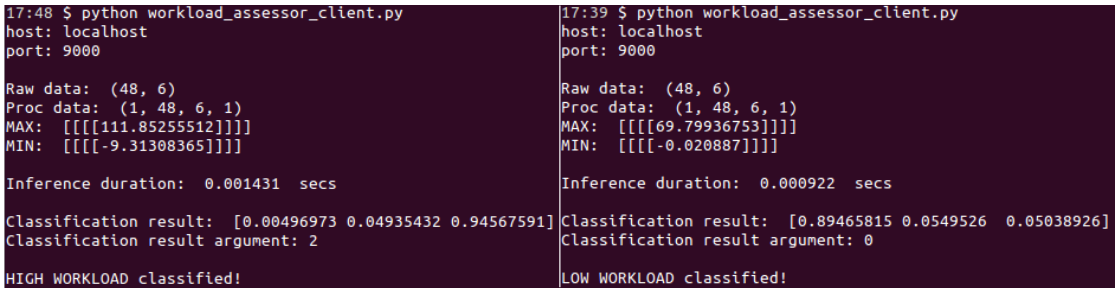



**Figure 5.6:** Representation of the Model Server reply from the two client requests presented in the previous figure, respectively (Figure 5.5). This reply encompasses the probability for each class and the correspondent classification.

## 5.2 Workload Manager

Regardless the fact that this system is tightly linked to the results dropped by the WA, the Workload Assessor, it still has a crucial role in the final stage of normalizing unwanted abnormal workload levels. As its operation was already described in Chapter 3, Section 3.1.2, the goal is to propose actions to the HMI system in order to normalize abnormal workload levels. Therefore, one of the most important inputs comes from the Workload Assessor and gives information about

the driver's workload level. From this information and additional knowledge given by other systems in the holistic HMI system architecture, the Workload Manager proposes a set of actions that are personalized for each different driver and sends it to the HMI system in order to be later applied. Depending on how well the action application occurred, the system behaves accordingly.

This section aims to evaluate the communication between these three systems (*i.e.*, Workload Assessor and Manager, and the HMI system) and if the Workload Manager correctly purposes actions to the HMI system and behaves to possible responses from it.

In order to test the Workload Manager system, the driver's workload level classified in the Model Server component will be directly sent to it. Despite the fact that in the project's architecture this system acquires the workload level and driving context information from the Context component, this test approach proves the concept.

### 5.2.1 Action Selection

One of the Workload Manager cores relies on the action selection and its communication to the HMI system. Therefore, for the results that will be later presented, driving context information was simulated and the driver workload level was obtained using the previously data batch example (Section 5.1.3).

Figure 5.7 shows the results regarding the Workload Manager system and the output actions that best counteract a possible abnormal workload level.

```
hmi@ubuntu:~/PycharmProjects/DriverWorkloadManager$ python dmwLP.py
Driver Workload Manager running with Prolog SWIP engine.

Driver ID:
Driver1
Creating profile directory.

Current driver workload level:
Overload

Workload trigger:
Traffic

Current vehicle automation level:
zeroOne

['prioHMIinfo', 'switchForAS', 'limitUseNDR', 'aromatic', 'turnONmusic', 'prioHMItasks', 'visualAudito
ryWarn', 'turnONlights']

Chosen action index:  0
The action for an 'Overload' wlLevel, triggered by 'Traffic', is 'prioHMIinfo'
```

**Figure 5.7:** Action selection based on the array of actions returned by the Workload Manager. In this high workload example, the selected actions dictates to prioritize the information displayed in the HMI (*i.e.*, abbreviation of `prioHMIinfo`).

By the above illustration, the set of actions that this system considers to be the best suited to combat the abnormal workload level. In this example, the driver workload level is *high* (*i.e.*, *overload*), the abnormal driver workload trigger is the traffic and he is driving in an automation level between 0 and 1. As can be verified, the system creates the profile directory if a new user entered the car. Moreover, it correctly returns all possible actions (*i.e.*, represented in the figure in form of an array) when comparing to the rules from the Prolog file with the knowledge base (Appendix A). Moreover, since just one action can be suggested, the system understands which one has the highest reward score, based on the driver history, and selects it to be sent to the HMI system. This can be visualized in the figure above by the action index chosen from the returned array of actions and, literally, by the terminal sentence field that is enclosed in quotation marks. Since the driver's profile does not have any reward assigned, the first action is the chosen one.

### 5.2.2 Reward Mechanism

The other component of the Workload Manager system relies on the reward mechanism that enables the assignment of positive or negative rewards whereas the driver workload level normalizes or not, respectively.

Afterwards the action dispatch to the HMI system in the previous section, it is necessary to understand if the suggested action took effect in the driver and act accordingly. Therefore, in the following example, the driver holds a new abnormal workload level: the *Underload* or *Low* workload. Moreover, a new set of actions is presented and, due to the fact that none reward was previously assigned, the first action is selected. Figure 5.8 depicts the system behavior after the new driver workload level classification.

Analyzing the result from Figure 5.8, the last figure section shows the recently new driver workload assessment. However, the abnormal state, *underload*, remains and, therefore, the Reward Mechanism must assign a reward discount to this action. Figure 5.9 shows the reward assignment result and, also, the `Driver1` profile.

```
Current driver workload level:
Underload

Workload trigger:
Drowsiness

Current vehicle automation level:
twoThree

['visualAuditoryWarn', 'turnONlights', 'switchBackAL']

Chosen action index:  0
The action for an 'Underload' wlLevel, triggered by 'Drowsiness', is 'visualAuditoryWarn'

Resulting driver workload level:
Underload
```

**Figure 5.8:** Action selection based on the array of actions returned by the Workload Manager (*Underload* context).

Based on the figure below (Figure 5.9), it can be concluded that the Reward Mechanism correctly attributes the reward factor in each situation. This profile information can be visualized when restoring the driver's profile.

```
Should restore driver profile.
State: Underload| trigger: Drowsiness| rewards: [-1  0  0]
State: Overload| trigger: Traffic| rewards: [1 0 0 0 0 0 0 0]
```

**Figure 5.9:** Driver's profile showing the set of abnormal workload levels detected and the correspondent array of rewards.

## 5.3 DALI questionnaire

The subjective workload assessment questionnaire was introduced in this dissertation as an extra workload measure. As already mentioned in previous chapters (Chapter 2, Section 2.1.1.3.1), this measures can produce different types of answers to a specific stimulus due to the fact that is a subjective measure and, therefore, different sensitivities are involved.

The following figure shows a statistical plot with information about the subjects that performed the experiment:

The figure 5.10 illustrates that the overall mean age range between 24 to 27 years. The ideal scenario would be to have a wider range in terms of subjects age but that was not possible due to time and project resources constraints. Moreover, about 21 subjects performed with success the designed experiment (14 male and 7 female subjects).

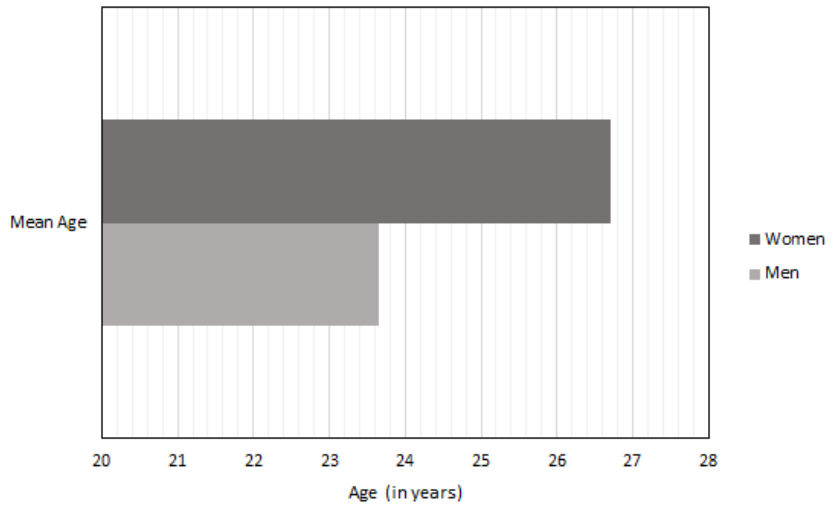



**Figure 5.10:** Chart representing the information about the average age between subjects and differentiating its gender.

Regarding the questionnaire itself, DALI was applied to each subject presents a specific set of factors and are used in a diverse range of scenarios. Appendix B (Figure B.1) shows the questionnaire format that was presented during the experiment. The methodology application specification of this questionnaire is properly described in Section 3.2 of Chapter 3.

### 5.3.1 Statistical Analysis

An analysis regarding the questionnaire reliability and based on the subjects answers was performed using the most common measure of internal consistency (*i.e.*, reliability): Cronbach's Alpha. It is widely used for multiple Likert questions[1] in questionnaires that form a scale and that intends to determine if the scale is reliable. Therefore, the aim of this step is to compute the Chronbach's alpha factor. A commonly-accepted assumption is that an alpha of 0.7 (others say 0.6) indicates acceptable reliability and 0.8 or higher indicates good reliability [107].

Two statistical analysis were performed in order to encompass the questionnaire answers from the Experiment 1 that encompassed the high and low traffic

[1] A Likert scale is a psychometric scale widely used for research reasons and that employs questionnaires. It is the most widely used approach to scaling responses in a questionnaire research [106].

scenarios, where each one intended to impose and analyze a high and normal workload level, respectively. Only the questionnaire answers regarding the scenario sections used to build the training dataset were included in this analysis.

#### 5.3.1.1 High Traffic scenario

The high traffic scenario statistical analysis upon the subjects questionnaire answers is presented in the Table 5.4.

**Table 5.4:** Results regarding the ranks given by subjects for each DALI factor. The N column represents the number of subjects that performed the respective High Traffic scenario.

| Factor Statistics | | | |
|---|---|---|---|
| **Factors** | **Mean** | **Std. Deviation** | **N** |
| Effort of Attention | 7.68 | 1.600 | 19 |
| Visual Demand | 7.32 | 1.797 | 19 |
| Auditory Demand | 6.21 | 2.175 | 19 |
| Temporal Demand | 6.26 | 1.939 | 19 |
| Interference | 7.42 | 1.644 | 19 |
| Situation Stress | 6.37 | 2.432 | 19 |
| Overall Workload | 2.79 | 0.419 | 19 |

Based on the table above, the mean values for each factor presents a fairly normal distribution. Moreover, the overall workload factor is not part of the DALI set of factors but was added in order to understand the perception that each subjects had regarding their inherent workload. The Cronbach's alpha computation is presented in form of a table (Table 5.5).

**Table 5.5:** Results regarding Cronbach's alpha computed for the subjects answers during the High Traffic scenario.

| Reliability Statistics | | |
|---|---|---|
| **Cronbach's Alpha** | **Cronbach's Alpha based on Standardized items** | **N of items** |
| 0.867 | 0.859 | 7 |

The reliability statistic shows a Cronbach's alpha value above 0.85, which is considered a good value. Moreover, analyzing all statistics computed based on the subjects answers (Table 5.6), it can be understood that the Cronbach's alpha value increases if the factor OW (*i.e.*, Overall Workload) is removed from the analysis.

**Table 5.6:** Overall statistics regarding values from the questionnaire answered by subjects performing the High Traffic scenario.

| Item-Total Statistics | | | | |
|---|---|---|---|---|
| **Factor** | **Mean if item deleted** | **Variance if item deleted** | **Corrected item-Total Correlation** | **C. A. if item deleted** |
| EA | 36.37 | 65.468 | 0.863 | 0.824 |
| VD | 36.74 | 64.427 | 0.672 | 0.830 |
| AD | 37.84 | 63.363 | 0.668 | 0.853 |
| TD | 37.79 | 62.842 | 0.732 | 0.831 |
| INT | 36.63 | 67.801 | 0.824 | 0.840 |
| SA | 37.68 | 59.561 | 0.572 | 0.854 |
| OW | 41.26 | 88.205 | 0.368 | 0.889 |

After removing this factor from the statistics, the new Cronbach's alpha value takes the rate of 0.889.

#### 5.3.1.2 Low Traffic scenario

The low traffic scenario statistical analysis upon the subjects questionnaire answers is demonstrated in the Table 5.7.

**Table 5.7:** Results regarding the ranks given by subjects for each DALI factor. The N column represents the number of subjects that performed the respective Low Traffic scenario.

| Reliability Statistics | | | |
|---|---|---|---|
| **Factor** | **Mean** | **Std. Deviation** | **N** |
| EA | 3.45 | 1.432 | 20 |
| VD | 3.60 | 1.273 | 20 |
| AD | 3.05 | 1.191 | 20 |
| TD | 3.55 | 1.395 | 20 |
| INT | 3.45 | 1.669 | 20 |
| SS | 3.00 | 1.622 | 20 |
| OW | 1.45 | 0.510 | 20 |

The table above presents mean values with a similar distribution. Based on the values described in Table 5.7, the Cronbach's alpha value is presented in the following table:

**Table 5.8:** Results regarding Cronbach's alpha computed for the subjects answers during the Low Traffic scenario.

| Reliability Statistics | | |
|---|---|---|
| **Cronbach's Alpha** | **Cronbach's Alpha based on Standardized items** | **N of items** |
| 0.926 | 0.925 | 7 |

The reliability statistic shows a Cronbach's alpha value above 0.90, which is considered a really good value. Moreover, analyzing all statistics computed based on the subjects answers (Table 5.6), there is no need to delete the OW factor because the increment in the Cronbach's alpha is not significant.

**Table 5.9:** Overall statistics regarding values from the questionnaire answered by subjects performing the Low traffic scenario.

| Item-Total Statistics | | | | |
|---|---|---|---|---|
| **Factor** | **Mean if item deleted** | **Variance if item deleted** | **Corrected item-Total Correlation** | **C. A. if item deleted** |
| EA | 18.10 | 42.832 | 0.899 | 0.901 |
| VD | 17.95 | 45.629 | 0.842 | 0.908 |
| AD | 18.50 | 47.316 | 0.793 | 0.914 |
| TD | 18.00 | 44.632 | 0.813 | 0.910 |
| INT | 18.10 | 40.832 | 0.849 | 0.908 |
| SS | 18.55 | 41.839 | 0.823 | 0.911 |
| OW | 20.10 | 58.305 | 0.406 | 0.944 |

### 5.3.2 Global DALI score

Founded on the scores provided by the subjects, a global workload score was computed based on the NASA-TLX method [108]. This global score aimed to give an orientation regarding the workload level that each subject perceived for each experiment and for its road sections. Therefore, a specific methodology was applied. All DALI factors were confronted with each other and whenever it was understood that one was more important than the other for the imposition of workload on the driver, a point was attributed to the same. In this way, we obtained factors that were most significant in relation to the imposition of high workload in scenarios designed for such. From these factors, a weight for each was assigned so that its contribution to the overall workload score was concordant. This weight is called the Tally. The Table 5.10 shows the tally distribution between factors.

**Table 5.10:** Tally computed for each factor based on its level of influence in the driver workload level.

| DALI Factor | Tally Score |
|---|---|
| Effort of Attention | $\frac{5}{15} = 0.333$ |
| Visual Demand | $\frac{4}{15} = 0.267$ |
| Situation Stress | $\frac{3}{15} = 0.2$ |
| Temporal Demand | $\frac{2}{15} = 0.133$ |
| Interference | $\frac{1}{15} = 0.067$ |
| Auditory Demand | $\frac{0}{15} = 0.0$ |

The tally score of each factor was then multiplied with the rank given by the subject for that same factor, during the experiment. Then, each one of the five weighted ranks are summed up and a global score is obtained. The mathematical statement that represents the global DALI score computation is presented in the equation 5.3.

$$DaliScore = \sum_{i=1}^{6} rank_{factor} * tally_{factor}, \tag{5.3}$$

where $factor$ represents each DALI factor, $rank$ is the score given by the subject and $tally$ is the weight computed for each factor.

The results computed for the DALI score, and concerning to each experiment (*i.e.*, Low and High Traffic scenarios), is presented in the table below (Table 5.11):

**Table 5.11:** Results regarding the global DALI score for the Low and High Traffic scenario.

| Global workload score | | |
|---|---|---|
| **Scenario** | **Average DALI score** | **Std. Deviation** |
| Low traffic | 3.41 | 1.32 |
| High traffic | 6.96 | 1.69 |

Based on the results provided by the table above, it can be confirmed that a coherent outcome was accomplished. Since the average DALI score for the High traffic achieved a value of approximately 7, it is coherent to obtain a near 3.4

value for the low traffic scenario due to the fact that a normal workload level was imposed and desirable.

# Chapter 6

# Conclusions

Lastly, in the final dissertation's chapter, conclusions obtained from all the performed work are presented (section 6.1) and suggestions for future improvements are proposed (section 6.2).

## 6.1 Discussion

For the beginning of this section of discussion, the dissertation's author intends to emphasize the challenging nature that work had both in the acquisition and understanding of demanding concepts, as well as in personal and work management. These thematics range from (i) obtaining knowledge regarding the applicability and structure of different Machine Learning algorithms in specific domains; (ii) the assimilation of concepts regarding different agents that make up a machine learning algorithm; (iii) a direct contact with a framework designed to develop algorithms in this area (*i.e.*, TensorFlow); (iv) and to the acquisition of concepts and techniques for analysis and enhancement of data for algorithm training.

Regarding this dissertation, it described all steps taken during the development of a coupled system capable of (i) classify the driver workload level and (ii) propose actions to counteract abnormal workload levels, achieved through the implementation of a Machine Learning algorithm and using concepts of Reinforcement Learning and Artificial Intelligence in general. The classification algorithm that classifies driver workload, denominated as Workload Assessor, is based on a Convolutional Neural Network properly trained in combination with a system (*i.e.*, Model Server) that allows it to be horizontally scalable, allowing several other

systems to require the classification of their data. This characteristic is enhanced due to the implemented batching mechanism that makes possible the simultaneous querying of different clients. The workload management (*i.e.*, Workload Manager) is taken by an algorithm inspired in Reinforcement Learning and implemented using Artificial Intelligence (*i.e.*, Prolog), where actions are proposed to an actuator (*i.e.*, HMI system) and rewards given based on the effectiveness in normalizing abnormal workload levels.

In what concerns to the goals proposed in the beginning of this dissertation, the author considers that these were successfully fulfilled. The design and development of the Workload Assessor system proved to be a difficult and time-consuming task. This task was composed by the model implementation which encompassed the application of the previously assimilated Machine Learning concepts and the selection of the best approach for each layer. Moreover, the other goal's part embraced the model training and training data processing. This last part proved to be a time-consuming stage due to the extensive research that was necessary in order to improve the algorithm's performance as well as the meaning and relevance of the previously collected data for the training step. In the other hand, regarding the design and development of the Workload Manager, the design of the management mechanism turned out to be challenging due to the lack of information in literature regarding systems implemented with the same purpose. The inspiration in a Machine Learning field (*i.e.*, Reinforcement Learning) enabled the continuation of the application of concepts related to this domain and gave the base structure to the action selector mechanism as well as the reward provider component. In order to continue in the Artificial Intelligence domain, Prolog was used in such a way that it was possible to represent the actions to be proposed and their relation to the other context variables. In this way, the creation of a knowledge base that enables an iterative updating or addition of actions or relation between these and the context variables, was feasible. Therefore, this second goal was accomplished with success.

Regarding the experimental results, the values obtained in relation to the Workload Assessor, rather to the algorithm of Machine Learning, proved to be very positive and promising. This is verified by the fact that it presents an accuracy percentage of around 90%, and sensitivity and recall values above 80%. One of the crucial factors for the model to reach a very high level of precision, besides having a direct influence of the algorithm, was the application of the oversampling technique. This made it possible for samples of the minority class represented in the training database (*i.e.*, high workload samples) to be added and, therefore,

build a balanced database. In this way, the model was allowed to better distinguish between the samples of the various classes and thus increase the aforementioned metrics values. Comparing the results of the algorithms presented in the literature review section (Chapter 2, Section 2.1.1.4), it is believed to present a similar or even superior performance, in some cases. The author believes that with better quality samples and with more resources, it is possible to present even better results. Moreover, concerning to the most significant features for the workload level classification, the group of 6 features achieved the highest accuracy score. However, it was proved that with only 2 top-ranked significant features in the training database the model achieved interesting results in terms of accuracy (*i.e.*, accuracy of 78%) and with 4 top-ranked features included it achieved good results in general (*i.e.*, accuracy of 83%, approximately, and precision and recall above 70%). Furthermore, the most significant group of variables found in the training dataset and that were not wrongly influenced by the scenario, refer to the physiological data (*i.e.*, heart rate, PERCLOS, blink rate and respiration rate). This result is in agreement with what the literature considers to be the type of measure that best represent the workload state changes in a human driver [7, 10, 11, 12, 13, 59, 109].

Concerning to the Workload Manager, all experimental results were accomplished successfully as expected. The communication between the Workload Assessor was established with success, allowing an effective message exchange between systems. Moreover, as already mentioned in this chapter, this system presents a high capability of adaptation in what concerns to the set of actions available in the knowledge base (*i.e.*, new actions can be added or old ones deleted) and also regarding the addition of different policies for the action selection and the reward attribution.

As a way of evaluating the experiment, the presented results were based on the reliability level of the answers with the aim to obtain a global scale. The results obtained, both in the DALI questionnaire related to the High and Low traffic scenarios, proved to be good. However, as explained when presenting the results, answers considered were related to experiment course sections that were used to train the Machine Learning algorithm. Thus, there were answers regarding the remaining course sections that were neither presented nor considered for analysis. Allied to the questionnaire analysis presented in chapter of results (Chapter 5), the dissertation's author empirically observed that the answers to the remaining questionnaires were dubious. This is intended to be true because the scores attributed by subjects to each DALI factor are, most of the time, very close or superior in the Low than the High Workload experiment. Such a conjugation is considered not

normal due to the much less complex environment and context when comparing the Low traffic versus the High traffic scenarios. In this way, a possible conclusion may be that subjects did not have the clearest perception of what each factor means when answering the questionnaire.

In conclusion, the author considers that the system and the architecture developed form a viable solution that corresponds in a reliable way to classify and manage the driver workload and with considerably good and promising results.

## 6.2 Future Work

Although the goals of this dissertation have been achieved, the author considers that after acquiring a thorough knowledge about the implemented system, and also some knowledge regarding the Machine Learning domain, there are indeed some points where it can be improved and extended with other functionalities.

The first suggestion is to construct a new database for training the algorithm. This new iteration would be composed of data collected in new scenarios conducted during the experiment, where the abnormal states of workload would be provoked in several situations during the trajectories. Therefore, it would be possible to obtain more data on changes in conductor workload and in more comprehensive situations, allowing a greater generalization of the workload states. Moreover, a more balanced database would be achieved.

The second suggestion relates to the fact that, for example, velocity and environment features wrongly influences when the model is training. A possible solution could be to embed the information that these features present in other features, through the calculation of addition facts or as a discount in the latter. Therefore, it would be possible to continue to include the information that these features hold concerning the driving scenario.

A third suggestion would be to deploy the model server to a Cloud environment. Although the current implementation does not require a lot of processing power from the host machine, it would be a way to increase the reach with potential clients and to free up a development board from this task. Also, the training process of new trained versions of the model would not need to be in contact with the final environment (*i.e.*, in this case, the car). This approach was somehow already explored by the dissertation's author but it considers that there is still margin of exploration.

The forth suggestion would be to restructure the model implementation us-

ing a higher abstraction layer that TensorFlow provides. Instead of building the algorithm layer by layer, it can be built by using a single method that highly abstracts the implementation but could somehow introduce some optimization that were not the focus of this dissertation. Then, a comparison in terms of memory used or time of compilation and other aspects could be performed.

Finally, the last suggestion is the integration of the system developed in this dissertation in the P689 project final system. This step would be important to analyze the communication effectiveness between all components that compose the P689 project architecture (Figure 3.1). After this integration has been performed, a validation phase would be applied to a specific scenario, where it would be possible to verify if the Driver Workload system had an acceptable performance and fulfilled the requirements that the scenario imposed.

# Glossary

**Boosting** Implements an iterative learning process. In other words, after a decision tree's training upon a training set, a resubstitution test is done in order to find out what are the mis-categorized samples. After that, a second round of training is done by increasing the percentage of exposure of the mis-categorized sample. Each step of retraining is called boosting [110]. 28

**Dynamic Driving Task** This task ncludes the operational (*i.e.*, steering, braking, accelerating, monitoring the vehicle and roadway) and tactical (*i.e.*, responding to events, determining when to change lanes, turn, use signals, etc.) aspects of the driving task, but not the strategic (determining destinations and waypoints) aspect of the driving task [25]. 75

**Electrocardiogram** Records the heart electrical activity through small electrode patches that attached to chest, arms, and legs skin [111]. 20

**Electrodermal activity** Is the property of the human body that causes continuous variation in the electrical characteristics of the skin, also been known as skin conductance, galvanic skin response (GSR), electrodermal response (EDR), psychogalvanic reflex (PGR), skin conductance response (SCR), sympathetic skin response (SSR) and skin conductance level (SCL). Nowadays, it is standardized to electrodermal activity (EDA) [112]. 20, 24

**Electroencephalogram** Measures and records the brain electrical activity through special sensors called electrodes attached to subject's head [113]. xiii, 20, 24

**Electromyogram** Measures the muscles electrical activity when they are at rest and when they are being used [114]. 24

**Electrooculogram** Is an elecrophysiologic test that measures the existing electrical potential between the cornea and Bruch's membrane [115]. xiii, 22, 24

**Forced or Breakdown flow** Every vehicle moves in lockstep with the vehicle in front of it, with frequent slowing required. Travel time cannot be predicted, with generally more demand than capacity. 89

**Free flow** Traffic flows at or above the posted speed limit and motorists have complete mobility between lanes. The average spacing between vehicles is about 167 m or 27 car lengths. 89

**Multiple-layer perceptron neural network** A class of feed-forward artificial neural network. An MLP consists of at least three layers of nodes, with each node using a nonlinear activation function (except for the input nodes). MLP are known as "vanilla" neural networks [116, 117]. 34, 41

**Session** A Session object encapsulates the environment in which operation objects are executed, and tensor objects are evaluated. It posesses the model graph launched in a session [22]. 109, 111

**Softmax** It is an activation function commonly used in classification problems. The exponential is a steeply increasing function, and therefore the differences between the elements of the vector will increase, rapidly producing large values. Then, as the vector is being normalized, the larger element, which dominates the norm, will be normalized to a value close to 1, while all the other elements will end up divided by a large value and its normalized value will be close to zero. The resulting normalized vector clearly shows what was its greatest element, the "maximum", but maintains the original relative order of its values, hence the "soft" [118]. 104

**Stable flow** Ability to maneuver through lanes is noticeably restricted and lane changes require more driver awareness. Minimum vehicle spacing is about 67 m or 11 car lengths. 89

**Support Vector Machines** Supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier [116, 119]. 30

**Trukey-Kramer test** Single-step multiple comparison procedure and statistical test that can be used on raw data to find factors that can be significantly

different from each other [120]. 44

# Appendix

# Appendix A

# Action rules file

**Listing A.1:** ServerCore module definition.

```
% States that characterize driver
wlLevel(overload).
wlLevel(underload).

% Possible triggers
trigger(weather).
trigger(traffic).
trigger(physiologic).
trigger(ndrtasks).
trigger(drowsiness).

% Cause tuple that represents wich principal
% trigger caused the wlLevel
cause(weather, overload).
cause(traffic, overload).
cause(physiologic, overload).
cause(ndrtasks, overload).
cause(drowsiness, underload).

% Driving automation level
automationLevel(zeroOne).
automationLevel(twoThree).
automationLevel(fourFive).

% Actions to apply
action(prioHMIinfo, weather, zeroOne).
action(prioHMIinfo, traffic, zeroOne).
action(prioHMIinfo, ndrtasks, zeroOne).
```

```
29 action(switchForAS, weather, zeroOne).
30 action(switchForAS, traffic, zeroOne).
31 action(switchForAS, physiologic, zeroOne).
32 action(limitUseNDR, weather, zeroOne).
33 action(limitUseNDR, traffic, zeroOne).
34 action(limitUseNDR, ndrtasks, zeroOne).
35 action(aromatic, physiologic, zeroOne).
36 action(aromatic, drowsiness, zeroOne).
37 action(turnONmusic, physiologic, zeroOne).
38 action(turnONmusic, drowsiness, zeroOne).
39 action(prioHMItasks, traffic, zeroOne).
40 action(prioHMItasks, ndrtasks, zeroOne).
41 action(visualWarn, drowsiness, fourFive).
42 action(visualAuditoryWarn, weather, zeroOne).
43 action(visualAuditoryWarn, traffic, zeroOne).
44 action(visualAuditoryWarn, physiologic, zeroOne).
45 action(visualAuditoryWarn, ndrtasks, zeroOne).
46 action(visualAuditoryWarn, drowsiness, zeroOne).
47 action(visualAuditoryWarn, drowsiness, twoThree).
48 action(turnONlights, drowsiness, zeroOne).
49 action(turnONlights, drowsiness, twoThree).
50 action(switchBackAL, drowsiness, twoThree).
51 action(switchBackAL, drowsiness, fourFive).
52
53 % Simple Validation
54 simple_validation(X,Y,Z) :- wlLevel(X), trigger(Y),
       automationLevel(Z).
55
56 % Cause validation
57 cause_validation(X,Y) :- cause(X,Y).
58
59 % Get action based on wl level, automation level and trigger
60 get_action(X,Y,Z) :- action(X,Y,Z).
```

# Appendix B

# DALI subjective measure

**Scenario:** <scenario>
**Road Sec:** Roundabout

| Title | Value(1-10) | Description |
|---|---|---|
| Effort of attention | | evaluate the attention required by the activity – to think about, to decide, to choose, to look for and so on |
| Visual demand | | evaluate the visual demand necessary for the activity |
| Auditory demand | | evaluate the auditory demand necessary for the activity |
| Temporal demand | | evaluate the specific constraint owing to timing demand when running the activity |
| Interference | | evaluate the possible disturbance when running the driving activity simultaneously with any other supplementary task such as phoning, using systems or radio and so on |
| Situational stress | | evaluate the level of constraints/stress while conducting the activity such as fatigue, insecure feeling, irritation, discouragement and so on |
| | | |
| | **Value(1-3)** | |
| **Overall workload** | | |

**Road Sec:** Straight

| Title | Value(1-10) | Description |
|---|---|---|
| Effort of attention | | evaluate the attention required by the activity – to think about, to decide, to choose, to look for and so on |
| Visual demand | | evaluate the visual demand necessary for the activity |
| Auditory demand | | evaluate the auditory demand necessary for the activity |
| Temporal demand | | evaluate the specific constraint owing to timing demand when running the activity |
| Interference | | evaluate the possible disturbance when running the driving activity simultaneously with any other supplementary task such as phoning, using systems or radio and so on |
| Situational stress | | evaluate the level of constraints/stress while conducting the activity such as fatigue, insecure feeling, irritation, discouragement and so on |
| | | |
| | Value(1-3) | |
| **Overall workload** | | |

**Road Sec:** Interception/Crossroads

| Title | Value(1-10) | Description |
|---|---|---|
| Effort of attention | | evaluate the attention required by the activity – to think about, to decide, to choose, to look for and so on |
| Visual demand | | evaluate the visual demand necessary for the activity |
| Auditory demand | | evaluate the auditory demand necessary for the activity |
| Temporal demand | | evaluate the specific constraint owing to timing demand when running the activity |
| Interference | | evaluate the possible disturbance when running the driving activity simultaneously with any other supplementary task such as phoning, using systems or radio and so on |
| Situational stress | | evaluate the level of constraints/stress while conducting the activity such as fatigue, insecure feeling, irritation, discouragement and so on |
| | | |
| | Value(1-3) | |
| **Overall workload** | | |

**Figure B.1:** DALI questionnaire presented to subjects throughout the city experiment.

# Appendix C

# Karolinska sleepiness scale (KSS)


**Figure C.1:** KSS questionnaire presented to subjects though an Android application, throughout the highway experiment.

# Bibliography


[1] A. Heigemeyr and A. Harrer, "Information Management for Adaptive Automotive Human Machine Interfaces," in *Proceedings of the 6th International Conference on Automotive User Interfaces and Interactive Vehicular Applications - AutomotiveUI '14*. New York, New York, USA: ACM Press, 2014, pp. 1–8. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2667317.2667341

[2] A. Amditis, L. Andreone, K. Pagle, G. Markkula, E. Deregibus, M. Romera Rue, F. Bellotti, A. Engelsberg, R. Brouwer, B. Peters, and A. De Gloria, "Towards the automotive HMI of the future: Overview of the AIDE - Integrated project results," *IEEE Transactions on Intelligent Transportation Systems*, vol. 11, no. 3, pp. 567–578, 2010.

[3] A. Heigemeyr and A. Harrer, "An integrated method for Adaptive automotive Human Machine Interfaces," in *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*. IEEE, oct 2013, pp. 558–564. [Online]. Available: http://ieeexplore.ieee.org/document/6728290/

[4] N. H. T. S. Administration and U. D. of Transportation, *Research Note Distracted Driving 2014*, 2015, vol. 2014, no. April 2016.

[5] V. L. Neale, T. A. Dingus, S. G. Klauer, and M. Goodman, "An overview of the 100-car naturalistic study and findings," *Traffic Safety*, vol. 19, pp. 1–10, 2005. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.172.2366{&}rep=rep1{&}type=pdf

[6] D. Meister, "Behavioral foundations of system development." 1976. [Online]. Available: http://psycnet.apa.org/psycinfo/1976-28894-000

[7] D. de Waard and V. Studiecentrum, *The Measurement of Drivers ' Mental Workload*, 1996, vol. 39, no. 4. [Online].

Available: http://dissertations.ub.rug.nl/FILES/faculties/ppsw/1996/d.de.waard/09{%}7B{_}{%}7Dthesis.pdf

[8] R. Srinivasan, N. Ganti, M. Krishnan, G. Krishnan, P. Ram, and D. Marur, "Remote Heartbeat Monitor: Analog Devices Wiki." [Online]. Available: https://wiki.analog.com/university/contest/design/submissions/the{_}sentinels

[9] National Institutes of Health, "Sleep: Information about Sleep." [Online]. Available: https://science.education.nih.gov/supplements/webversions/SleepDisorders/guide/info-sleep.html

[10] Y. Z. Y. Zhang, Y. Owechko, and J. Z. J. Zhang, "Driver cognitive workload estimation: a data-driven perspective," *Proceedings. The 7th International IEEE Conference on Intelligent Transportation Systems (IEEE Cat. No.04TH8749)*, pp. 642–647, 2004.

[11] J. Son, H. Oh, and M. Park, "Identification of driver cognitive workload using support vector machines with driving performance, physiology and eye movement in a driving simulator," *International Journal of Precision Engineering and Manufacturing*, vol. 14, no. 8, pp. 1321–1327, 2013.

[12] P. Besson, E. Dousset, C. Bourdin, L. Bringoux, T. Marqueste, D. R. Mestre, and J. L. Vercher, "Bayesian network classifiers inferring workload from physiological features: Compared performance," *IEEE Intelligent Vehicles Symposium, Proceedings*, pp. 282–287, 2012.

[13] E. T. Solovey, M. Zec, E. A. Garcia Perez, B. Reimer, and B. Mehler, "Classifying driver workload using physiological and driving performance data," *Proceedings of the 32nd annual ACM conference on Human factors in computing systems - CHI '14*, pp. 4057–4066, 2014. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-84900446066{&}partnerID=tZOtx3y1

[14] J. H. Yang and S. Lim, "Driver state estimation by convolutional neural network using multimodal sensor data," *Electronics Letters*, vol. 52, no. 17, pp. 1495–1497, 2016. [Online]. Available: http://digital-library.theiet.org/content/journals/10.1049/el.2016.1393

[15] RedMonk, "A Look at Popular Machine Learning Frameworks – Charting Stacks," 2016. [Online]. Available: http://redmonk.com/fryan/2016/06/06/a-look-at-popular-machine-learning-frameworks/

[16] S. M. Patterson, "Google open source TensorFlow 1.0 debuts – vies for platform status," 2017. [Online]. Available: https://www.networkworld.com/article/3171272/software/google-opens-source-tensorflow-1-0-debuts-vies-for-platform-status.html

[17] Google, "TensorFlow," 2017. [Online]. Available: https://www.tensorflow.org/

[18] D. Remboski, J. Gardner, D. Wheatley, J. Hurwitz, T. MacTavish, and Robert, "Driver Performance Improvement Through the Driver Advocate: a Research Initiative Toward Automotive Safety," 2000.

[19] M. R. Smith, G. J. Witt, and D. L. Bakowski, "Results of the US SAVE-IT project," Delphi, Tech. Rep., 2008.

[20] A. Amditis, A. Polychronopoulos, L. Andreone, and E. Bekiaris, "Communication and interaction strategies in automotive adaptive interfaces," *Cognition, Technology & Work*, vol. 8, no. 3, pp. 193–199, sep 2006. [Online]. Available: http://link.springer.com/10.1007/s10111-006-0033-0

[21] Centro de Computação Gráfica (CCG), "HMIEXCEL - soluções multimédia para indústria automóvel - CCG." [Online]. Available: http://www.ccg.pt/projetos/hmiexcel-bosch/

[22] Martin Abadi and Others, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," 2015. [Online]. Available: http://tensorflow.org/

[23] H. P. Martínez and G. N. Yannakakis, "Deep Multimodal Fusion: Combining Discrete Events and Continuous Signals." [Online]. Available: http://dx.doi.org/10.1145/2663204.2663236.

[24] M. Rubashkin, "Getting Started with Deep Learning," 2017. [Online]. Available: http://www.kdnuggets.com/2017/03/getting-started-deep-learning.html

[25] SAE, "Summary of SAE international's levels of Driving Automation for On-Road vehicles," 2014. [Online]. Available: https://www.sae.org/misc/pdfs/automated{%}7B{_}{%}7Ddriving.pdf

[26] H. Alm and L. Nilsson, "The effects of a mobile telephone task on driver behaviour in a car following situation," *Accid. Anal. and Prev*, vol. 27, no. 5, pp. 707–715, 1995.

[27] M. H. Kim and J. Son, "On-road assessment of in-vehicle driving workload for older drivers: Design guidelines for intelligent vehicles," *International Journal of Automotive Technology*, vol. 12, no. 2, pp. 265–272, apr 2011. [Online]. Available: http://link.springer.com/10.1007/s12239-011-0031-y

[28] V. Trent, "Keeping eye and mind on the road." Ph.D. dissertation, Uppsala, 2005.

[29] K. L. Y. michael A Regan, Johan D Lee, *Driver Distraction: Theory, effect and mitigation*, 2015, vol. 1.

[30] P. By, T. Brown, D. Marshall, J. Moeckli, and T. Smyser, "A Final Report of SAfety Vehicle(s) using adaptive Interface Technology (SAVE-IT) Program: Task 14a Evaluation," University of Iowa, Tech. Rep., 2007.

[31] F. Bellotti, A. D. Gloria, R. Montanari, N. Dosio, and D. Morreale, "COMUNICAR: designing a multimedia, context-aware human-machine interface for cars," *Cognition, Technology & Work*, vol. 7, no. 1, pp. 36–45, mar 2005. [Online]. Available: http://link.springer.com/10.1007/s10111-004-0168-9

[32] S. Damiani, E. Deregibus, and L. Andreone, "Driver-vehicle interfaces and interaction: where are they going?" *European Transport Research Review*, vol. 1, no. 2, pp. 87–96, jul 2009. [Online]. Available: http://link.springer.com/10.1007/s12544-009-0009-2

[33] T. Fleischmann, "Model based HMI specification in an automotive context," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 4557 LNCS, no. PART 1, pp. 31–39, 2007. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-38149052898{&}partnerID=tZOtx3y1

[34] U. d. M. Bosch, "Project Charter: P689 Cockpit of the Future," p. 22, 2016.

[35] W. Rouse, S. Edwards, and J. Hammer, "Modeling the dynamics of mental workload and human performance in complex systems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 23, no. 6, pp. 1662–1671, 1993. [Online]. Available: http://ieeexplore.ieee.org/document/257761/

[36] K. D. d. W. Roskam, A.J.; Brookhuis and Others, "HASTE Deliverable 1: Development of Experimental Protocol A.J." *Human Machine Interface and the Safety of Traffic in Europe Growth Project. HASTE.*, 2002.

[37] K. A. Brookhuis and D. de Waard, "Monitoring drivers' mental workload in driving simulators using physiological measures," *Accident Analysis and Prevention*, vol. 42, no. 3, pp. 898–903, 2010. [Online]. Available: http://dx.doi.org/10.1016/j.aap.2009.06.001

[38] T. Nilsson, T. M. Nelson, and D. Carlson, "Development of fatigue symptoms during simulated driving," *Accident Analysis and Prevention*, vol. 29, no. 4 SPEC. ISS., pp. 479–488, 1997.

[39] J. A. Michon, "A Critical View oF Driver Behavior Models: What do we know, what should we do?" *Human behavior & traffic safety*, pp. 485–524, 1985. [Online]. Available: http://jamichon.nl/jam{_}writings/1985{_}criticial{_}view.pdf

[40] F. P. da Silva, "Mental Workload, Task Demand and Driving Performance: What Relation?" *Procedia - Social and Behavioral Sciences*, vol. 162, no. Panam, pp. 310–319, 2014.

[41] R. D. O'Donnell and F. T. Eggemeier, *Workload assessment methodology*, L. K. &. J. P. T. K. R. Boff, Ed. New York: Wiley, 1986. [Online]. Available: http://apps.usd.edu/coglab/schieber/docs/odonnell.pdf

[42] S. G. Hart, "NASA Task Load Index," NASA Ames Research Center, California, Tech. Rep., 1986.

[43] S. G. Hart and L. E. Sta, "Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research," N. Hancock, P.A.; Meshkati, Ed. San Jose, California: Elsevier Science Publishers, 1988, p. 45.

[44] R. J. Jansen, B. D. Sawyer, R. van Egmond, H. de Ridder, and P. A. Hancock, "Hysteresis in Mental Workload and Task Performance: The Influence of Demand Transitions and Task Prioritization," *Human Factors: The Journal of the Human Factors and Ergonomics Society*, vol. 58, no. 8, pp. 1143–1157, dec 2016. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/27613827http://hfs.sagepub.com/cgi/doi/10.1177/0018720816669271

[45] S. Miller, "Workload measures," *The University of Iowa*, no. August, pp. 1–65, 2001.

[46] W. W. Wierwille and J. G. Casali, "A validated rating scale for global mental workload measurement applications," in *Proceedings of the Human Factors*

*and Ergonomics Society*, Virginia, 1983, pp. 129–133.

[47] G. E. Cooper and R. P. Harper, "The Use of Pilot Ratings in the Evaluation of Aircraft Handling Qualities," no. AD689722, report 567, pp. 1–56, 1969.

[48] L. Zijlstra, F.R.H. and van Doorn, "The Construction of a Scale to Measure Perceived Effort," Department of Philosophy and Social Sciences, Delft, The Netherlands, Tech. Rep., 1985.

[49] A. Pauzie, "A method to assess the driver mental workload: The driving activity load index (DALI)," *IET Intelligent Transport Systems*, vol. 2, no. 4, p. 315, 2008. [Online]. Available: http://digital-library.theiet.org/content/journals/10.1049/iet-its{_}20080023

[50] D. De Waard, "Mental Workload," in *Human Factors for Highway engineers*, R. Fuller and J. A. Santos, Eds. University of Groningen: Pregamon Press, 2002, ch. 11, pp. 161–175.

[51] M. De Rivecourt, M. N. Kuperus, W. J. Post, and L. J. M. Mulder, "Cardiovascular and eye activity measures as indices for momentary changes in mental effort during simulated flight," *Ergonomics*, vol. 51, no. 9, pp. 1295–1319, 2008. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/00140130802120267

[52] B.-W. Hsu, M.-J. J. Wang, C.-Y. Chen, and F. Chen, "Effective Indices for Monitoring Mental Workload While Performing Multiple Tasks," *Perceptual and Motor Skills*, vol. 121, no. 1, pp. 94–117, 2015. [Online]. Available: http://journals.sagepub.com/doi/10.2466/22.PMS.121c12x5

[53] J.-A. Bachorowski, "Vocal Expression and Perception of Emotion," *Current Directions in Psychological Science*, vol. 8, no. 2, pp. 53–57, 1999. [Online]. Available: http://journals.sagepub.com/doi/10.1111/1467-8721.00013

[54] A. Gevins, H. Leong, R. Du, M. E. Smith, J. Le, D. DuRousseau, J. Zhang, and J. Libove, "Towards measurement of brain function in operational environments." *Biological psychology*, vol. 40, no. 1-2, pp. 169–86, may 1995. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/7647178

[55] K. R. Boff, J. P. Thomas, D. Gopher, and E. Donchin, *Workload - An examination of the concept*, ii ed., K. R. Boff, J. P. Thomas, D. Gopher, and L. Kaufman, Eds. New York: Wiley, 1986.

[Online]. Available: http://usd-apps.usd.edu/coglab/schieber/psyc792/workload/GopherDonchin1986.pdf

[56] M. Myrtek, E. Deutschmann-Janicke, H. Strohmaier, W. Zimmermann, S. Lawerenz, G. Brügner, and W. Müller, "Physical, mental, emotional, and subjective workload components in train drivers," *Ergonomics*, vol. 37, no. 7, pp. 1195–1203, 1994. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/00140139408964897

[57] M. F. Folstein, S. E. Folstein, and P. R. McHugh, "Mini-mental state. A practical method for grading the cognitive state of patients for the clinician." *Journal of psychiatric research*, vol. 12, no. 3, pp. 189–98, nov 1975. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/1202204

[58] M. F. Akay, "Support vector machines combined with feature selection for breast cancer diagnosis," *Expert Systems With Applications*, vol. 36, pp. 3240–3247, 2009. [Online]. Available: https://pdfs.semanticscholar.org/fe83/150bc326fd62d352cb2993ac91344f195e10.pdf

[59] R. Hoogendoorn and B. Van Arem, "Driver workload classification through neural network modeling using physiological indicators," *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, no. Itsc, pp. 2268–2273, 2013.

[60] W. Helly, *Simulation of bottlenecks in single-lane traffic flow.*, 1961. [Online]. Available: http://www.worldcat.org/title/simulation-of-bottlenecks-in-single-lane-traffic-flow/oclc/173358195

[61] S. Hoogendoorn and R. Hoogendoorn, "Generic Calibration Framework for Joint Estimation of Car-Following Models by Using Microscopic Data," *Transportation Research Record: Journal of the Transportation Research Board*, vol. 2188, pp. 37–45, dec 2010. [Online]. Available: http://trrjournalonline.trb.org/doi/10.3141/2188-05

[62] B. Mehler, B. Reimer, and Y. Wang, "A Comparison of Heart Rate and Heart Rate Variability Indices in Distinguishing Single-Task Driving and Driving Under Secondary Cognitive Workload," 2011. [Online]. Available: https://trid.trb.org/view.aspx?id=1112888

[63] J. H. Yang and H. B. Jeong, "Improvement of driver- state estimation algorithm using multi-modal information," in *2015 15th International Conference on Control, Automation and Systems (ICCAS)*. IEEE, 2015,

pp. 2011–2015. [Online]. Available: http://ieeexplore.ieee.org/document/7364698/

[64] Dr Reshma Khemchandani, "Machine Learning - SAU - South Asian University," 2016. [Online]. Available: http://www.sau.int/research-themes/machine-learning.html

[65] G. Grana and J. O. Windell, *Crime and intelligence analysis : an integrated real-time approach*, C. Press, Ed. CRC Press, 2016.

[66] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014. [Online]. Available: http://www.cs.huji.ac.il/

[67] B. Furht and A. Agarwal, Eds., *Handbook of Medical and Healthcare Technologies*. New York, NY: Springer New York, 2013. [Online]. Available: http://link.springer.com/10.1007/978-1-4614-8495-0

[68] C. D. Manning, P. Ragahvan, and H. Schutze, *An Introduction to Information Retrieval*. Cambridge University Press, 2009, no. c.

[69] R. Caruana and A. Niculescu-Mizil, "An Empirical Comparison of Supervised Learning Algorithms," Department of Computer Science, Cornell University, Ithaca, Tech. Rep.

[70] M. N. Murty and V. S. Devi, *Pattern Recognition*, ser. Undergraduate Topics in Computer Science. London: Springer London, 2011, vol. 0. [Online]. Available: http://link.springer.com/10.1007/978-0-85729-495-1

[71] C. Stergiou, "Neural Networks." [Online]. Available: https://www.doc.ic.ac.uk/{~}nd/surprise{_}96/journal/vol1/cs11/article1.html

[72] J. V. Tu, "Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes," *Journal of Clinical Epidemiology*, vol. 49, no. 11, pp. 1225–1231, nov 1996. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0895435696000029

[73] V. Cheung, K. Cannons, and W. Kinsner, "An Introduction to Neural Networks," 2002.

[74] J. Kabuba, A. Mulaba-bafubiandi, and K. Battle, "A Critics Study of Neural Networks Applied to ion-Exchange Process," vol. 6, no. 8, pp. 1226–1229, 2012.

[75] B. Networks, F. Faltin, and R. Kenett, "Bayesian Networks," *Encyclopedia of Statistics in Quality & Reliability*, vol. 1, no. 1, p. 4, 2007.

[76] M. E. Kragt, "A beginners guide to Bayesian network modelling for integrated catchment management," *Landscape Logic*, no. 9, p. 22, 2009.

[77] V. C. Science, "History of Machine Learning," Vtu Computer Science, Tech. Rep.

[78] L.-C. Chen, A. G. Schwing, A. L. Yuille, and R. Urtasun, "Learning deep structured models."

[79] Y. LeCun, Y. Bengio, and G. Hinton, *Deep learning - Review*. Macmillan Publishers Limited, 2015, vol. 521.

[80] G. Mesnil, Y. Dauphin, X. Glorot, S. Rifai, Y. Bengio, I. Goodfellow, E. Lavoie, X. Muller, G. Desjardins, D. Warde-Farley, P. Vincent, A. Courville, and J. Bergstra, "Unsupervised and Transfer Learning Challenge: a Deep Learning Approach," *JMLR: Workshop and Conference Proceedings*, 2012.

[81] Google, "TensorFlow Serving," 2017. [Online]. Available: https://www.tensorflow.org/serving/

[82] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional Architecture for Fast Feature Embedding," *arXiv preprint arXiv:1408.5093*, 2014. [Online]. Available: http://arxiv.org/abs/1408.5093

[83] Caffe2, "Caffe2: A New Lightweight, Modular, and Scalable Deep Learning Framework." [Online]. Available: https://caffe2.ai/

[84] The Theano Development Team, R. Al-Rfou, G. Alain, A. Almahairi, and Others, "Theano: A Python framework for fast computation of mathematical expressions," pp. 1–19, 2016. [Online]. Available: http://arxiv.org/abs/1605.02688

[85] F. Chollet, "Keras," 2015. [Online]. Available: https://github.com/keras-team/keras

[86] S. Dieleman, J. Schlüter, C. Raffel, E. Olson, S. K. Sønderby, D. Nouri, D. Maturana, M. Thoma, E. Battenberg, J. Kelly, J. D. Fauw, M. Heilman, Diogo149, B. McFee, H. Weideman, Takacsg84,

Peterderivaz, Jon, Instagibbs, D. K. Rasul, CongLiu, Britefury, and J. Degrave, "Lasagne: First release." aug 2015. [Online]. Available: https://zenodo.org/record/27878

[87] B. van Merriënboer, D. Bahdanau, V. Dumoulin, D. Serdyuk, D. Warde-Farley, J. Chorowski, and Y. Bengio, "Blocks and Fuel: Frameworks for deep learning," pp. 1–5, 2015. [Online]. Available: http://arxiv.org/abs/1506.00619

[88] F. Seide and A. Agarwal, "CNTK: Microsoft's Open-Source Deep-Learning Toolkit," *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*, pp. 2135–2135, 2016. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2939672.2945397

[89] R. Collobert, S. Bengio, and J. Mariéthoz, "Torch: a modular machine learning software library," p. 7, 2002. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=CBB0C8A5FE34F6D6DAFF997F6B6A205A?doi=10.1.1.8.9850{&}rep=rep1{&}type=pdf

[90] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems," *eprint arXiv:1512.01274*, pp. 1–6, 2015. [Online]. Available: http://arxiv.org/abs/1512.01274

[91] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2078195{%}5Cnhttp://arxiv.org/abs/1201.0490

[92] K. Torkkola, "Intelligent Driving Data Analysis," Motorola Labs, Human Interface Lab, Tech. Rep.

[93] Chrysler Group, "Electronic Driver Advocate Helps Keep Focus on the Road In DaimlerChrysler Technology Concept," 2003. [Online]. Available: http://www.prnewswire.com/news-releases/electronic-driver-advocate-helps-keep-focus-on-the-road-in-daimlerchrysler/technology-concept-vehicle-71293152.html

[94] A. Amditis, L. Andreone, A. Polychronopoulos, and J. Engström, "Design and Development of an Adaptive Integrated Driver-Vehicle Interface: Overview of the AIDE Project," *IEEE Transactions on Intelligent Transportation Systems*, vol. VOL. 11, no. 3, 2010.

[95] G. J. Bowden, H. R. Maier, and G. C. Dandy, "Optimal division of data for neural network models in water resources applications," *Water Resources Research*, vol. 38, no. 2, pp. 2–1–2–11, 2002. [Online]. Available: http://doi.wiley.com/10.1029/2001WR000266

[96] M. Nielsen, "Improving the way neural networks learn," 2017. [Online]. Available: http://neuralnetworksanddeeplearning.com/chap3.html

[97] J. Brownlee, "Adam Optimization Algorithm for Deep Learning - Machine Learning Mastery," 2017. [Online]. Available: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

[98] Google, "gRPC: The open-source universal RPC framework," 2017. [Online]. Available: https://grpc.io/

[99] J. Wielemaker, T. Schrijvers, M. Triska, and T. Lager, "SWI-Prolog," *Theory and Practice of Logic Programming*, vol. 12, no. 1-2, pp. 67–96, 2012.

[100] H.-p. Krueger, M. Grein, A. Kaussner, and C. Mark, "SILAB – A Task Oriented Driving Simulation," *DSC 2005 North America*, pp. 323–331, 2005.

[101] A. Shahid, K. Wilkinson, S. Marcu, and C. M. Shapiro, "STOP, THAT and one hundred other sleep scales," *STOP, THAT and One Hundred Other Sleep Scales*, pp. 1–406, 2012.

[102] J. I. Harbison, S. M. Atkins, and M. R. Dougherty, "N-back Training Task Performance: Analysis and Model Experiment," *Proceedings of the 33rd Annual Conference of the Cognitive Science Society*, pp. 120–125, 2011. [Online]. Available: https://mindmodeling.org/cogsci2011/papers/0026/paper0026.pdf

[103] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.

[104] G. Lemaitre, F. Nogueira, and C. K. Aridas, "Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning,"

*Journal of Machine Learning Research*, vol. 18, no. 17, pp. 1–5, 2017. [Online]. Available: http://arxiv.org/abs/1609.06570

[105] R. L. Wasserstein and N. A. Lazar, "The ASA's Statement on p -Values: Context, Process, and Purpose," *The American Statistician*, vol. 70, no. 2, pp. 129–133, 2016. [Online]. Available: https://www.tandfonline.com/doi/full/10.1080/00031305.2016.1154108

[106] A. Joshi, S. Kale, S. Chandel, and D. Pal, "Likert Scale: Explored and Explained," *British Journal of Applied Science & Technology*, vol. 7, no. 4, pp. 396–403, 2015. [Online]. Available: http://www.sciencedomain.org/abstract.php?iid=773{&}id=5{&}aid=8206

[107] C. Zaiontz, "Cronbach's Alpha: Real Statistics Using Excel," 2013. [Online]. Available: http://www.real-statistics.com/reliability/cronbachs-alpha/

[108] NASA, *NASA TLX.pdf*, 1st ed. Clifornia: NASA Ames Research Center, 1986.

[109] G. F. Wilson and C. A. Russell, "Real-Time Assessment of Mental Workload Using Psychophysiological Measures and Artificial Neural Networks," *Human Factors: The Journal of the Human Factors and Ergonomics Society*, vol. 45, no. 4, pp. 635–644, dec 2003. [Online]. Available: http://journals.sagepub.com/doi/10.1518/hfes.45.4.635.27088

[110] S. Ray, "Quick Guide to Boosting Algorithms in Machine Learning," 2015. [Online]. Available: https://www.analyticsvidhya.com/blog/2015/11/quick-introduction-boosting-algorithms-machine-learning/

[111] American Heart Association, "Electrocardiogram," 2015. [Online]. Available: http://www.heart.org/HEARTORG/Conditions/HeartAttack/DiagnosingaHeartAttack/Electrocardiogram-ECG-or-EKG{_}UCM{_}309050{_}Article.jsp

[112] J. Braithwaite, D. Watson, J. Robert, and R. Mickey, "A Guide for Analysing Electrodermal Activity (EDA) & Skin Conductance Responses (SCRs) for Psychological Experiments," pp. 1–42, 2013. [Online]. Available: http://www.bhamlive.bham.ac.uk/Documents/college-les/psych/saal/guide-electrodermal-activity.pdf{%}5Cnhttp://www.birmingham.ac.uk/documents/college-les/psych/saal/guide-electrodermal-activity.pdf

[113] E. K. St. Louis and L. C. Frey, *Electroencephalography*, 2016. [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK390354/{%}0Ahttps://www.osapublishing.org/abstract.cfm?URI=boe-7-7-2524

[114] S. Kishner and Medscape, "Electromyography and Nerve Conduction Studies: Background, Indications, Contraindications," 2015. [Online]. Available: https://emedicine.medscape.com/article/2094544-overview

[115] U. Siddiqui and A. N. Shaikh, "An Overview of " Electrooculography "," *Advanced Research in Computer and Communication Engineering*, vol. 2, no. 11, pp. 4328–4330, 2013.

[116] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*, 1991. [Online]. Available: http://link.springer.com/chapter/10.1007/978-94-011-3532-0{_}2

[117] J. Brownlee, "Multi-Layer Perceptron Neural Networks," 2016. [Online]. Available: https://machinelearningmastery.com/neural-networks-crash-course/

[118] Stanford University, "Unsupervised Feature Learning and Deep Learning: Softmax Regression," 2018. [Online]. Available: http://ufldl.stanford.edu/tutorial/supervised/SoftmaxRegression/

[119] Scikit-Learn, "Support Vector Machines," 2017. [Online]. Available: http://scikit-learn.org/stable/modules/svm.html

[120] National Institute of Standards and Technology, *Engineering Statistics: Handbook*, 2006.