POLITECNICO
MILANO 1863

POLITECNICO DI MILANO
Master of Computer Science and Engineering
Department of Electronics, Information and Bioengineering

I.DRIVE: A software system for driver, vehicle, and environment interaction data analytics

AIRLab
Artificial Intelligence & Robotics Lab
Politecnico di Milano

Advisor: Prof. Matteo Matteucci
Co-advisor: Alessandro Gabrielli

Master Thesis:
Claudio Sesto, matricola 841623

Academic Year 2018-2019

A Martina

Abstract

With the progression of technology, automatic functions implementations in cars are becoming more common, reducing the human effort for basic actions or manoeuvres getting closer to the realization of a fully autonomous car; however, there are still a lot of issues to overcome before a fully autonomous car could be used in practice without harming passengers and pedestrians. Although many studies describe possible implementations of autonomous car, not many focuses on mood of passenger during the drive.

The goal of this work is to develop a software system able to process collected data during driving tasks and highlight possible stressing events occurred such that future autonomous cars know in advance possible stressing situations and behave accordingly, thus preventing passenger discomfort. To reach this goal we used a properly modified car on which have been installed cameras, Global Positioning System (GPS) and Light Detection and Ranging (LiDAR) sensors driven in a urban environment and measuring sweating and heart beat rate of the driver: the driver also wears an eye tracker in order to contextualise data collected by the other sensors through information about the observed objects during driving. Once data have been collected, we developed a software system that processes data and plots results on graphs to show factors that could be related to a stressing event.

Sommario

Il progetto Interaction between Driver, Road Infrastructure, Vehicle and Environment (I.DRIVE) è un progetto interdisciplinare del Politecnico di Milano il cui scopo è acquisire, analizzare e modellare l'interazione tra il guidatore, il veicolo e l'ambiente circostante. All'interno di questo progetto si è svolto il lavoro descritto in questa tesi. Infatti, abbiamo progettato e sviluppato un sistema software in grado di elaborare i dati raccolti dai sensori montati su un'automobile al fine di ricavare le differenze nei segnali fisiologici durante eventi di stress; in questo modo siamo in grado di distinguere e rilevare situazioni potenzialmente stressanti che possono verificarsi durante la guida tradizionale o autonoma.

Questo perché in un veicolo completamente autonomo, il controllo è affidato esclusivamente al veicolo stesso e non più al guidatore: questa perdita del controllo sul veicolo, come mostrato in seguito, può causare insicurezza e stress. Per incrementare la confidenza che un passeggero ha verso un'auto autonoma, è importante comprendere le reazioni emotive di quest'ultimo durante la marcia in modo da adattare il comportamento dell'auto autonoma così che vengano evitate situazioni che possano generare stress nel passeggero. Lo scopo di questo lavoro di tesi è quello di sviluppare un algoritmo che estrae automaticamente gli indici di stress, evidenziando gli istanti in cui è possibile si sia verificato un evento stressante per il passeggero; unendo questi dati con le informazioni acquisite sull'ambiente circostante è possibile dare un contesto allo stress rilevato.

Per raggiungere il nostro obiettivo, è necessario raccogliere più informazioni possibili sul guidatore e sull' ambiente esterno: per far ciò abbiamo utilizzato dei sensori applicati sul guidatore per monitorare alcuni segnali fisiologici, e sensori installati sull'automobile, tra cui telecamere, per acquisire informazioni sull'ambiente circostante. Per contestualizzare meglio i segnali di stress, abbiamo poi elaborato le immagini tramite Deep Learning.

Una volta che i dati sono stati raccolti, salviamo quelli più rilevanti in file rosbag, che sono particolari file accessibili sequenzialmente: questi file sono

stati utilizzati per disegnare grafici e mostrare possibili misure di stress. Durante lo sviluppo di questo progetto, abbiamo avuto modo di verificare i risultati di altri paper che mostravano le diverse affidabilità di diversi segnali fisiologici utilizzati come misure di stress.

Acknowledgments

First of all I would like to thanks Martina, which supported me a lot in this last period of my university career even in the worst moments and helping me to overcome them, and her parents that were patients as her in this period. Thanks a lot also to all my collegues Salvatore, Erica, Andrei, Gioele, Paolo, Luca and all the others which made these years unforgettable making great this period of my life. Thanks to my friends Moreno, Gabriele, Camilla and Martina with which I spent the funniest moments of my life and Davide and Federico with which I spent all my summer holidays making me understand the importance of friendship.

Thanks to all people of the AIRLab, especially to Ewerton with which I had lot of positive dialogues, helped my really a lot with the drafting of this thesis and had a lot of fun even by knowing for few months. Thanks also to Alessandro that followed me in this works and gave an important contribution to it and professor Matteucci that allowed me to work in this very interesting, stimulating and innovative project. Last but not least I would like to thank my parent for this opportunity of my life.

Contents

List of Figures

Glossary

AI Artificial Intelligence. 7, 19

BVP Blood Volume Pulse. 27, 67

CNN Convolutional Neural Network. 20, 21, 42, 49, 88

CPU Central Processing Unit. 32

DARPA Defense Advanced Research Projects Agency. XIII, 5, 6

DGPS Differential Global Positioning System. 5

ECG Electrocardiogram. 15, 29, 30, 35, 37, 42, 45, 47, 65

EMG Electromyogram. 15

GPS Global Positioning System. I, XIII, 7, 8, 25, 26, 30, 39, 76

GPU Graphics Processing Unit. 31, 32, 88

GSR Galvanic Skin Response. 13, 15, 16, 29, 30, 36, 37, 39, 42–47, 50, 51, 57, 59, 60, 65, 67, 85–87, 90, 91, 93

HBR Heart Beat Rate. 13, 14, 29, 30

HDD Hard Disk Drive. 31

HF High Frequencies. 13–16, 45, 46, 51, 57, 59, 65, 87, 93

HRV Heart Rate Variability. 13–16, 30, 57

I.DRIVE Interaction between Driver, Road Infrastructure, Vehicle and Environment. III, 1–3, 8

IMU Inertial Measurement Unit. XIII, 25

ISCR Integrated Skin Conductance Response. 45

LAN Local Area Network. 39

LF Low Frequencies. 13–16, 45, 46, 51, 57, 59, 65, 87, 93

LiDAR Light Detection and Ranging. I, XIII, 7, 23–26, 30, 31, 66, 67

LSE Least Squared Error. 20

MF Mid Frequencies. 16

ML Machine Learning. 3, 19

NHTSA National Highway Traffic Safety Administration. 4

NLMSE Normalized Least Mean Squared Error. 46, 87

NN Neural Network. 2, 13, 19, 20, 31, 32, 66

NTP Network Time Protocol. 39

PCC Pearson Correlation Coefficient. 45, 46, 50, 86, 87

RAPT-3 Risk Awareness and Perception Training. 18

RLSE Recoursive Least Squared Error. 46, 57, 65, 87

ROI Region Of Interest. 18, 85, 86

ROS Robot Operating System. X, 32, 44, 49, 51, 75, 76, 89

SC Skin Conductance. 15

SLAM Simultaneous Localisation and Mapping. 7

SSD Solid State Drive. 31

SVM Support Vector Machine. 20

UI User Interface. 8, 38, 47, 48

YOLOv3 You Only Look Once. X, XIII, XIV, 20, 21, 31, 32, 42–44, 49, 65–68, 75–77, 80, 83, 84, 87, 88

Chapter 1

Introduction

The I.DRIVE project is an interdisciplinary Politecnico di Milano project which goal is to acquire, analyse, and model the interaction between driver, vehicle and environment. The thesis work described in this paper was carried out as part of the project. Indeed, we designed and developed a software system in order to process data collected from the sensor system of a car to spot differences in physiological signals trend during stressing events; the resulting information shows us the physiological differences whenever a stressing event occurs. In this way we are able to distinguish and detect potential stressing situations that can occur while being passenger in an autonomous car.

This because in a fully autonomous car, the disengagement of the passenger implies loss of control on the car and this, as we show later in this work, causes insecurity and stress. So, in order to make people trust on autonomous car, we must understand how does the passenger feels during driving tasks and adapt the car's behaviour: in this way, knowing in advance possible stressing situations, is possible to avoid them making the trip more comfortable.

The main goal of this work is to develop an algorithm that automatically extracts stress indexes by process collected data, highlighting also stress events and contextualise this information in order to understand which could be the possible stressing situations. To achieve this goal, we need to retrieve as much information as possible about the driver to understand his/her mood, while he/she drives in a urban context and the external environment, to get possible uncomfortable situations or stressing events: to do this we used some sensors attached to the driver to monitor some physiological signals, and other sensors, among which cameras, to get information about the surrounding environment. We also used an eye tracker to see where the

driver was focused on, so to contextualise better the stressing event.

Once data have been collected, we store the relevant ones, that into rosbag files, that are special sequentially accessible files: we used these files to build graphs and show possible stress measurements to have a better view of the overall situation and spot the stress event beginning moment.

During the work of this project, we were able also to verify the results of papers which purpose was to demonstrate how reliable can be different physiological signals to be used as stress measurement.

The structure thesis is the following:

- Chapter 2 lists some works that have been done during last years concerning autonomous vehicles, describing also the topic of sensations, trust, and feelings human may have in an autonomous car. Then, it lists physiological stress indexes, some eye tracking implementations and the applications in a car, and finally a section briefly describing Deep Learning and Neural Network (NN) to explain how we performed object detection to increase obtained information from cameras.

- Chapter 3 describes briefly the experimental car, including the sensors mounted on the car and which data we get from them, the computer hardware needed and used to process the data collected from the sensor system.

- Chapter 4 focuses on the actual work of the thesis, describing the software structure, acquisition protocol, the data processing we performed to get the desired information and the software architecture, including developed and installed third party software.

- Chapter 5 describes the experiment in details and shows the final result of software execution.

- Chapter 6 lists conclusions of this work and possible future works that can further improve the information acquired and which could be the further steps for the I.DRIVE project.

Chapter 2

State of art

In this chapter we see the state of art concerning autonomous driving: since one of the possible evolution of I.DRIVE project is to develop an autonomous car that cares about passenger's mood, we analyse different works concerning developed autonomous car technologies and how to measure stress level.

First of all we give an overview about autonomous driving, describing the used technology and how it changed over the years, the different levels of automation defined and which level is achieved today and focus also about the financial impact of the autonomous car. Then we talk about human factors and autonomous cars, mainly those factors that hinder the entrance of autonomous car in the market and which solutions are proposed to ease this entrance: related to this topic, we focus also about the possible driving styles an autonomous car should adopt to drive the passenger comfortably.

To understand better which event have to be considered as a stressing event, we should measure the stress level of a passenger, so we cover also this topic by seeing the differences between physiological stress indexes in terms of reliability and usability and see which of these stress indexes well fits in our experiments. Since in our work we want to find causes of stress level increase on the passenger, we used an eye tracker to understand if the stress on the passenger was due to something he was looking at or not: we describe different eye-tracking implementation and which are the advantages of these implementations.

Finally, we conclude this chapter talking about Machine Learning (ML) and Deep Learning as we performed this kind of elaboration to increase information amount from the collected data.

2.1 Autonomous driving: levels of automated driving systems

Before describing the existing projects about autonomous driving, we have to define what an autonomous car is.

In 2013, the US Department of Transportation's National Highway Traffic Safety Administration (NHTSA) defined five different levels of autonomous driving differentiating how much control has the car or the driver:

- **Level 0:** No automation, the driver manages everything.
- **Level 1:** Most of functions are still controlled by the driver but one can be controlled by the car like cruise control; this level is also known as "driver assistance level".
- **Level 2:** The automated function disengage the driver from physically operating the vehicle by having his or her hands off the steering wheel AND foot off pedal at the same time like adaptive cruise control and lane centering.
- **Level 3:** Many function are managed by the car, however the driver is still present and has to intervene if necessary, but is not required to monitor the situation in the same way it does for the previous levels.
- **Level 4:** Fully autonomous car are able to perform all critical and safety manoeuvres and monitor roadway conditions in all the trip limited to the operational design domain
- **Level 5:** Fully autonomous system in any driving condition including extreme ones

There are few cars in the market that implement different levels of autonomy and at most they reach level 2 by implementing park assist or adaptive cruise control. In some countries is possible to find autonomous vehicle with research purpose only such as Google cars or Ford's autonomous car (*Figures 2.1b and 2.1a*).

2.2 Autonomous driving overview

In general an autonomous car should drive without causing accidents; this is translated in the realization of collision avoidance algorithms, bringing the passenger to the destination, i.e. navigation. One of the first works about

(a) Ford autonomous car

(b) Google car

(c) Uber autonomous car

Figure 2.1: Autonomous vehicles coming from different companies

autonomous car is Weisser et al. (1999) that developed a slow speed autonomous driving system based on computer vision and Differential Global Positioning System (DGPS) in which a robot was driving a van; the idea of a robot is a bit far from today's autonomous car concept, but it was necessary since the vehicle was not electric and needed interaction with throttle, brake pedal and shift: a further concept of that project is that it could be implemented in all cars so they developed a modular system. With progression of technology, more recent implementation of autonomous car avoid the construction of a robot by exploiting the drive-by-wire concept in a car in which all the driver's interactions are managed by a control unit.

Other popular works in the area of Autonomous Driving have, as main goal, the reduction in car accidents, which naturally have an implication in saving money and human lives. Although a huge progress has been done, many of the experiments were performed on driving simulators or secured environments that excluded from their model standard real world elements like pedestrians and traffic lights thus limiting the possible direct implementations of the solutions found. As an example we have the work of Urmson et al. (2008) (*Figure 2.2a*) during the DARPA Urban Challenge in which

(a) BOSS

(b) Junior

Figure 2.2: DARPA Urban Challenge partecipants

autonomous cars had to drive 97km through a secured urban environment interacting with the other vehicles and reaching predefined goals.

Although DARPA challenge brought interesting step forward in the autonomous driving, the urban environment was highly restricted as mentioned in the work, excluding cyclists, pedestrians, traffic lights and also some rules of the Department of Motor Vehicles. Since the car needed to manage lot of data in a short time, they used a parallel hardware architecture with 10 Core2Duo processors and 1TB of hard drives for logging.

Levinson et al. (2011) describes another work made for the DARPA Urban Challenge (*Figure 2.2b*) in which they had tackled the problem in an optimal sense in order to take advantage of optimal control theory asserting consistency in the choice of the best feasible trajectory over time. They considered also the ride comfort as subject of study, described as the derivative of lateral or longitudinal acceleration. The sensors used in this case are

quite the same of ones used in the previous described work, including sensors for localisation and environment perception (LiDAR and GPS), so they can apply all the robotic theory about Simultaneous Localisation and Mapping (SLAM); in the paper they describe also a method for calibration of LiDAR with an unsupervised intrinsic calibration to reduce sensor's noise by optimising parameters of each laser. About the computational power, they used two on-board Xeon processors, multi-core architecture that well fit the problem due to multiple simultaneous input from many sensors. In their work, they also faced the problem of object detection combining computer vision and, in the case of traffic lights, prior information. Luckily this technology has considerably improved in the last years and we focus on details later. In our work, we are doing experiments in a non-modified urban environment so that we know better the environment dynamics avoiding biased data.

2.2.1 Autonomous driving: economic benefits

In literature we can find many studies that focuses on the benefit on having autonomous cars: Fagnant and Kockelman (2015) estimate benefit around 3.000$ per year per vehicle if considering the fuel optimisation (or energy consumption in general with electric cars) and crash savings. Energy optimization is not only considered on how the Artificial Intelligence (AI) manage the throttle and brake, but also how managing the congestion in an intersection or a traffic roundabout can save time and energy.

In Fagnant and Kockelman (2015) study, a focus about car sharing is also given; may be much more convenient by offering on-demand services thus decreasing the on-road vehicles and the need of parking near malls; as counterpart, the autonomous cars may provide mobility for those who cannot effort it (too young/old to drive or disabilities) thus increasing the traffic congestion bringing problems mainly during the rush hours in already congested traffic roads.

Briefly, autonomous cars can increase safety, reduce congestion and need of parking having an estimated benefits of about $196 billion with 90% autonomous car market penetration rate, indeed this is the main obstacle to overcome. Other problems that prevent the autonomous car entering the market are discussed in Section 2.2.2.

2.2.2 Autonomous driving: human factors

Do people trust autonomous cars?

Recent studies focus on how much do people trust this technology by searching for key factors that can increase trust. Kaur and Rampersad (2018)

to achieve this goal made a survey in which they split the concept of trust in order to understand more in detail which are the major influencing factors. First, they divided the concept in trust the technology adoption and trust the autonomous car; about the technology, the key factors are trust in technology (placing the passenger in a vulnerable position with respect to the car), performance expectancy (concerning safety and free time) and reliability, limited only by the computational power.

About the autonomous car, the key factors are security meant as computer security, road irregularities avoidance, privacy as loss of autonomy as the car has a GPS may be subject to redirection aimed at targeted marketing or law enforcement; moreover we must consider the need for communication for intersection solving thus controlling cars each other and knowing the destination of the others. The way the I.DRIVE project increase trust in the technology concept, is mainly to avoid stressing condition to keep a "trust level" at a good level.

Helldin et al. (2013) demonstrate that using a User Interface (UI) to inform the passenger about the uncertainty of sensors of an autonomous car leads to a faster reaction in case of emergency thus avoiding accidents; in general, keep control brings better confidence in an autonomous car. The experiment was conducted in a simulated environment consisting of a fully functional cockpit in which the participants drove the car simulator through a snowed two-lane country-side road, with several sharp turns, but with no other traffic. The study concluded that showing the level of system uncertainty increases consciousness and trust in the passenger so that even if he spent more time looking away, he actually was ready to react and get the control on the vehicle in less time.

Inform the passenger about how and why the autonomous car is making choices is important to increase the consciousness and knowledge about how the system works hence the confidence a person gives to it. Koo et al. (2015) focus on this topic by making considerations about how a feedback system should be in order to make the human confident with an autonomous car; at first, they assumed that a feedback message should include the action the car is about to perform and then the reason of it. Again a secured environment was used for the experiment in which the street was projected in front of the driver: the projected road contain also some unexpected events, in those cases the car automatically brakes while a recorded message inform the passenger about the executed manoeuvre. During the experiment there were used different kinds of vocal messages to understand which words are better to use: the voice used was a standard American accent with no particular mood avoiding biased reactions.

After the experiment, the subjects filled a survey with questions about the driver emotional valence and the machine acceptance: they concluded that informing the passenger about "how and why" the autonomous car is acting brings a worse emotional valence and machine acceptance with respect to the only "why" message. This is explained by the fact that informing with a "how and why" message, gives more information in a context in which fast response time is required, thus increasing anxiety because more information had to be processed: quite the same results are obtained in case of simple "how" message because the passenger could not get the reason of the autonomous car action. About the overall driving performance instead, the "how and why" message brought better results, still less appreciated by the passengers.

Another more recent approach to increase confidence in autonomous cars is presented in Frison and Riener (2018) in which they suggest to inform the passenger about the autonomous car behaviour by using augmented reality approaches such as full-sized windscreen displays or contact lenses for a better system understanding to the user. This work hypothesises is that augmented reality assistance has the potential to increase user acceptance and trust by communicating system decisions.

To prove this hypothesis they conducted two driving simulator studies; in the first experiment, they evaluated whether augmenting traffic objects, relevant for manoeuvre decisions, can increase trust in autonomous car by driving the car in a simulated environment through a dense fog faster than a human would do, so actually driving in such condition human could not achieve without compromising safeness.

In the second experiment, participants were sitting on a rotating seat, facing against the driving direction and participants were presented the upcoming manoeuvres on a head-mounted display. The main idea of these conditions is to hide important parts of the environment to the participants so that creating augmented reality aids they can communicate system decisions and let users anticipate upcoming driving manoeuvres. In the first experiment, the car performed multiple overtaking manoeuvres with the car coming in the opposite lane not visible but detected and a marker saying if the manoeuvre was safe or not (a small triangle green or red).

For the first experiment, they concluded that augmenting traffic objects relevant to a driving scenario can increase user trust: for this conclusion, they measured different acceptance parameters. The second study shows that even if faced backward, presenting the manoeuvre as the actual driving direction rather than as perceived by the driver is much better and more accepted by the participants.

In conclusion, in the future, we can exploit augmented reality to show passengers information about the upcoming manoeuvre in order to encounter large acceptance and ease the entrance of the autonomous cars in the market.

Waytz et al. (2014) describe in a general way the design barriers engineers face when creating a new technology that can substitute human operators (in the paper, doctor's mind in diagnosis and autonomous driving are used as an example) and how anthropomorphism influences the perception of a person.

Anthropomorphism is a process of inductive inference whereby people attribute to non-humans agents distinctively human characteristic, particularly the capacity for rational thought and conscious feeling, so is not just adding superficial human characteristics, but rather attributing essential human characteristics to non-human agents; what Waytz et al. try to verify in their work is if people would trust more an anthropomorphized technology due to the common association between people's perceptions of others' mental states and competent actions.

To do this, they made trials with a normal car, autonomous car and anthropomorphic autonomous car giving the last one a name, gender, and a human voice; since they supposed that anthropomorphism could also mitigate blame in case of undesirable outcome, they implemented a virtually unavoidable accident during the driving simulation caused by another driver.

The results show that anthropomorphism is highly perceived and people trusted more the anthropomorphic car than the normal one; still, only 55% of participants preferred the anthropomorphic setting than the non-anthropomorphic car.

In general, the human factor is something we must consider in order to ease the autonomous car market introduction. In the next section, we discuss how a car should drive; although this concern human factor yet, it is not about trust the autonomous car but how pleasant it could be the journey by comparing common human driving styles and their reaction when they are passengers and the car drove autonomously.

2.2.3 Autonomous driving: driving styles

The idea one can have about the driving style is that a car should drive as the driver does; actually, this is not so straightforward and simple as Basu et al. (2017) explain in their work.

The goal of that study is to see the differences between the driving style of a driver and the driving style of the autonomous car with the same driver as a passenger; at the beginning, they have hypothesised that users of au-

tonomous cars prefer a driving style that is significantly different than their own. To verify this hypothesis they build a simulation in which people where asked to drive as they do as usual so to identify the driving style of the subject.

Then they set up four classes of driving style basing on the levels of defensiveness: aggressive, defensive, own style, and a distractor style (driving style of another test subject). Finally, each subject took place as a passenger and the car drove according to some different style, of course without informing him/her about which style was following, and at the end the subject had to answer three questions about the similarity of the perceived driving style with the proper style and if the journey was enjoyable. The results suggest that the car should drive in a safer way concerning the driver's usual style mainly because of the loss of control.

Going deeper on this topic, we can identify parameters affecting passenger's perception of autonomous car behaviour thus distinguish parameters that influence passenger's stress level and use them to develop an algorithm that takes care of comfort. What Bellem et al. (2016) do in their work is to identify physical parameters by performing two studies conducted one in a rural and urban environment and the other one on a highway; first, they identify 4 different manoeuvres:

- *Decelerating to a moving target*
- *Accelerating from non-zero speed*
- *Lane change*
- *Following at constant speed*

Knowing that human perception is highly sensitive to accelerations and jerks, as pointed out in Bellem et al. (2016) paper, they measured the trends of these measures during the manoeuvres performed in different driving styles: comfortable, dynamic and every day trip. These driving styles were chosen during the trip in random order by asking the driver to drive with his/her idea of the specific driving style: in this way, they were able to measure physical differences between the driving styles thus showing the ones with largest changes About decelerating to a moving target, they were not able to obtain enough data so in the urban case this manoeuvre is excluded: we show results in that environment since we are working in the same conditions.

The results of the study show that according to the performed manoeuvre, it is possible quite always to find differences between dynamic and the

		Overall effect		Everyday – comfortable		Everyday – dynamic		Comfortable – dynamic	
		F	*p*	*t*	*p*	*t*	*p*	*t*	*p*
Accelerating from non-zero speed	Acceleration	48.49	<.001***	5.06	.001**	−5.30	<.001***	−9.34	<.001***
	Jerk pressing gas pedal	17.64	<.001***	3.90	.010**	−2.69	.009**	−5.67	<.001***
	Jerk releasing gas pedal	34.87	<.001***	−4.29	.315	5.46	<.001***	6.48	<.001***
	Quickness	26.71	<.001***	3.40	.028*	−4.25	<.001***	−6.61	<.001***
Lane change	Acceleration	19.77	<.001***	4.35	.009**	−3.02	.002**	−5.81	<.001***
	Jerk	10.53	<.001***	2.94	.046*	−2.03	.071	−4.60	<.001***
	Quickness	14.32	<.001***	2.46	.071	−2.95	.005**	−5.59	<.001***
Following	Headway distance in seconds	31.47	<.001***	−2.51	.012*	5.07	<.001***	8.72	<.001***
	Standard lane deviation	3.10	.049*	0.65	1.000	−1.67	.313	−2.57	.052

* $p < .05$.
** $p < .01$.
*** $p < .001$.

Figure 2.3: This table summarise the results obtained in the urban environment showing comparisons between the driving styles comfortable and dynamic driving

other driving styles; the only exception is "standard lane deviation" that did not show marked differences between the driving styles.

Now that we can identify parameters that influence driving styles a question may arise: is it possible to model a customized automated driving style? An attempt to answer this question can be found in Scherer et al. (2015) work, in which they try to classify a manual driving style as comfortable and extracting different parameters, implementing an automated driving style. In order to extract the driving features and understand which one impact on the ride comfort, they used a driving simulator in which was installed a handset control that allows the driver to evaluate continuously the perceived comfort.

The experiment presented the same modalities as the previous ones cited, a simulation environment in which participants drove first and then experienced an automated ride with the driving style of other (or themselves) participants: the difference is in the handset control that allows the authors to find which were the situations in which the drivers felt uncomfortable and which were the comfort parameters; the main parameter was the longitudinal control so they focused on it. By approximating the driving style as the set of comfort parameters they were able to derive the optimized automated driving profile providing a modular procedure to build a driving profile.

2.3 Physiological indexes of stress

In our work, we are also aiming to study the interaction between driving styles and passengers by using an eye tracker trying to find a correlation between driver and environmental factors to identify the situations that impact most on the driver's stress level and successively on passenger's stress level.

In order to classify stressing situations, we need a physiological index of

the passenger's stress level: in general, many studies show that the sympathetic nervous system is activated in case of emergencies, giving more blood to the brain to make quick decisions thus increasing Heart Beat Rate (HBR) and sweating, hence, Galvanic Skin Response (GSR). What (Pedrotti et al., 2014) do in their work, is to mention many other studies that focus on the pupil dilation behaviour and on the reaction in the response of external stimuli like auditory, tactile, gustatory, olfactory, or noxious. Given this knowledge, they tried to measure the stress level on a person in a driving environment by simulating a route in which at some time a stress event occurred.

By maintaining a constant level of light, they were able to measure the pupil dilatation in the instants when the stress event occurred, finding a correlation between pupil diameter and stress event: finally, they built a NN that was able to classify stress moments from the pupil diameter. Since we are using the eye tracker to analyse driver gaze, we have information about pupil diameter so we try to extract stress levels by analysing pupil diameter trend.

Another physiological stress index known in literature is Heart Rate Variability (HRV), and in particular heart Low Frequencies (LF)/High Frequencies (HF) ratio. We have found different studies with different results: instantaneous LF/HF ratio is a well-known estimator of sympathovagal balance as described in Barbieri et al. (2018), citing other works that confirm this fact. However the correlation between sympathetic and parasympathetic nervous systems activation and LF/HF ratio depends on many other factors *(Figure 2.4)* as Billman (2013) demonstrates in their work, showing the assumptions to be made in order to consider that ratio as a reflection of sympathovagal balance. More specifically:

- cardiac sympathetic nerve activity is a major factor responsible of the LF peak.
- cardiac parasympathetic is exclusively responsible of HF peak.
- disease or physiological challenges provokes reciprocal changes in both sympathetic and parasympathetic activities still keeping a balance.
- there is a simple linear interaction between the effects of cardiac sympathetic and cardiac parasympathetic nerve activity on HRV.

A review about HRV made by Shaffer et al. (2014) explains more precisely how HBR varies and how this influences the different frequency components of HRV signal. This review states that the sympathetic system does not

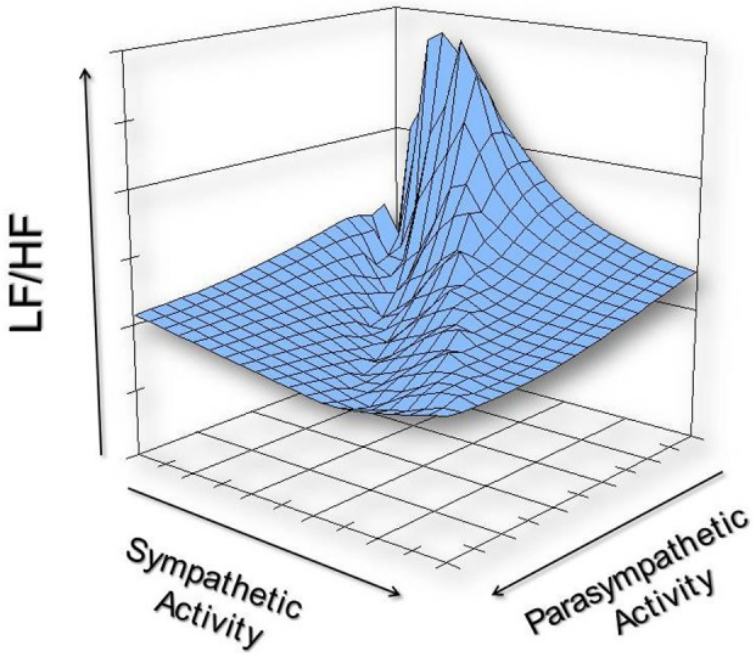


Figure 2.4: A 3D representation of all possible combination of sympathetic and parasympathetic activity with relative LF/HF ratio

appear to produce rhythms much above 0.1Hz, while the parasympathetic system can be observed to affect heart rhythms down to 0.05Hz: however during periods of slow respiration rates, vagal activity can easily generate oscillations in the heart rhythms that crossover into the LF band; this means that there is this third factor that can influence the LF component, altering the information about sympathetic - parasympathetic nervous system balance hence the measured stress level.

The review says also that, in long-term ambulatory recordings (about 24 hours), the LF band fairly approximates sympathetic activity when increased sympathetic activity occurs due to physical activity and also emotional stress reactions: however this is not true in short-term recordings thus decreasing the reliability of LF/HF ratio as stress index.

The paper concludes that psychological stress can also result in independent changes in sympathetic and parasympathetic activity, activating them simultaneously, confirming the fact that the relation between those nervous systems is not linear but much more complex.

In our study we must then take this into account if we want to get reliable indexes of stress: still we decided to measure HBR to see if in our dynamic and stressful environment the HRV gives useful information anyway.

Another stress index we can use is GSR; by applying a small electric charge on the skin surface, we can measure its conductance, giving us a sweating level of a person. Although sweating is the natural reaction to reduce human body temperature, Villarejo et al. (2012) described in their work how they built a galvanometer and which were the data collected in relaxing or high effort situations such as mathematical calculation.

Matteucci et al. (2016) in their work computed also two features of GSR: phasic GSR and tonic GSR. First, they compute the function describing the skin sweat concentration as a consequence of a stressing event; this function is named Bateman function. Then they deconvoluted the GSR signal with the Bateman function to obtain a *driver* signal and convoluted with a Gaussian filter to remove high-frequency noise. The result represents the Skin Conductance (SC) signal composed by the sum of phasic GSR and tonic GSR. By extracting local maxima, they obtained the phasic component as a series of pulse: the sections not included in those local maxima, indicates the absence of phasic component, thus representing the tonic one.

Finally, they interpolate data in these sections reconstructing tonic signal where also the phasic component was present finding the complete tonic component, then they subtract it to the original SC signal obtaining the phasic component that is the more informative part since more related with the sympathetic nervous system. Besides, they also developed a software that computes the HRV feature, LF/HF ratio, and we exploit that software for our purpose.

All previous physiological measurements have been used in Healey and Picard (2005) study in which they measured stress level by collecting data from GSR, Electrocardiogram (ECG), Electromyogram (EMG) and chest cavity expansion sensor for breathing rate. This is one of the few works in which the experiment was conducted in real-world and not in a simulated environment by let people drive in urban roads and highways; the goal of their study was in fact to find stress measures to monitor stress level in those different situations by combining physiological signals.

They performed two kinds of analysis, to point out general stress levels between the road sections and within a road section; the first kind of analysis results in a stress recognition developed using features derived from 5 minutes non-overlapping segments of data. More precisely the recognizer uses a total of 22 features extracted from the sensors to measure stress level:

- 9 features based on normalization of mean and variance extracted from respiration, heart rate, skin conductivity for both hand and foot and normalized mean of electromyogram signal.

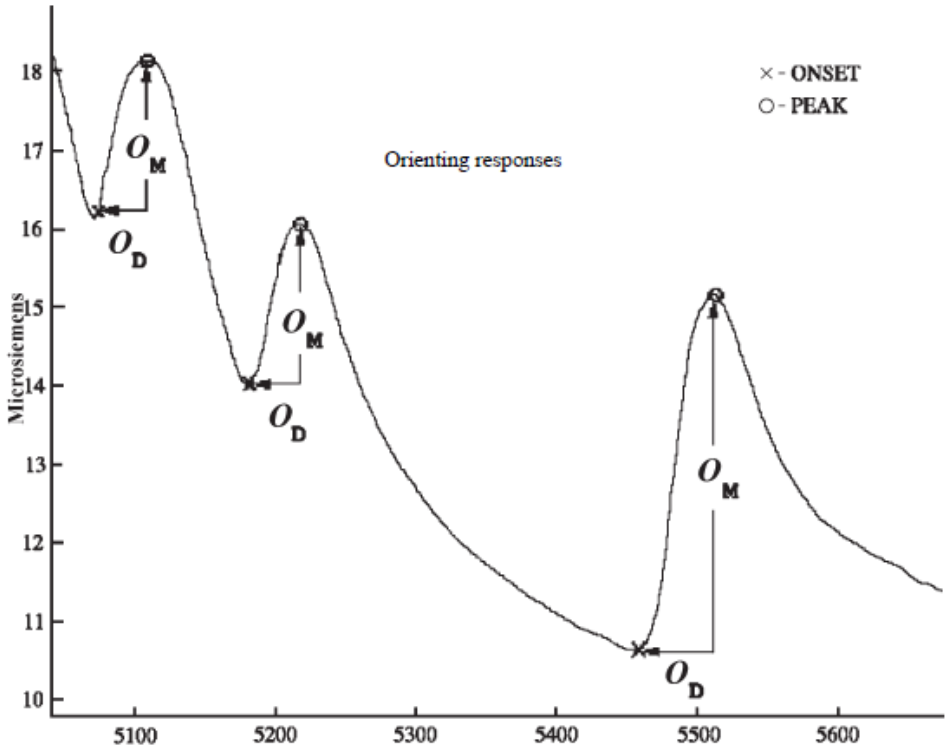


Figure 2.5: An example of three orienting responses, computation is performed after noise had been removed

- 4 features extracted from the breathing sensor in each of four bands (0-0.1, 0.1-0-2, 0.2-0.3, 0.3-0.4Hz).
- 8 features computed from the GSR signal, more precisely 4 features (the amount, the magnitude, the duration and estimated area) of the orienting responses that indicates sudden rising of the signal for both hand and foot
- 1 feature is the balance of sympathetic - parasympathetic nervous systems extracted from HRV by computing LF/HF

For the second analysis, which aims to get a piece of fine-grain information about the stress, first Healey and Picard identified potential stress factors as a list of observable actions and events by check the recorded video, like gaze changes, head turning, stops or turning. Once stress factors have been detected, they performed continuous calculations on physiological sensor signals at 1s intervals in order to find the features that might describe at best the stress level in the short period. Among all features, they also considered Mid Frequencies (MF), computing (LF+MF)/HF with 100s and

Figure 2.6: example of a small camera used in car's cockpit

300s windows advancing by 1s finding that the index calculated with the first window had good confidence thus used as stress index.

The study concluded that heart rate and skin conductivity metrics provided the highest overall correlations with continuous driver stress levels and that toll and urban roads had a higher stress level with respect to driving in a highway.

2.4 Eye tracker: methods and implementations

What we are doing in our work is to find as many indexes as possible to find, or validate, stress situations; to achieve this requirement we add the eye tracker that gives us information about the driver's gaze and pupil diameter. In this way, we know where the driver is focused on and moreover, we can try to correlate pupil diameter with other physiological features thus get a new potential stress index, as Pedrotti et al. (2014) did.

Today, eye-tracking implementation in driving field aim to give support during the driving task; to do this, the best way is to use the least invasive system possible such as cockpit mounted cameras (*Figure 2.6*) pointing the driver for fatigue monitoring and apply countermeasure like warning to bring again attention to the driving task (Singh et al., 2011) or give the vehicle a behaviour to avoid collision (Zutao Zhang and Jia-shu Zhang, 2006).

This kind of implementation however is very limited if we want to get the information about what the driver is looking at; Ahlstrom and Kircher (2017) made an important work about the analysis of the information retrieved by the eye-tracking system giving a semantic meaning; here the eye-tracking data were acquired with a remote five-cameras eye-tracking system and combining this with a 3D model of the car, they were able to say exactly what cockpit's component the driver was focused on.

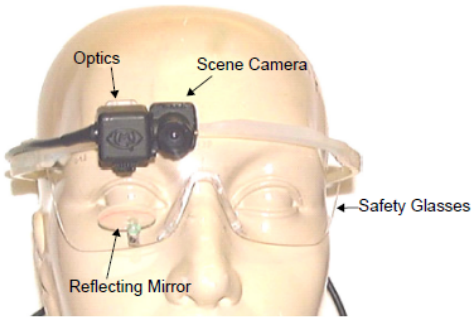


Figure 2.7: Single camera eye tracker

Of course, the system was enhanced with many other cameras to get information inside the car and outside, so they could correlate driver's behaviour to external situations, as overcoming cars, but the information retrieved in this case lacks in accuracy because they do not know exactly which external object the driver is observing.

The other interesting point of Ahlstrom and Kircher (2017) work is how the gaze sequences during the driving task change in various situations: with this analysis they discover similar sequences in similar situations so detecting behavioural patterns, then they could compute the average pattern time the driver spends to get the desired information, e.g., what sequence of glances and how much time do the driver need to get the speedometer value without distracting too much from the main task.

Fisher et al. (2007) in their work used a headset eye-tracking system (*Figure 2.7*) in a simulation environment to study the behaviour of the driver and evaluate his ability to predict possible hazards during the driving task. In particular, this headset was composed of an eye camera and a scene camera mounted on safety glasses and by having the road projected in front of the subject, they were able to see where the gaze was pointed in some hazardous circumstances.

Before the experiments, some subjects were trained with the Risk Awareness and Perception Training (RAPT-3) program developed at the University of Massachusetts that shows different categories of hazardous scenarios, so when the driver in the simulation encountered a similar scenario (or the same since some routes were repeated), they expect that he looks in the Region Of Interest (ROI) shown during the training. This was the indicator of how the training was successful and the results had shown that the trained group were focused more on the ROI thus able to anticipate hazard situations and react better to any eventuality.

Another interesting usage of the eye tracker is presented by Cerf et al.

(2008) in which they used the eye tracker to observe what object attract attention the most by showing the subjects different images that can be seen in an everyday situation, e.g., shops in a mall, a backpack with some objects, stairs with people, etc. Mainly they compare the attraction level of faces, text, meaningless scrambled text and smartphone, and the experiments shows that faces attract the most attention and they used this information to build a heatmap that could help prediction of scan path and the results were pretty interesting as they were able to predict subject scan path with a large confidence. For the experiment they used static images, so applying a heatmap in our highly dynamic context may not bring many advantages; however, we can exploit the information about faces that attract attention for a better analysis of our data.

2.5 Object detection with Machine Learning

For the analysis of the images acquired with the world camera and in particular performing object detection, we have used ML; in this section, we give a brief introduction to ML and Deep Learning, explaining differences and which is the NN architecture we used to perform object detection.

ML is that branch of AI which deals with finding a mathematical model to perform different tasks, e.g., explaining historical data to perform prediction or finding a separating hyperplane to perform classification.

This last task is the one we required since we have to detect and classify different objects we encounter during the drive and for this, Deep Learning is well suited since it has a good performance on speech recognition and image classification (Krizhevsky et al., 2012).

Schmidhuber (2015) presents a general overview of NN and Deep Learning, explaining the main applications, how generally works a NN, and which are the most common architectures. In general, a NN takes the idea on how a human brain works by simulating the behaviour of the neurons: dendrites collect input charges from weighted excitatory and inhibitory synapses and the axon fires the information once the charge is above a threshold. This behaviour is implemented with the artificial perceptron in which the threshold is implemented with an activation function and weighted synapses are simply the weighted input.

The perceptron is a very simple concept that can be used for simple linear separable input or the implementation of logical functions such as AND and OR: by increasing the complexity of the logical function we need to implement, we require more perceptron to work simultaneously, hence creating the NN. The same concept is applicable in case of classification task

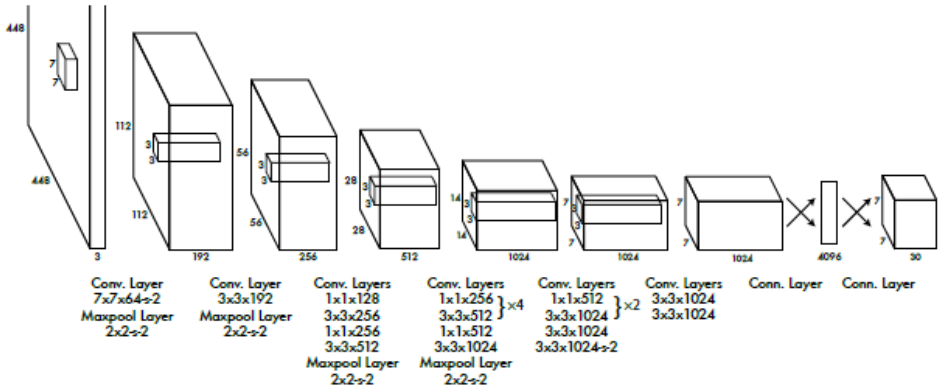


Figure 2.8: YOLOv3 architecture used for object detection

and, by increasing the number of perceptrons and layers of the NN, it is possible to distinguish many features of the input thus being more precise in the classification; finding the hyperplane that separates the input into classes requires training in which a loss function, most common the Least Squared Error (LSE), is used to adjust the weights' values.

There are some problems during the design of a NN; the most common is overfitting. The input we give to a NN comes from the real world, thus subject to noise; a complex NN would be able to distinguish many features but a too complex NN may consider noise as a feature, causing overfitting on the training set thus correctly classifying the training images but misclassifying all new images, making the NN useless. To achieve image classification we need to build particular complex NN architectures thus implementing Deep Learning: the idea of Deep Learning comes from cells found in the cat's visual cortex that fire in response to certain properties of visual sensory inputs like the orientation of the edges.

Girshick et al. (2016) discuss in their paper how Convolutional Neural Network (CNN) is a proper solution for detection, different from image classification because detection requires multiple classifications in a single image: the way CNN perform detection is simply implementing a sliding-window detector thus performing classification in a different part of the image returning probabilities of all possible objects. At first, they used the "recognition using regions" paradigm, which consists of generating category-independent region proposals and then extract the feature vector with a CNN: finally classifying each region with linear Support Vector Machine (SVM).

More detail on how Deep Learning is used for visual understanding is given by Guo et al. (2016) describing the most common Deep Learning algorithms and explaining also the role of convolution operation in a CNN and

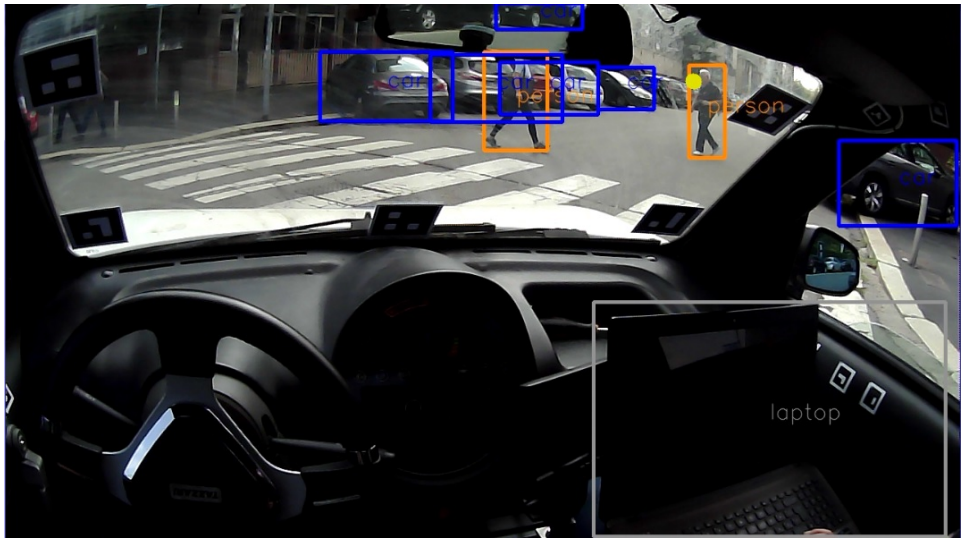


Figure 2.9: A frame showing the information acquired by the eye tracker: we can observe that even with objects partially covered, YOLOv3 is able to build a correct bounding box, also for cars visible from the rear-view mirror

the general structure and functions implemented. The paper focuses also on the roles of different layers inside a CNN and also why: in general the first part implements convolutional and pooling layers to extract as many features as possible. At last, we can find several fully connected layers that perform further features representation until the last layer that performs classification.

For our purpose, we used YOLOv3, a framework described in Redmon et al. (2016) work. With this framework is possible to perform object detection in real-time with enough computational power: since we do not have it, the elaboration is performed offline leaving a door open for improvement of the system with an online elaboration. YOLOv3 uses a convolutional architecture with 24 layers followed by 2 fully connected layers (*Figure 2.8*); the network is trained with ImageNet dataset and is able to classify more than 100 types of objects and animals; in order to simplify our work, we just consider classes of common objects can be encountered in an urban context. Compared with other deep networks used for object detection, YOLOv3 performances in terms of time and accuracy are better. in Figure 2.9 you can observe an example of YOLOv3 result combined with the information of the eye tracker (yellow spot): you can also see that markers for the definition of a surface do not limit the visibility of the road.

Chapter 3

Experimental vehicle

To achieve our goal, we have to collect as much information as possible about the stress level of the driver and the surrounding environment; as previously mentioned, many studies uses a simulated environment for their experiment thus the equipment was usually consisted in half car cockpit and a specific sensor for the study; to avoid biased data, our experiment took place in a real urban environment. The car we used for this project is a *"Tazzari Zero" (Figure 3.1)*, a small electric citycar with an asynchronous brushless engine that supply 15 KW power and max torque of 150 Nm The choice of an electric car is due to the simplicity of the throttle/break management algorithm since there is no need of shift and the electric engine can be easily directly controlled. More details about the structure of the sensors system, communication protocols, power consumption and acquisition and storing methods are reported in Matteucci and Gabrielli (2015) work; below a review of the sensors used for collecting data.

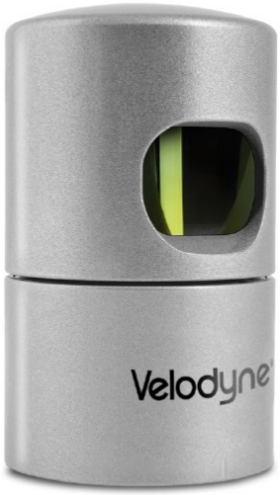
In this section we describe all the sensors by listing them according to the returned information, and grouping them according to the context data collected.

3.1 Environmental Sensors

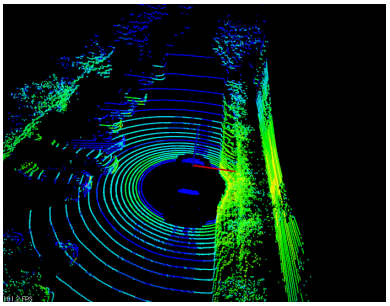
In this section we describe the sensors we used to get information about the surrounding environment, so to contextualise the situation and detect potential stressing factors and avoid them, as driving close to the preceding car or turning at a certain speed to reduce lateral acceleration.

The first sensor we introduce is the LiDAR sensor *Velodyne HDL-32E* (*Figure 3.2(a)*); LiDAR is a technology that measures distance by lighting a target with a laser and analyzing the reflected signal. Placed on top of

Figure 3.1: Tazzari zero without external sensors



(a) Velodyne HDL-32E



(b) point cloud

Figure 3.2: An example of point cloud (b) generated by LiDAR (a)

the car, *Velodyne HDL-32E* has 32 vertical lasers to get the distance of all the objects surrounding the vehicle in the vertical field of view between +10.67° and -30.67° and thanks to the rotating head at maximum rotation frequency of 20Hz and a sample rate of 700.000 samples/sec, it is able to

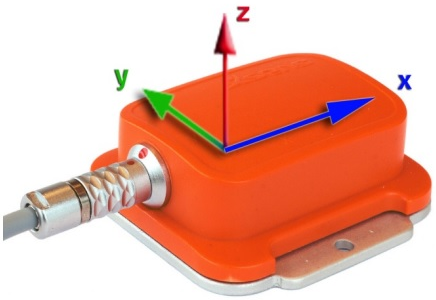


Figure 3.3: External IMU with the relative reference system

get a 360° horizontal field of view, getting the information of all surrounding environment between 1 m and 70 m far from the sensor with an accuracy of ± 2 cm.

Today is possible to find LiDAR with 64 or 128 levels in order to reconstruct better the surrounding environment but 32 levels are also good enough. The *Velodyne HDL-32E* model we used is also equipped with an internal IMU consisting in three gyroscopes and three two-axis accelerometers giving us also information about the dynamics of the vehicle during the driving task: this information could be useful for a more specific segmentation of the road by looking at the dynamics of the vehicle.

We used also an external IMU *Xsens MTi* (*Figure 3.3*) module that gives us more information with respect to the LiDAR's built-in IMU. The difference holds in the internal sensor of this unit: while LiDAR's built-in IMU consists of gyroscopes and accelerometer, while this one uses a 3D magnetometer giving data of terrestrial magnetic field, a single 3D gyroscope, and a 3D accelerometer. The maximum working frequency of the sensor is 100Hz and the information about the terrestrial magnetic field gives us the orientation of the vehicle; data collected by the *Velodyne HDL-32E* are sent to the on-board mini-pc through Ethernet cable, fast so can manage the large amount of collected data, while *Xsens MTi* exploit the USB port of the mini-pc.

The Velodyne LiDAR is equipped also with a Garmin 18LV GPS receiver. Although the path was on a fairly busy road, the GPS (*Figure 3.4*) information could be useful in order to retrieve possible environmental stress factors by performing street segmentation and distinguish different kind of road like intersections, straight roads, and turns; in this way we are able to associate instantaneous stress level to the performed manoeuvre in a specific condition. For the segmentation, we have retrieved manually the coordinates of

Figure 3.4: The GPS used for the acquisition

(a) (b)

Figure 3.5: Prosilica GC1020/650 (a) with Theia wide-angle optics (b)

beginning of each switch of type of road (straight line, turns, and campus), then we process all the coordinates to get an association between timestamp and type of road.

For a better view of what happens all around the car, we have also installed an external camera (*Prosilica GC1020 Figure 3.5(a)*) with a Theia wide-angle lens (*Figure 3.5(b)*) (125° horizontal field and 109° vertical field) but a very low distortion. The acquisition rate of 30 fps allows us to acquire enough data per second; resolution of 1024 x 768 is a good trade-off between the level of detail for the camera purpose and storage optimization. The time exposure of this camera was limited. To adjust the amount of light without compromising the fps rate. Diaphragm and focus point are set physically. In our acquisition we used those images to see the differences of the field of view with respect to the driver and compare also with data got from the LiDAR; in the future, with a proper setup, could be possible to combine eye tracking with the external cameras.

We also have internal cameras to monitor what happens inside the car; although these cameras are not strictly environmental, it can be useful to

Figure 3.6: Axis P13 camera

understand what is happening inside the car, so contextualising data. Two *Axis P13* (*Figure 3.6*) cameras are placed inside the car: one pointing to the driver and the other placed between the seats framing the cockpit and the windscreen; both cameras acquire data with resolution 800x600 @ 30fps. We have included data acquired from these cameras for future implementation like keep track of what happens inside an autonomous car, more specifically, the driver's camera frames could be used for Blood Volume Pulse (BVP) measured non-intrusively by evaluating the amount of red refraction from the skin of the driver; although very difficult, still is a way to get another sympathetic-parasympathetic nervous system balance index.

Concerning the central camera, acquiring both internal and external situation, its first purpose was for report events during the driving task, more precisely the car was going in a closed path many times and the passenger was visibly notifying the event of a complete lap: for future works, getting this kind of information could be useful to keep track of human behaviour while being a passenger in a fully autonomous car.

3.2 Cognitive sensor: eye tracker

The eye tracker we used in our case is the Pupil Labs® (Kassner et al., 2014) headset with 3 cameras, 1 for the world image and 2 for the eyes *(Figure 3.7)*. It is the only sensor that actually can be used to get environmental data, through the world camera, and physiological data, through the eye cameras measuring the pupil diameter.

The world camera uses a wide-angle lens and is capable to acquire images at different resolution and frame rate: 1920x1080 @30fps, 1280x720 @60fps, 640x480 @120fps. Since we need high quality for further elaboration, we have chosen the 1280x720 @60fps configuration that is a good trade-off be-

Figure 3.7: Pupil Labs® eye tracker with two cameras for the eyes and a front camera

tween sample frequency and resolution. The eye cameras can acquire the eye images up to 200Hz with an infrared sensor so it can find easily the pupil independently from the colour of the iris; their default and used configuration were 360x240 @120fps so to get enough information with a reduced workload and storage. Other possible configurations are 1920x1080 @30fps, 1280x720 @60fps and 640x480 @120fps. Together with the headset, the company provides also software for the acquisition and execution of acquired data and data exportation plus a USB(C) - USB 3.0 cable to easily connect the eye tracker to the computer.

A feature implemented in the latest version of Pupil Labs® software is the possibility to modify acquiring parameters to adapt images to the environmental light conditions thus increasing the quality of acquired images and reducing noise. This functionality allows for example to adjust the exposure time of camera sensors; actually, the setting is not fully manual because in case of low exposure the values of sensor sensibility ISO is increased to restore a good exposure level, indeed it was possible to notice some digital noise with low exposure time. This influences the quality of the images and the frame rate (longer exposure does not allow a quick frame rate); this parameter was adjusted for the world camera in order to get good visibility of the images even in the light conditions imposed by the dynamic environment we are working on.

Another important parameter that can be adjusted is image contrast: by increasing the image contrast amount on eye cameras, the pupil results much darker thus increasing the result confidence of the pupil detection algorithm.

(a) Procomp Infiniti

(b) GSR electrodes

(c) ECG unit for acquisition of HBR

Figure 3.8: Procomp Infiniti encoder (a) with GSR (b) and ECG (c) acquisition units

3.3 Physiological sensors

All sensors described below were used to get physiological indexes of the driver to understand his stress level during the experiment. The sensors used to get physiological signals were plugged to the Procomp Infiniti card (*Figure 3.8 (a)*), an 8 channels encoder used for real-time data acquisition that we used to synchronise and sample data from all the previous described sensors. Two of the available acquisition channels sample at 2048 samples/sec while the remaining at 256 samples/sec; after data are sampled, they are transmitted through TT-USB interface to the computer, with optic fibre cable ensuring fast elaboration and noise isolation.

Figure 3.9: This schema describes the placement of the electrodes with an example of the expected graph.

As described in the Chapter 2, the GSR is a well known method for measuring the instantaneous stress level on a person; in our case we use a pair of electrodes placed on index and middle fingers to measure the skin conductance through sweating (*Figure 3.8(b)*) between a range of 0.01μSiemens and 100μSiemens. This is the main measure we consider for the stress level: the sensor is attached to the Procomp Infiniti port that samples at 256 samples/sec.

Three electrodes are placed on the driver's body (*Figure 3.9*) to get the ECG hence the heartbeat frequency and obtaining more information about his general stress level computed in an interval of time; we have chosen this setup because R-R peaks are much more evident with respect to other electrodes placement, as. As in the previous chapter was mentioned the fact that computing HRV may not reflect at best the balance of sympathetic and parasympathetic nervous systems, still, we decided to compute this physiological feature to confirm this statement. The control unit that measures HBR is shown at Figure 3.8 (c) and is attached to the 2048 sample/sec channel.

3.4 Car preparation

The car has been modified with cables to the battery to electrically supply sensors and the on-board mini-pc *"Shuttle slim"* (*Figure 3.10*): then we used modular aluminium profiles inside the car to be able to install internal cameras, a USB hub to connect other components to the mini-pc. A second structure built with modular aluminium profiles is used to place outside the car GPS, LiDAR and two cameras.

Some markers stuck on doors and windscreen to identify different surfaces allow us to get information about the surface observed; in Section 4.1.1 we describe more this surface definition, giving detail about these markers and their functioning.

Figure 3.10: Shuttle slim mini-pc

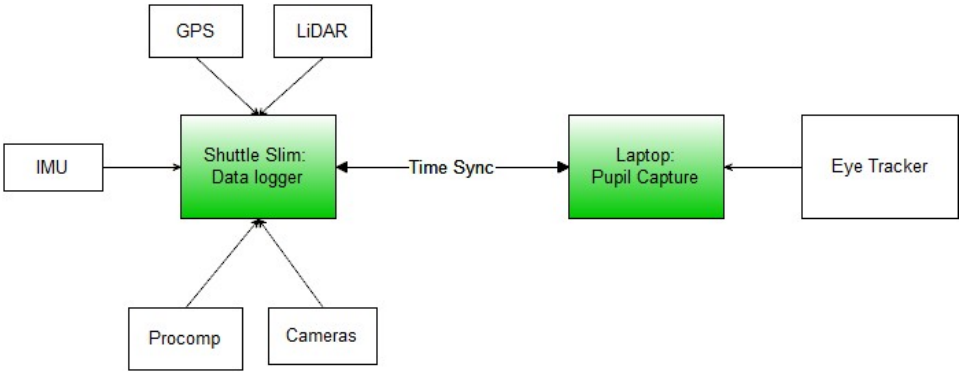


Figure 3.11: A schema of the hardware architecture for data acquisition with processes that run during the experiment and linked sensors.

The *"Shuttle slim"* mini-pc is equipped with an Intel Core i7-4790S with 4 cores in Hyperthreading (8 logical cores overall) and base clock frequency of 3,2 GHz, 8GB of RAM and 64GB Solid State Drive (SSD) for OS and 750GB Hard Disk Drive (HDD) for storage. This mini-pc is equipped with two network cards, one used for communication with cameras, the other with LiDAR. The compact dimensions make the mini-pc very suitable for our application also considering the limited available space inside the car (having only 2 seats): the only drawback is that it is not equipped with an nVidia Graphics Processing Unit (GPU) so to compute YOLOv3 detection we need a different computer with a high-performance graphic card. In Figure 3.11, you see a sketch of the hardware architecture, showing the two computers used during the experiment with the attached sensors.

In general, object detection is performed by using Deep Learning which is basically NN with a specific architecture generally with many hidden layers: performing object detection requires a large amount of computational resources, luckily is highly parallelizable process and this is why GPUs are highly suitable for this task: moreover, some developed scripts were designed to support parallel execution.

With these considerations, the hardware we recommend to execute our

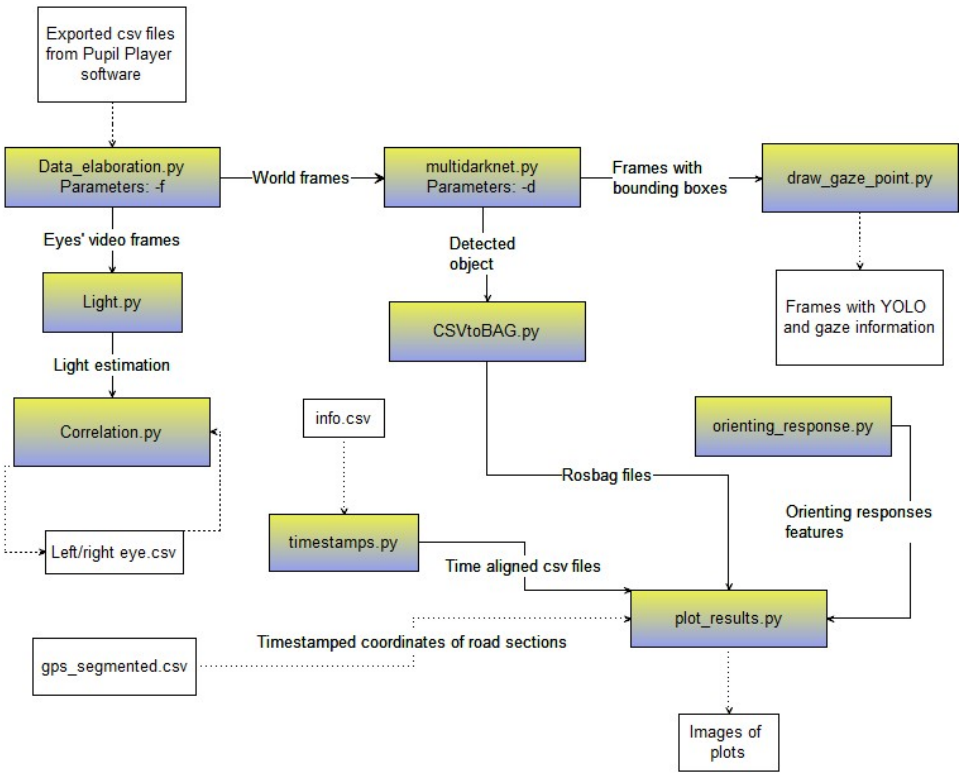


Figure 3.12: This dependencies graph shows the order scripts are executed: white boxes represent generated files while coloured boxes are the actual scripts.

script should include a multi-core architecture Central Processing Unit (CPU) with at least 4 virtual cores, 8 GB RAM and a GPU with at least 6GB of dedicated memory because YOLOv3 loads the entire network on memory for fast computation; we tried a GPU with 1GB of dedicated memory but wasn't enough to load even half of the NN. The configuration we used consist of a 8-virtual cores architecture CPU, 16GB of available RAM and an NVidia GeForce RTX 2060 GPU with 6GB dedicated memory.

3.5 Software architecture

Since use ROS for data storing and the APIs are already available in Python 2.7 and C++, we decided to use Python2.7 also for ease of usage for image elaboration thanks to the OpenCV module.

In order to maintain readability of software, we developed different scripts that implement different functionalities; in Figure 3.12 is represented the dependencies graph of the developed scripts showing input/outputs and

mandatory parameters for the first execution: since many of them generate files with the processed data, is possible then to call the scripts independently. A script with an incoming arrow cannot be executed until the script with the same arrow outgoing is executed.

Chapter 4

Data acquisition and elaboration

In this chapter we describe the activities we performed about data acquisition, going into detail on the operation we conduct before, during and after the driving: in particular about the setting phase, so all parameters we had set in order to increase the retrieved information from all the sensors, and the protocol we create to allow repeatability of the experiment.

After a list of preliminary operations and data processing, we briefly describe also the software architecture, including the developed, and the already available software for correctly use the eye tracker. Finally we show the results of our experiment to test the effectiveness of our developed software.

4.1 Data acquisition

With the goal to standardise the acquisition process, we had defined an acquisition protocol in order to increase repeatability of the experiment and try to evidence as much as possible the unexpected events; moreover this simplify future setups in case the car or some sensors changes.

4.1.1 Acquisition protocol

In this section we describe the acquisition protocol in terms of preliminary operation to do to collect data correctly. This protocol illustrates in details the following operations, listed in the order we performed them:

- Cockpit surfaces definition
- ECG electrodes placing

Figure 4.1: An example of detected surfaces: the right window is rarely visible unless the driver is turning right.

- Eye tracker calibration
- GSR electrodes placing
- Eye tracker synchronization
- Computing biometric sensors baseline

In the following sections we show more details for each step of the set up protocol.

4.1.1.1 Surface definition

To gain more information during the driving task, more specifically about the direction the driver is looking with respect to the car orientation, we had paste inside the car some markers specially created by Pupil Labs® (*Figure 4.2*) to define surfaces and study the behaviour of the gaze on them.

We used those markers to differentiate the surfaces of windscreen, plud left and right windows (*Figure 4.1*): the script that generate the images of markers can be retrieved from the Pupil Labs® website and by setting some parameters we were able to print larger markers. This was necessary because the original script produces very small markers that could not be recognized if pasted on windscreen due to the long distance with respect to the driver; moreover, the position of markers for windscreen detection was carefully chosen in order to exclude the rear-view mirror thus avoid false positives due to cars reflected on it.

Figure 4.2: Example of markers used for surface definition

The markers recognition is a built-in feature of the eye tracker software, so we had just to put the markers inside the car, start surface definition on Pupil Capture software, look at the surface ensuring all markers are visible, wait for recognition and name the different surfaces to identify them and adjust surface shape in case of detection errors.

In this way, the capture software had also the ability to recognize surface-gaze related events such as how long did the driver focus on a particular surface and when the gaze moved from a surface to another; this can be useful to determine shift of attention and the direction of the distraction factor thus improving the information acquired. The surface definition do not require eye calibration, so is performed at first, to reduce risk of moving the headset after calibration thus compromising accuracy.

After surfaces have been defined, we can proceed with the placement of electrodes for ECG signal as described in Section 3.3. Since the electrodes for measuring GSR are a bit annoying due to the cables connecting the electrodes to the processing unit, the first electrodes placed were for measuring ECG, less annoying since not limiting the body movements.

4.1.1.2 Eye tracker calibration

The first operation on the eye tracker's parameters is to set some parameters for camera eyes in order to increase pupil detection confidence thus reducing noise. Due to the high variability of light conditions dependent from weather, period of the year, the street and so on, there is not a setup that works in all conditions so you may adjust parameters according to the light conditions. Below a list of operations on parameters that influence pupil detection confidence:

- Increasing the contrast on the acquired images from the eye cameras makes pupil darker so easier to detect.

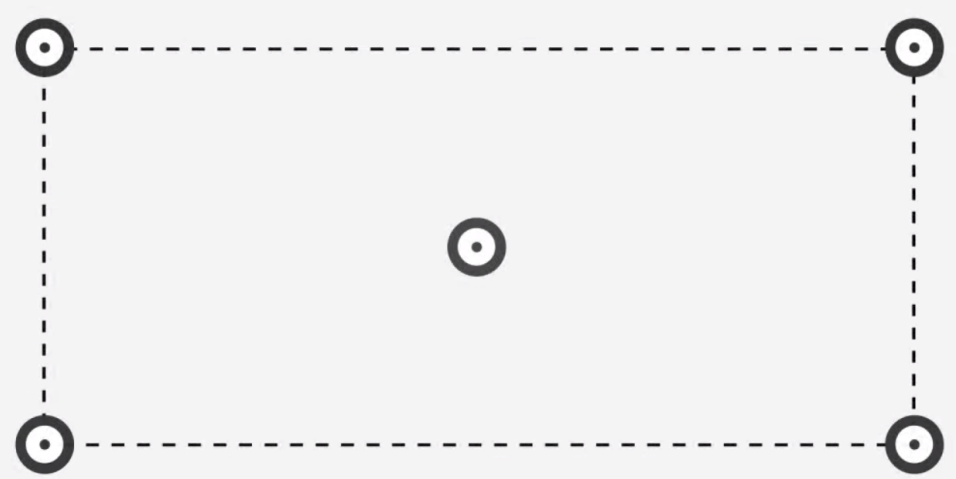


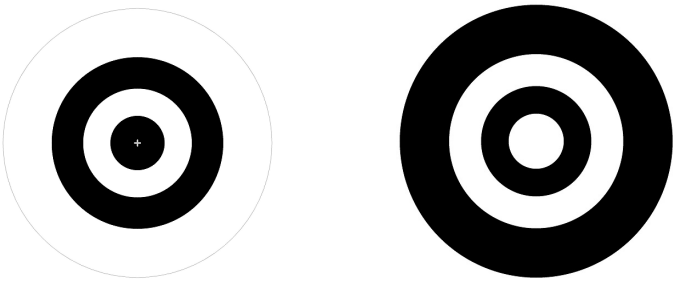
Figure 4.3: This figure shows approximately where the markers appear on the screen: the markers are shown in sequence starting from the one in the centre then proceeding clockwise from the top-left marker

- Setting minimum and maximum pupil diameters helps the detection algorithm to reduce false positives.
- Low values of Pupil intensity range help to reduce false positives with dark eyelashes.

All the above parameters can be adjusted from the Pupil Capture eye UI: after setting all parameters and verifying that pupils are detected with high confidence, we proceed with calibration.

Pupil Capture software default calibration method uses laptop's screen showing five calibration markers on centre and on the four corners of the screen (*Figure 4.3*): the drawback of this method is that is optimized for close range applications, unfeasible for our application in which gaze focus is relatively far. However the software has the possibility of calibration for long distance applications by using the manual calibration method in which an operator holds a printed calibration marker shown in Figure 4.4 (a) while the subject wearing the eye tracker focuses on the marker centre without moving the head.

The operator moves the marker inside the field of view of the camera at a fixed distance of 2 meters; a sound produced by the software informs about the progress of the calibration task. Once over, the operator shows the end calibration marker visible in Figure 4.4 (b) so that the system stops acquiring data; then the driver must be careful to do not move the headset otherwise the accuracy could be compromised: the entire procedure is performed in the



(a) Calibration marker (b) End calibration marker

Figure 4.4: default markers used for manual calibration

lab to keep the computer plugged and for better manoeuvre space and static light conditions. We are then ready to enter the car for the last operations.

4.1.1.3 Eye tracker synchronisation

Using different devices for acquiring the information, brings the problem of timestamp synchronisation between computers. Since all on board sensor were connected to the installed mini-pc through Ethernet cable or USB using Procomp Infiniti encoder (*see Chapter 3.3*), the timestamp was based on the on-board computer. We decided, since eye tracker's software needed a great computational power in order to avoid loss of data due to large workload, to bring an external laptop.

So before start recording we had to synchronise the clock of laptop and the on-board mini-pc by connect both to the car's Local Area Network (LAN). The synchronization method we used was the Network Time Protocol (NTP), setting a hierarchical structure in which on top we place the trusted information source (in our case the GPS receiver); the information is then propagated to the lower levels of the hierarchy with the client-server paradigm in which top levels act as server while lower as client. This protocol continuously synchronises the computers' clock thus increasing reliability and robustness in case of failure. More details about the protocol can be found in Matteucci and Gabrielli (2015).

4.1.1.4 GSR electrodes placement

While waiting for the computers synchronization we finally placed the electrode for measuring GSR: easy to wear since must be placed on index and middle fingers with Velcro. Finally we do a brief visual check to ensure that all the sensors were working properly.

4.1.1.5 Biometric sensors baseline

Last step before start driving, is the creation of a baseline for the biometric sensors so the driver must stay still for about 3 minutes trying to relax as much as possible with closed eyes and in silence; this baseline is then compared with the driving task giving us a measure on how stressing is this task. Even if not required, we start getting data from the eye tracker through Pupil Capture software during this baseline, mainly to have images from the world camera: since eyes are closed, is impossible to retrieve data about gaze and pupil diameter.

4.1.2 Drive acquisition

The route the driver had to travel was predefined in order to be sure to include potentially stressing situation such as large and busy intersections with possible presence of on-rail public transportation and pedestrian crossing. Having a predefined route gives us the advantage of repeatability of the experiment with different subjects and see how they react in the same potentially stressing situations. Due to laws of the Italian highway code restriction and car's characteristics, we could not include highways so we focused on a urban environment (*Figure 4.5*). During the driving, the passenger just check the correct functioning of the data acquisition system without changing acquisition parameters of the eye tracker.

4.2 Data processing

This section is about all kind of data processing we performed to get the desired information, starting from preliminary operations to reduce noise and outliers, to all the different processing to increase retrieved information, store data, find stress indexes and perform data presentation.

4.2.1 Data cleaning

Once acquisition ended, we passed eye tracker data acquired from Pupil Capture software to the Pupil Player software that computed the surface positions from world camera images and the fixation points with its duration. After offline surface and fixation data computation ended, we selected the confidence threshold of pupil detection algorithm to remove noisy data and finally start export, generating *csv* files; we have used these files to build rosbag files. Before creating rosbag files and start data elaboration, data cleaning had to be done, mainly concerning the surface detection with

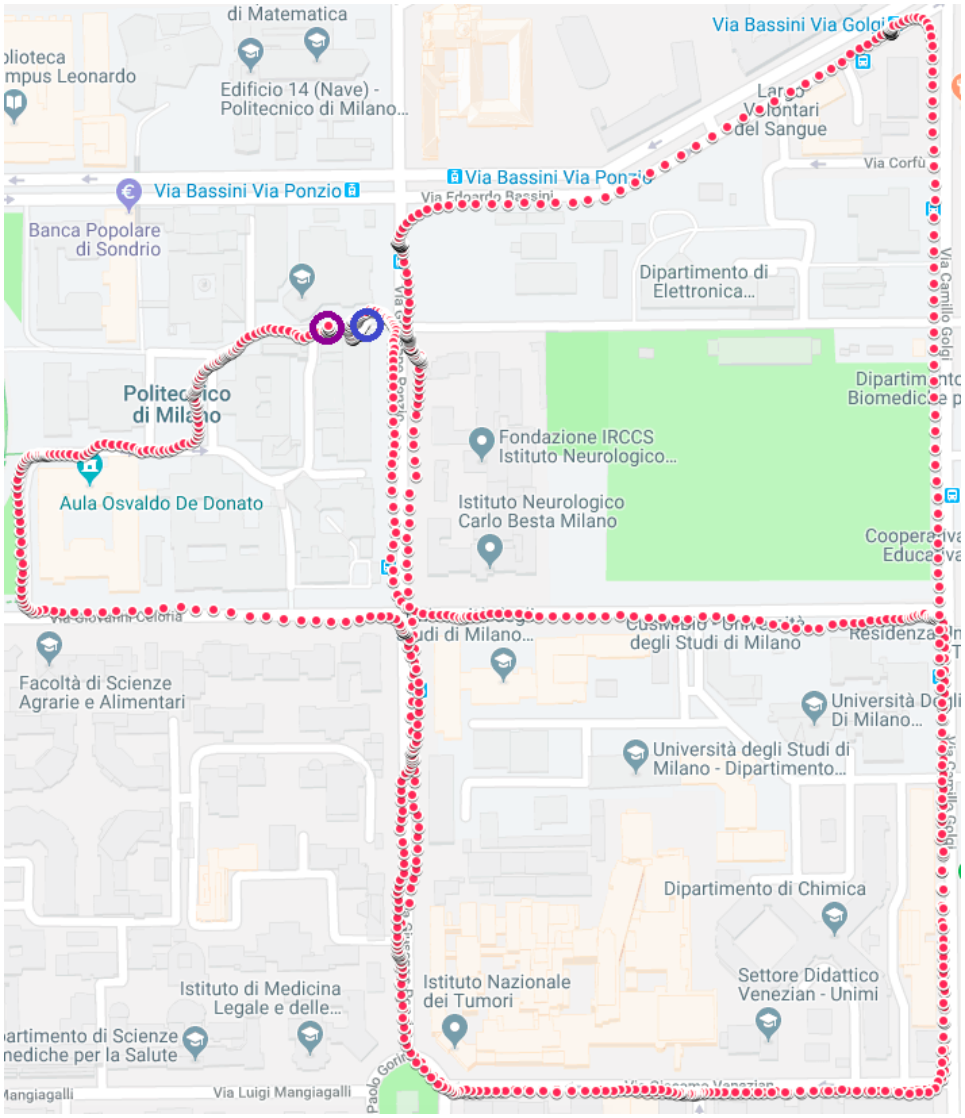


Figure 4.5: This is the route of the first acquisition we performed, starting from blue circle and ending in the purple one; the route included intersections, pedestrian crossings, traffic lights and a short path inside the Politecnico di Milano campus.

the eye tracker and the confidence of gaze position. The most noisy data about surfaces regard the detection of the left window's markers. Since the surface position is close to the driver and its orientation is almost orthogonal to the gaze direction in most of the time, the problem that arise is that some markers are often not visible hence reducing the probability of surface detection; another problem is that in some unknown circumstances, the algorithm fails to give the correct shape of the surface, by roto-translating it and over-

Figure 4.6: A picture of the result of YOLOv3 elaboration during the first acquisition

lapping with the windscreen surface thus bringing error. Luckily a manual correction of surface definition is available in Pupil Player software so, once video has been recorded, we were able to adjust the surface definition to minimize these errors.

About data collected by the eye tracker, cleaning was quite easy since every feature had a confidence related, e.g. pupil diameter and gaze position; for our studies we have chosen data with a confidence larger than 50%. About pupil diameter we have exploited the Pupil Capture software capability for blink detection to remove those data from the analysis, as blinks are considered outlier in this context. Concerning data acquired by ECG, the cleaning process aimed to remove power grid noise at 50Hz so was applies a band cut filter while for GSR a low pass filter was applied to remove high frequency noise. More details about the elaboration concerning these data can be found at Matteucci et al. (2016).

4.2.2 YOLO elaboration

Once cleaning is completed, we proceeded with YOLOv3 data elaboration (Redmon et al., 2016), a framework for Deep Learning that uses CNN to perform object detection. We use object detection in order to gain more information on what is happening during the driving task, trying to find as much factors as possible that can increase the stress level of the driver. Combining this with the gaze information we know what the driver is looking at during the driving and get, in this way, an eventual index of distraction

in case he is not paying attention to the road.

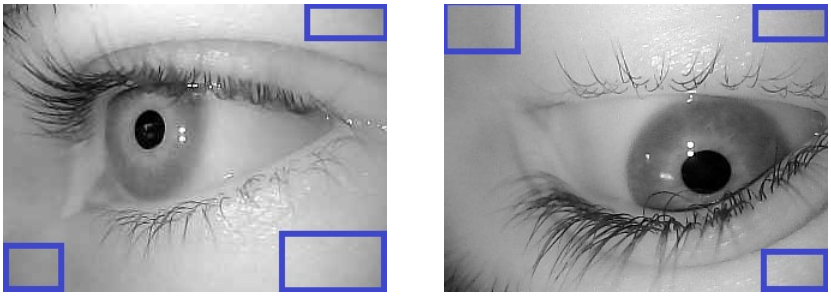
By default, YOLOv3 is trained to identify more than 100 classes of objects that can be encountered in a variety of situations: for our purpose we distinguish only 8 classes of interest to simplify our analysis, including:

- *car*
- *truck*
- *bus*
- *traffic light*
- *motorbike*
- *bicycle*
- *person*

Anything else is referenced as "other". After elaboration was performed, we proceeded with the cleaning of the output of YOLOv3 elaboration; more precisely the shown bounding boxes have a confidence of more than 25% by default and, although low confidence, the results were pretty accurate (*see Figure 4.6*). The only problem we faced is that since the images acquired were taken from inside the car, YOLOv3 detected most of the time the cockpit classifying it as a car, so nullifying the information that could be derived from this elaboration. To overcome this problem we simply excluded from the analysis all the bounding boxes which dimension was greater than 60% of the entire frame: the result is that no frame had detected the cockpit as car excluding any false positive.

4.2.3 Light levels

A second elaboration came from the need of correlating GSR with pupil diameter in order to validate it as stress measure: knowing that the pupil main function is to regulate the amount of light that reaches the retina, being in a dynamic environment brings us to be careful to correlate pupil diameter and GSR. We do not have a luxmeter to measure the level of light, but since we are interest more on identify the variation of light levels than the level itself, we remedy to this problem by computing the average RGB values in some region of the eye images where eyelashes and eyebrows were excluded (*Figure 4.7*): the regions are chosen by considering different images so to be sure to excluding eye lashes that could bring a misleading value with eye movement. We then have a value between 0 and 255 indicating the level of



(a) Left eye regions of interest (b) Right eye regions of interest

Figure 4.7: Two examples of region used for the computation of light level.

light: greyscale images simplify our work because RGB values are equals. Finally we correlated light level with pupil diameter finding a correlation of about 70%, good enough to state that beyond noise, this result evidence that there could be other factors influencing the pupil diameter among which stress level. In Subsection 4.2.6 we describe better the process we went through to compute the correlation between pupil diameter, GSR and light estimation.

4.2.4 Rosbag file creation

Once the acquisition session and data cleaning task terminates, we use the Pupil Player software to export the interesting data: the output of the exportation function is a set of *csv* files containing all data about eyes (pupil diameter, pupil position, gaze position in 3D space, blinks, fixation points) and surfaces (position and eye-related events). For a general better view of all data, we exploit a tool in ROS, a framework used for robot simulation that uses publisher/subscriber paradigm, and that allows us to store data in an indexed file named "rosbag"; by using the timestamp based on the same event as index, we had a perfect synchronization between all the sensors (with computers' clock already synchronized).

Data in rosbag files are referred as messages, hence we first build customized messages that are simply files with *.msg* extension containing all the information we want to store in the form of variables: then we write a python script that gathers data from the *csv* files gave as output from the export function of pupil player and build the rosbag files.

For presentation and debug purpose, we also decide to store all images acquired from the eye tracker, hence right-eye, left-eye and frontal camera images not processed with YOLOv3: so for each type of information we want to store we create a topic and by cycling row by row the desired *csv* file, we

fill the message and store in rosbag file. This is all about the data collected by the eye tracker, concerning data stored in the on-board mini-pc first data were stored in MongoDB a database for applications with heterogeneous data able to automatically define the data model to be stored, avoiding to define the model for each sensor. More details about rosbag file creation script in Section 4.4

4.2.5 *Mat* file creation

Matteucci et al. (2016) work already had computed stress level by extracting GSR and ECG features and storing data in *mat* file (LF/HF for ECG while Integrated Skin Conductance Response (ISCR), phasic and tonic GSR): this because data elaboration in that work was performed by using Matlab® and *mat* extension is a default storing format. Although specific for that software, is quite common file format and "Scipy" python module automatically implements a function that import *mat* file content into a python list, that could be easily transformed into a data frame. In the next section we focus on GSR and ECG features.

4.2.6 Stress indexes identification

In order to understand the causes of stress of the driver, we try to find a correlation between physiological and external factors. With the hypothesis that GSR is a reliable stress index, we used Pearson Correlation Coefficient (PCC) to see which of the others index can be used for validate stress level; given two random variables X,Y PCC is computed with the following formula:

$$\rho_{XY} = \frac{\sigma_X \sigma_Y}{\sigma_{XY}}$$

where σ_X and σ_Y is the variance of the random variables and σ_{XY} is the covariance between the random variables.

The first computation is performed in order to validate the stress level given by the physiological factors, in particular we tried to find correlation between GSR, pupil diameter and LF/HF. In general we expect to find large correlation between different stress indexes and assuming that GSR is the most reliable that we have, we developed a script that computes correlation between different data, more details in Section 4.4.5. As mentioned in Chapter 2, Pedrotti et al. (2014) demonstrate that pupil diameter can be used to determine stress level of a person. However they were in a simulation environment in which the light could be controlled; in our case we had a dynamic environment, e.g. a route with trees and buildings so we had also

to take care the amount of light: as already mention in Chapter 4.2.3 we did not had any photometer inside the car, hence we proceeded by analysing the image of the eyes given by the eye tracker and the greyscale image also simplify our purpose so we had just to compute the RGB average values in the areas around the eye excluding the eyelashes. This gave us knowledge about the trend of light during the driving task, so we were able to relate light with the pupil diameter thus get the information theoretically related to the stress level.

Before computing the PCC between amount of light, pupil diameter and galvanic skin response, we had to perform further computation since there are different sampling frequencies between sensors. Since PCC needed the same amount of data between signals, we had three choices: replicate or interpolate data to bring lower sample rated sensor to higher sample rated, or aggregate data from the higher sample rated sensor.

We actually tried all: replication simply, for each high sample rate data timestamp, copy the value with closest timestamp from the smaller data frame. Aggregation instead computes the average of chunks of high sample rate sensors values and associated to low sample rate sensors. Finally interpolation computes linear interpolation to assign a more precise value with respect to the replication, so we expect better results with this last method. As we could imagine, the pupil diameter had a larger correlation light estimation than with GSR so we applied an adaptive filter in order to estimate the pupil diameter given light estimation; assuming that pupil diameter depends only on stress events and light, the expected error computed with both Normalized Least Mean Squared Error (NLMSE) and Recoursive Least Squared Error (RLSE) should evidence stress events thus finding a correlation with measured GSR.

As already mentioned in Section 2.3, another possible stress index could be retrieved by computing LF/HF as it is a valid method for measuring the sympathetic - parasympathetic systems balance. Actually, since there are many factors that influence the nervous system balance is not always a good idea to use LF/HF ratio: beyond that, the fact that this ratio consider frequencies of a slow changing signal, in order to compute a good estimation of stress we need to evaluate this index in a time range of at least two minutes, making this index unfeasible for instantaneous stress level measuring but theoretically good for average stress level in that time range.

However, according to the computation performed by the script developed by Matteucci et al. (2016) in which the index was computed within a time range of 120 seconds, and knowing that at a certain point of the acquisition a stressing event occurred, by looking at the ratio we observed a

low level of computed stress close to that moment contrary to our expectations, so in the end we decided to consider only GSR as instantaneous stress index: more details in Section 4.4. From the GSR data we can extract the phasic and tonic components: those features could be used for average stress level while other computed features as magnitude, duration and number of orienting responses gives an instantaneous stress level within a time period.

4.2.7 Data visualization

Once all data were collected and cleaned, the idea of our work is to present the result in a easily understandable way. To do this we plot all data concerning GSR, ECG, observed objects, present objects, gaze position and pupil diameter: in Section 4.4.8 is described the script that generate plots with all the interesting information and how parameters affect the final result.

4.3 Pupil Labs® software

Here we describe briefly the eye tracker's available software and the used features; a complete description of both software and the user interface can be found on the website **https://docs.pupil-labs.com/**. For our work we installed version 1.11 of the software.

4.3.1 Pupil Capture

Pupil Capture is the software responsible for eye tracker calibration and acquisition; in Figure 4.8 is possible to see a sketch of UI and more precisely:

- **Panel 1** shows information about CPU utilization, FPS and eyes confidence
- **Panel 2** list the available hotkeys: calibration, test and recording are default
- **Panel 3** shows option of the selected functionality
- **Panel 4** list all the available functionalities

From panel 4, we can select the plugin to use to get more specific information, change image acquisition parameter, choose calibration method and more. For the acquisition we activated Blink Detection, Fixation Detector and Surface Tracker plugins: data about gaze and eyes are stored just after recording ended. In case of acquisition from smartphone, to reduce the workload, Pupil Capture software records only images from cameras without

Figure 4.8: A sketch of pupil capture interface in which we can see the acquired information.

further computation: to get information about gaze and eyes, Pupil Player software allow to compute them offline from images by using computer vision built-in algorithms.

4.3.2 Pupil Player

After the acquisition, we start Pupil Player software that automatically computes surface position and fixation detection from the recording. The UI is the same of Pupil Capture software so again from panel 4 (*Figure 4.8*) we can choose the plugins to export the desired data: we selected Offline Surface Tracker and Raw Data Exporter that generates *csv* files. Finally we press the export button (*Figure 4.8 panel 2*) and can proceed with Python script execution.

4.4 Developed Python scripts

In this section we show briefly the pipeline and role of the developed script; the overall documentation is in Appendix B, describing more in detail the implemented functions and the parameters of these scripts.

4.4.1 Data elaboration.py

This is the first script we run just after the Pupil Player exportation task terminates: the listed parameters in Figure 3.12 must be present in the first execution in order to generate files required by the other script such as frames from the eye tracker cameras read then from multidarkent.py script. It is possible to add other parameters to execute Data elaboration, multidarknet, Light and CSVtoBag scripts in the correct order thus increasing automation of the execution. This implementation choice increases the robustness of the scripts: in case of error of a called script, the caller could disable succeeded script to reduce execution time avoiding redoing job pointlessly. In the case of multiple acquisitions in a single day, this script can process data of all the acquisitions separately to automatize more the entire process.

4.4.2 Multidarknet.py

This second script simply calls YOLOv3 detection method that returns for each frame the list of all detected objects with centre, width, and height of the corresponding bounding box; in this phase, we also remove all bounding box larger than 60% of the frame resolution to avoid the cockpit to be classified. In this script we also reduce the number of classes of detectable objects by classifying as "other" whatever is not listed in a predefined dictionary (that also corresponds to the list of class in the custom ROS message). The output of the detection function is then saved into a .txt file so that script "CSVyoBAG.py" has all information needed to build rosbag files. By running this script with a specific parameter, enables a function that draw on the images the computed bounding boxes, basically to check the correct functioning of the CNN.

4.4.3 CSVtoBAG.py

This is one of the scripts with largest workload since has the role to merge all the collected information and store into rosbag files. Data in rosbag files are organized into ros messages with relative topic so they could be easily retrieved: a topic is created for each kind of information, then a set of custom messages is created with the related information, filling the header of each message with the correct timestamp to have synchronization between all topics. Among all information, we also save as special messages also all the frames of the three eye tracker's cameras: this is the bottleneck that slows down a lot the execution of the script. The script also computes automatically time alignment from pupil timestamp to Unix like timestamp so

that data in bag files are already synchronized with data stored in MongoDB.

4.4.4 Light.py

This script performs light estimation as described in paragraph 4.2.3 saving the results in a file, one for each eye, with just the values of light at location:

date_ of_ recording > Output > Light_ *acquisition number

The script that takes in input those files automatically matches light value to the diameter frame having a 1:1 correspondence and so easily associate light estimation with the timestamp.

4.4.5 Correlation.py

This script computes the correlation between data, in particular, we used it to correlate different stress indexes in order to validate them. Assuming that the GSR is the most reliable stress measurement we have, we expect to find large correlation with the other supposed stress indexes: the computed correlation index was PCC. It must be launched after Light.py script because we have to correlate pupil diameter with light estimation to retrieve stress levels from it. More specifically we have to find a process that decreases the pupil diameter correlation with light estimation but increasing pupil diameter with GSR otherwise we cannot distinguish which of those two measures affect most the diameter hence we could not be sure of using it as stress index. An issue we had to overcome was about the different amounts of data between GSR signal and pupil diameter because PCC can be computed only if two series have the same amount of it; we tackle this problem with 3 approaches:

- Aggregate data, performing a sort of downsampling
- Replicate data thus performing upsampling
- Interpolation to upsample data and have a better estimation of the result

For the aggregation, we computed the average value of light estimation and pupil diameter and associate the result to a single timestamp; as expected, since aggregating data brings a loss of information, the correlation showed the lowest values so we decided to discard it. Concerning upsampling, we tried interpolation that gives a good estimation of the halfway values but increasing the time complexity of the algorithm by a constant. The replication

solution was implemented mainly to reduce execution time still maintaining a good correlation result but in the end we decided to perform interpolation being more robust to noise and having better results overall.

4.4.6 Timestamps.py

Once on-board mini-pc and laptop clocks are synchronized, we noticed that Pupil Capture acquires data starting from an internal event of the computer while all the other sensors, since working with ROS, have all a Unix based timestamp that is the number of seconds from 1 January 1970 00:00 UTC, so before plot data, we have to align all the timestamps. Luckily when the acquisition starts, a *csv* file with Unix based timestamp (t_{Unix}) and the timestamp of the internal event (t_{start}) is created so we could align timestamps with this simple formula:

$$t_{aligned} = t_{acquisition} - t_{start} + t_{Unix}$$

This script save the elaboration in new *csv* files, in this way no further elaboration is needed to plot aligned data.

4.4.7 Orienting responses.py

In a study described in Section 2.3 a feature extracted from GSR data were the orienting responses, that are all quick changes on the value of GSR. Before detecting each orienting response, all data are first filtered with a low-pass Butterworth filter, reducing the probability of a false detection. This script simply extract these orienting responses by saving into a *csv* the initial timestamp, the duration, magnitude and estimated area (energy) of all orienting responses detected, then the *csv* file is read by Plot results.py script to see results directly on plots.

4.4.8 Plot results.py

This script reads data from created bag and MongoDB, more specifically from the database read GSR and LF/HF, and plot all data that could be used to understand the stress level of the driver and what causes this stress. According to the parameter passed to the program, there are two kind of output: plot of the whole acquisition (*Figure 4.9*) and plots for each road section (*Figure 4.10*). By doing so, is possible to view data with different levels of detail: specifically, in the image is possible to see 4 orienting responses and see where and what the driver looks plus what object is detected in the camera field of view. Moreover, is possible to choose which plot to show,

with a combination of parameters and keyboard input depending on what the user desire to check.

Due to a bug in the Pupil Capture software version used initially in which the whole acquisition was loaded in memory, in case of long acquisition (more than about 5 minutes) the system used all memory resources thus crashing the Operating System. This forced us to stop and resume acquisition during the drive and creating different bag files for a single acquisition so in the final plots is possible to see blank spaces: this is not because we do not have files but we generate different images for different bag as this, in sight of future implementation, means different acquisitions with different subjects. This bug has been solved in newer version of Pupil Capture software by implementing the incremental load, so future acquisition can be as long as we need.

4.4.9 Draw gaze point.py

This script simply draw on the multidarknet output images with bounding boxes a dot in the gaze position computed by the eye tracker: this to show the correct functioning of the system and eventually create a video clip with all processed information. The result of the elaboration is visible in Figure 4.12: the laptop in figure is the one used to collect eye tracker data and is actually ignored when that kind of data is store in rosbag file.

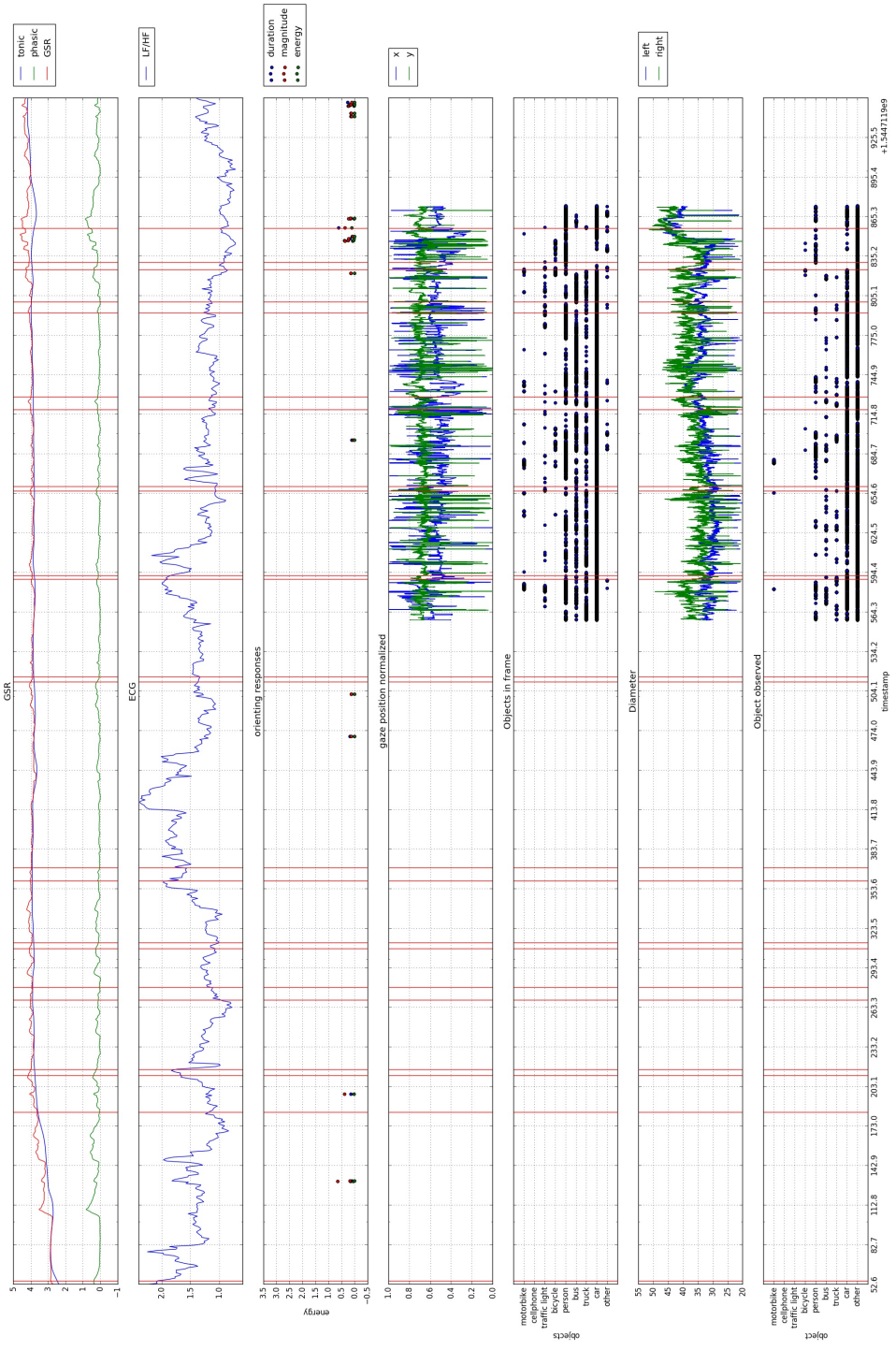


Figure 4.9: This is an example of plot result script output: vertical lines represents different road sections while blank spaces indicates that data are stored in other bag files. The scatter plot representing orienting responses shows possible stressing moments, confirmed by high values in GSR plot and contextualised by the other scatter plots.

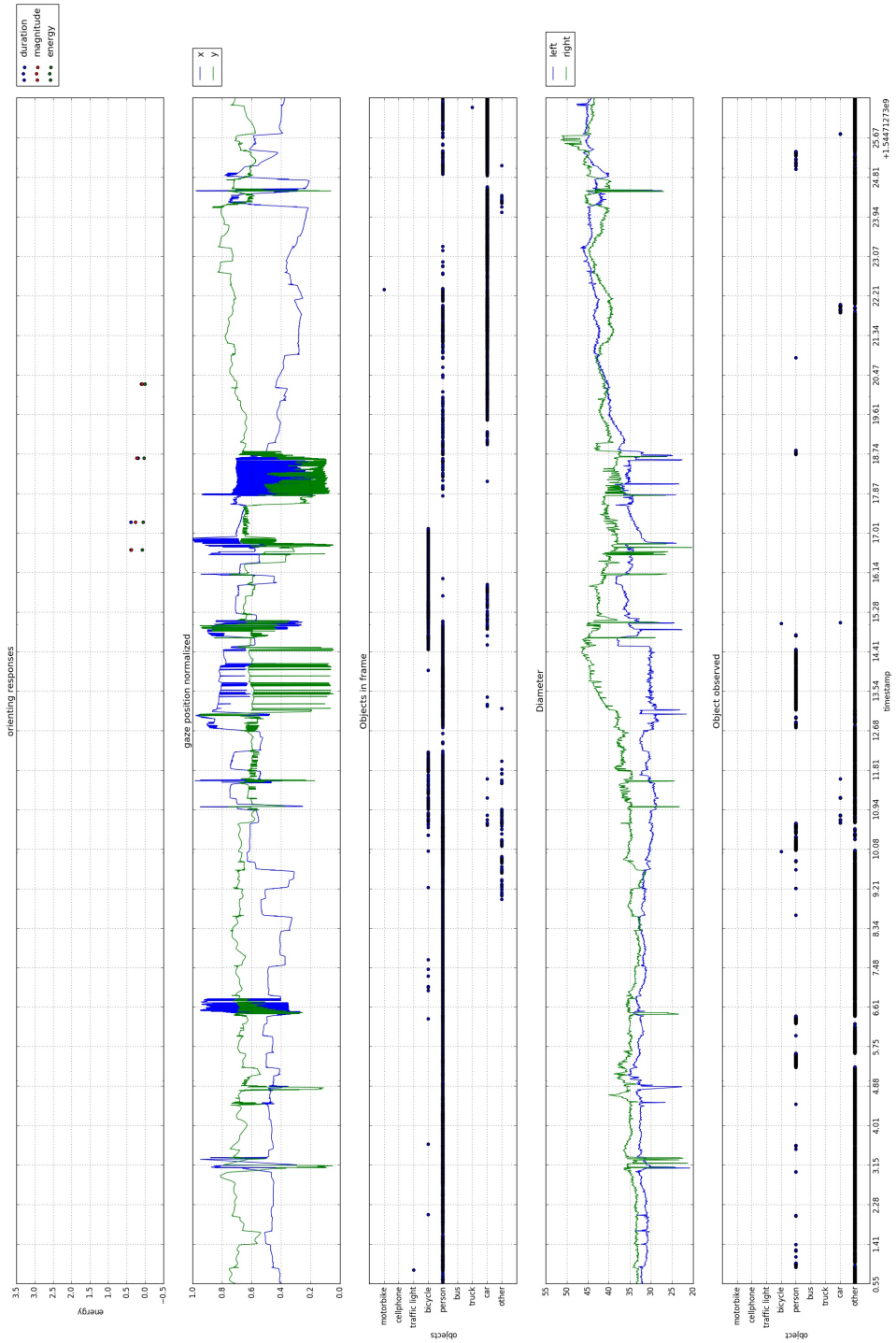


Figure 4.10: This plot shows an example of segmented plot: this gives a better view of a single road section situation including surroundings objects, observed objects and orienting responses.

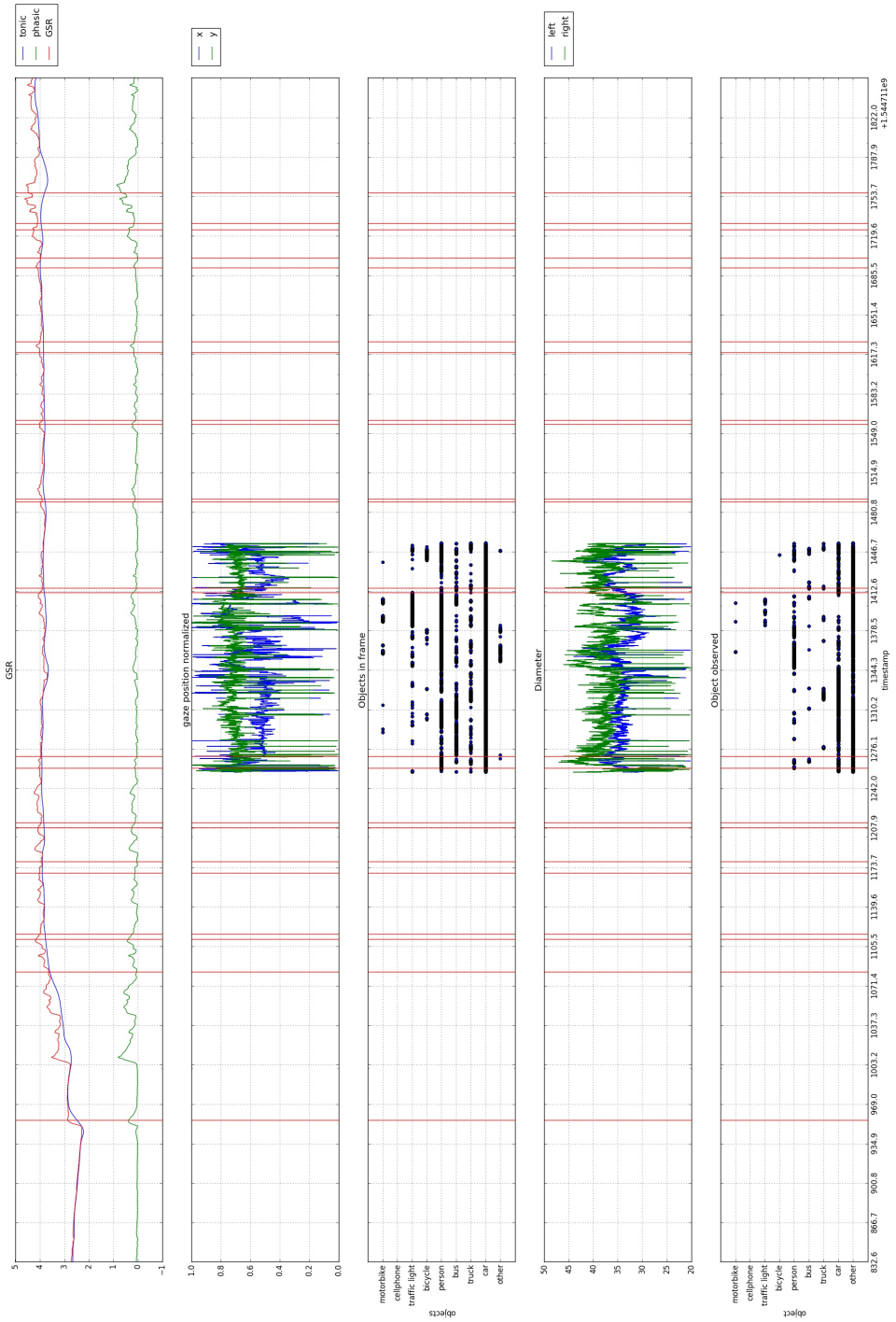


Figure 4.11: Comparing with Figure 4.9 you see that data enhance the content filling some blank spaces without overlaps.

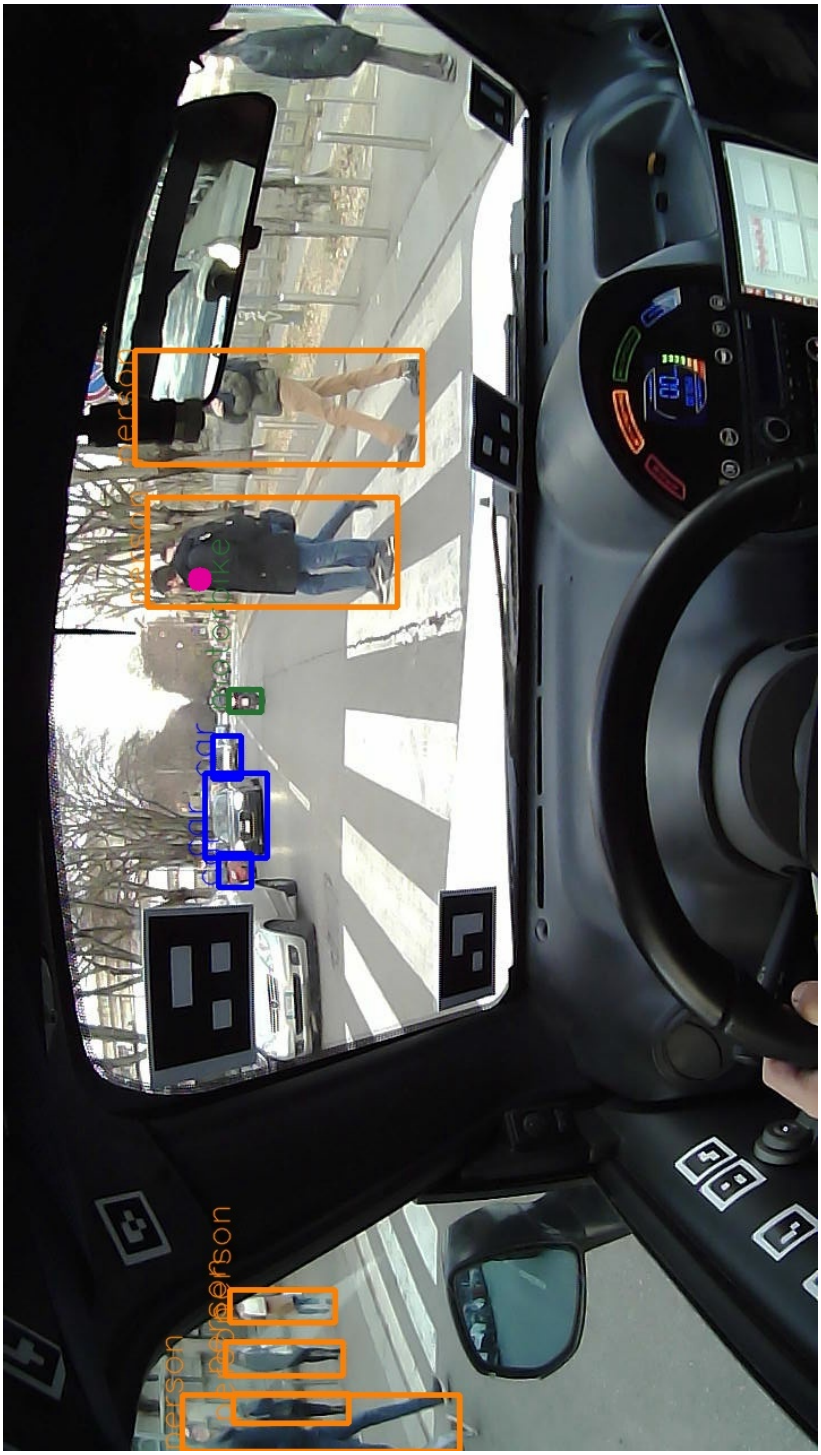


Figure 4.12: In this figure is possible to view the result of the elaboration, giving the idea of the available information that could be used for our elaboration.

Chapter 5

Experiment description and results

In this chapter, we describe the experiment and comment the obtained results also comparing them with the results described in other papers cited previously.

In order to test the developed software, we conducted an experiment in which we aimed basically to test and debug the developed software. Having listed in Section 2.3 some possible physiological signals that evidence stress levels, we tried them to confirm which of them best fits in our urban environment and since we have had a verified stressing event occurred during this experiment, we are able also to verify the reliability of the different stressing indexes.

So our first hypothesis is that HRV feature, LF/HF, is not a reliable stress index during the driving task as stated in Billman (2013) and Shaffer et al. (2014) works: this may clarify how this index behave in case of short observation in urban environment driving and find other possible HRV features that better describes stress level in this specific context.

Pedrotti et al. (2014) in their work was able to spot pupil diameter reaction to stress events but in a simulated environment. We want to verify this behaviour in a real urban environment in order to gain more information thus having a more robust measure of stress: to extract stress level from pupil diameter we have to find the pupil diameter signal component that is independent of the light variation. So our second hypothesis is that by processing the light signal with RLSE to predict the pupil diameter signal, the output error corresponds to the stress component of pupil diameter signal.

Our last hypothesis is that GSR is a reliable index of stress during the driving task as in Healey and Picard (2005) work is used in their experiments

computing orienting responses.

5.1 Experiment

The goal of the acquisition part of this work is to see how people react to multiple stimuli that may happen during the driving task: with this requirement, we have chosen an urban busy road to increase the probability of stressing events to occur and a minimum time of 20 minutes. For our experiment, we had a great limitation due to legal issues concerning car insurance and privacy, that constraint us to limit the number of drivers: in the end, we acquired data from a single driver.

The acquisition we performed had also the goal to develop and test the software system: this acquisition was taken with an old version of Pupil Capture® with a bug that made the computer crash in case of long acquisitions; this bug has been solved in more recent version of the software but to complete the acquisition we split the experiment into smaller recordings to avoid computer memory problems and that's why in Figure 5.1 plots regarding eye tracker related information have blank spaces.

We were lucky that in this acquisition, we encounter a stressing event so we were able to test the software also in that case, setting parameters to detect orienting responses. More recent acquisition are not included in this work due to some error that occurred during the experiment and low functions of the on-board mini-pc.

Additionally, we included road intersection with traffic light and others without them, both situation in which the driver had to turn left or right: in some intersection without traffic lights, there was also a pedestrian crossing trying to increase the workload of the driver (turning and paying attention to the sides of the road). More precisely our experimental path includes five intersections, three of them with traffic lights and a total of seven crosswalks, one of them particularly busy to see the shift of focus in this situation.

Before start driving, we proceed to eye tracker's calibration by following the protocol described in Section 4.1.1, then we enter the car, switch on the on board mini-pc, start the acquisition system without acquiring data to check if all sensors are working properly and finally compute the baseline; then the experiment started.

The driver then drove the car through a quite busy road near Politecnico di Milano, Leonardo campus. With the hypothesis that intersections could stress more than a straight road, we included them as much as possible, performed by turning left and right with also pedestrian crossings: due to the low amount of data acquired, we cannot state that this hypothesis is

not verified since our data do not show evident differences in GSR signal between road sections. After 20 minutes, the driver has to return the car inside the campus: this is a driving particular case so we still acquire data until the car is parked. Finally, we stop recording from Pupil Capture and then from the on-board mini-pc; with a pen-drive, we get the acquired data from the mini-pc and transfer to the laptop used for processing data, then we can start executing scripts.

5.2 Results

As we described in Chapter 4, vertical lines represent road segmentation: due to a lack of available space, adding the name of the road sections on plots cause them to be more confused so to know the data of each road section, the user must execute the script with - -segmented parameter and see the name of the output image. We were lucky to have a verified stressing event with a visible reaction of the driver and near that point, we can see an increasing GSR signal and a presence of orienting responses larger with respect to the rest of the experiment. Concerning LF/HF signal instead, it has a trend that actually is not attributable to a stress level since no orienting response occurs in correspondence of high values of this signal and vice-versa: low values that should indicate low-stress level are instead concentrated in the portion of the graph with many orienting responses and this confirms our first hypothesis that this feature is not reliable for short acquisitions.

About the second hypothesis, we know that the pupil main function is to govern the amount of light that reaches the retina, so in order to extract the stress component from the diameter signal, we elaborate the signal as described in Section 4.2.6 in order to remove the light component from the pupil diameter signal: the correlation index, however, is showing values close to 0 so demonstrating that the two signals are uncorrelated thus refuting our second hypothesis.

In Figures 5.2 and 5.3 we plot the portion of data referring to the entrance to the Politecnico campus that includes the stressing event moment. The first image shows data about GSR, orienting responses and observed object: notice that the first orienting response, that has the largest duration and that correspond to the verified stress event, has low energy though. At that moment as we can see from images, some people suddenly entered the field of view of the driver.

We noticed moreover that the following three orienting responses were related to the driver focus on the street so in Figure 5.3 we show all objects in the scene discovering the presence of bicycles and people. The time elapsed

between the orienting responses is quite short and by looking at the video we see that bicycles are parked and the person that reappears are the same as the first orienting response.

Another interesting point is at the beginning, when the baseline was computed and the subject had to start driving, an orienting response has been detected as can be seen in Figure 5.4. The first increase of GSR value could be interpreted as an orienting response: actually, that increase is pretty slow and was a distraction moment from baseline computation. By looking at the images we see that the orienting response was detected during the manoeuvres so probably stress due to high workload. Having a verified stressing event and an increasing trend of GSR signal near that event shows that between the three, GSR is the most reliable stressing index available.

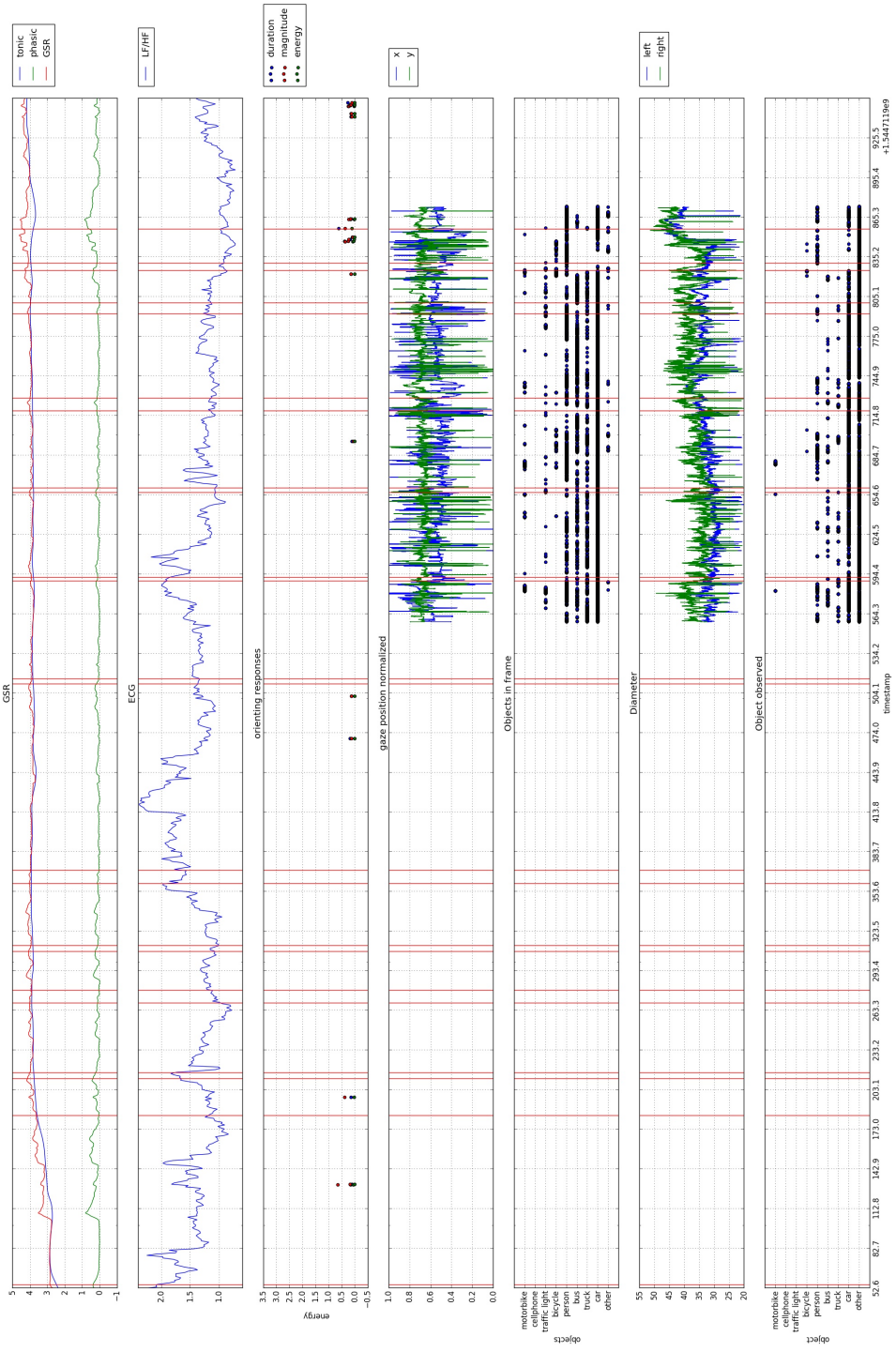


Figure 5.1: In this image is possible to see some important information collected during the driving task and some processed data.

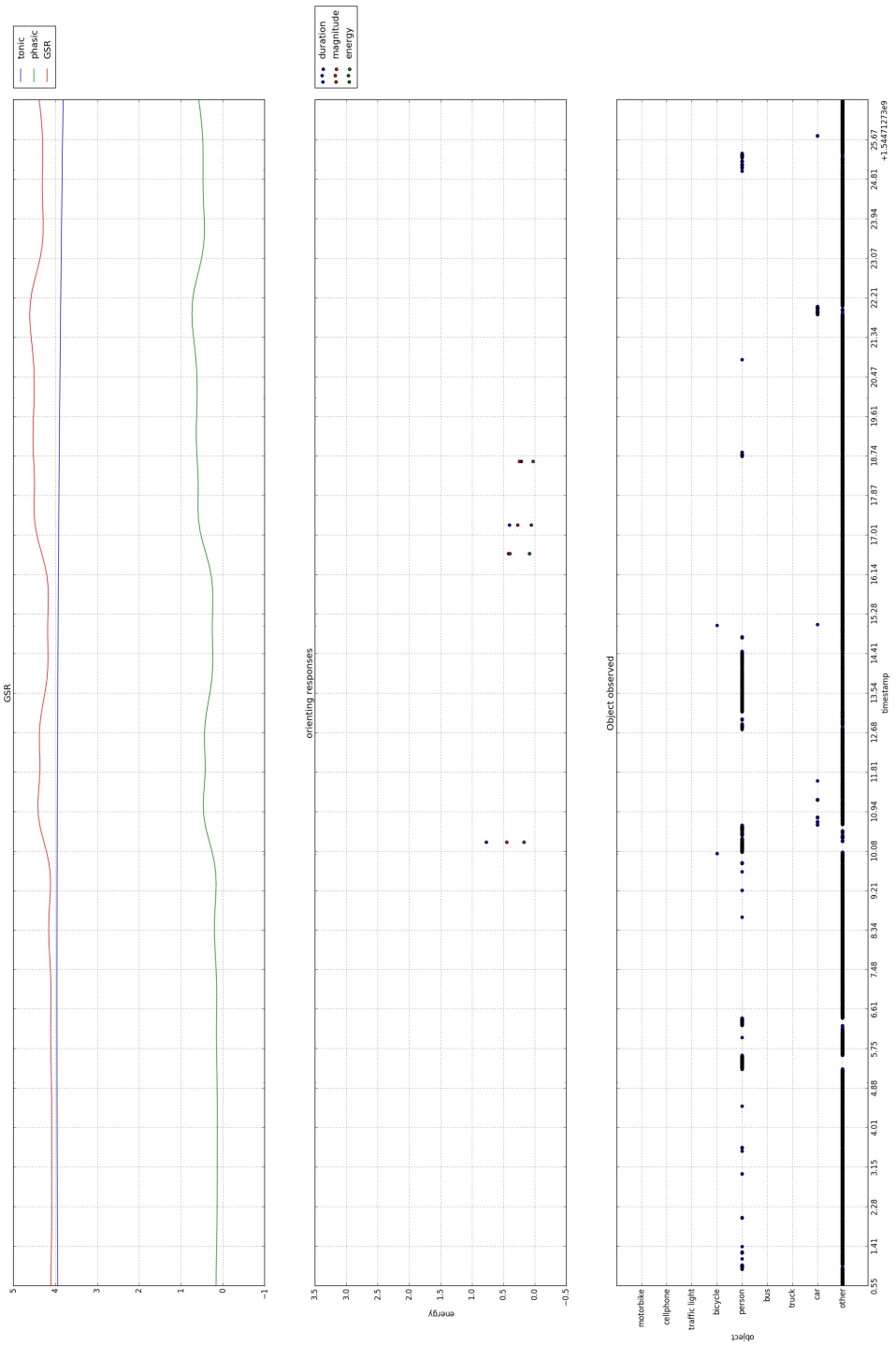


Figure 5.2: This is the output of a single road section, in the specific case was the entrance in university campus showing GSR, orienting responses and the observed object.

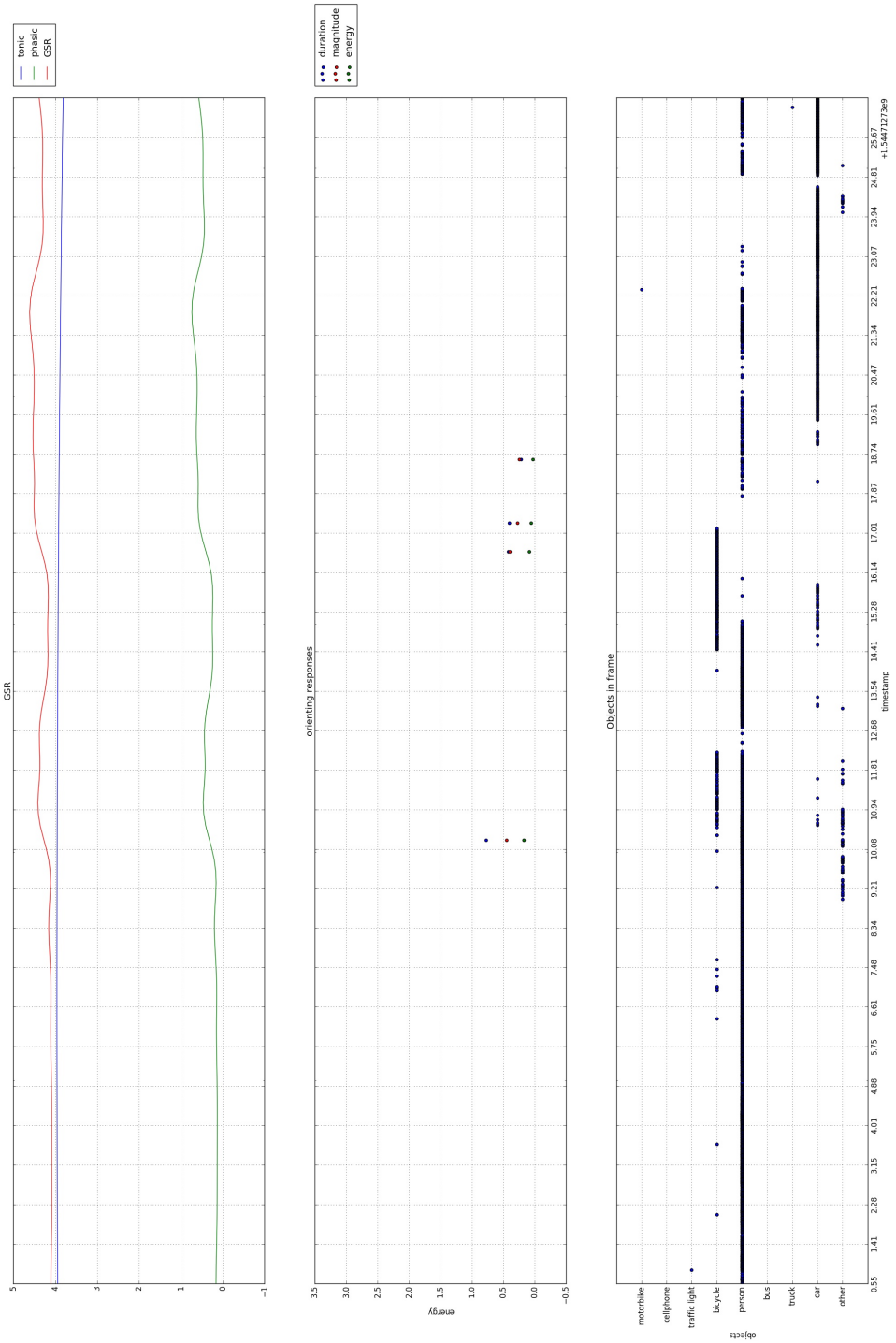


Figure 5.3: In this figure we show GSR tonic and phasic signal, orienting responses and the visible objects in frame while entering the campus.

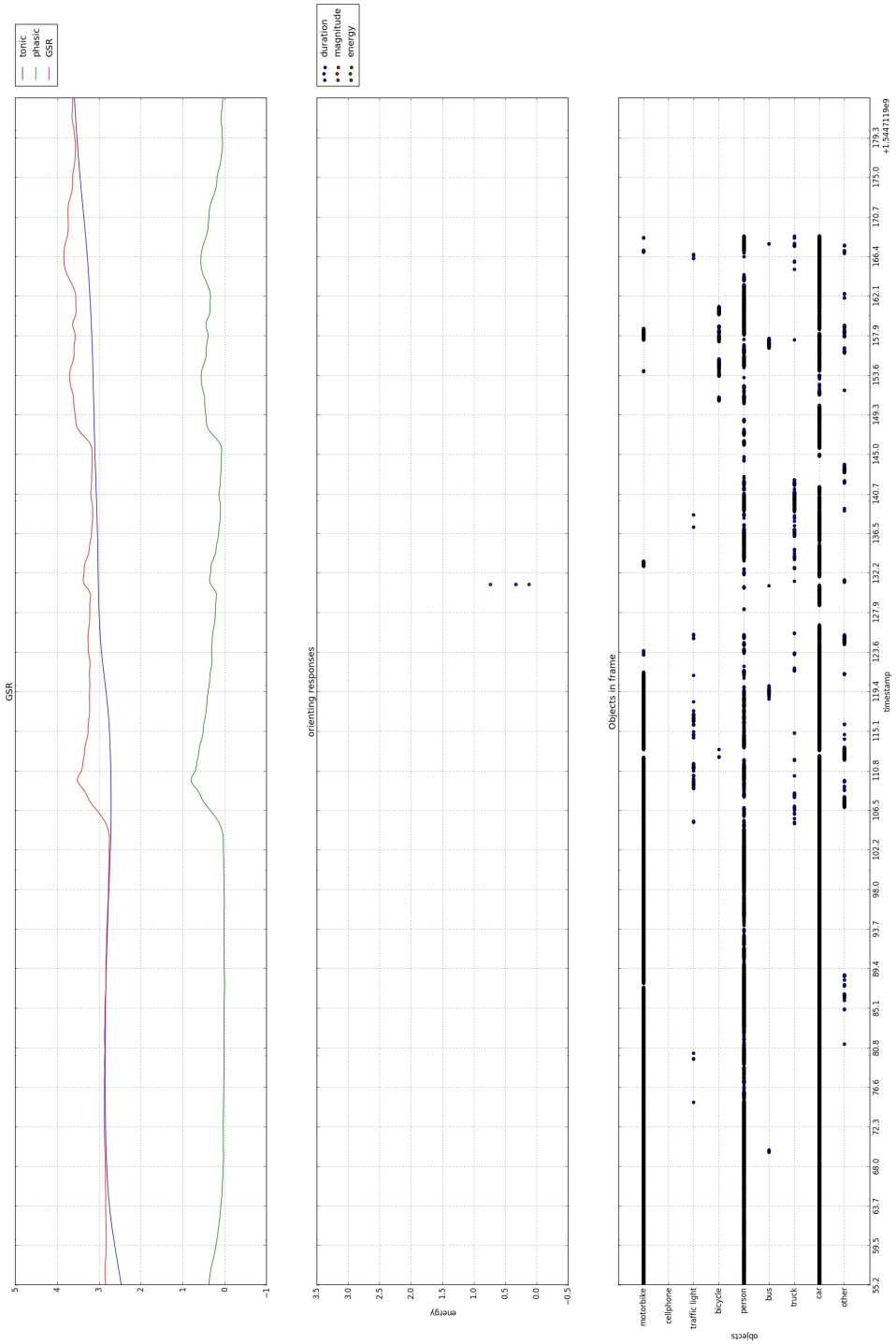


Figure 5.4: This graph concern the first part of the acquisition: the baseline is visible in the first half, then we start manoeuvres to exit the campus and the high workload shows an increase of GSR.

Chapter 6

Conclusions and future works

The developed software has simplified the elaboration task, extracting automatically feature for stress level monitoring: software modularity also allows to reduce execution time in case of data re-elaboration and simplifying also the debug process. Results are stored as images so to see the elaboration results one just has to open images unless you need to see other data displayed and other graphs, then a bit of re-elaboration is required.

Concerning stress indexes, by looking at the results described in the previous chapter, we noticed that driving is, in general, a stressing task, especially in a busy urban environment where unexpected may happen quite frequently so requesting focus and high attention by the driver. In such a dynamic environment, using the pupil diameter as stress index is not as reliable as GSR due to non-static light condition: the elaboration we performed with RLSE was not able to remove noise and correlate pupil diameter with GSR; so pupil diameter was excluded for our analysis at the moment. The same holds for ECG that shows totally different results with respect to GSR: more specifically we used, as some papers in literature cited in Section 2.3, LF/HF feature for sympathetic - parasympathetic nervous system balance but as also stated in some paper cited in that section, is not always reliable as in this case since the acquisition is not long enough.

To spot stressing events, we computed the orienting responses of GSR signal; without suggested thresholds, we computed orienting responses with gradient larger than 0.5 and noticed that near a stressing event, orienting responses are more likely to be visible so in the end, GSR signal shows larger reliability for stress measurement. A headset eye-tracking system improves a lot the available information since we can automatically see where the driver is focused on, specially if we combine eye tracker's data with YOLOv3 elaboration. Due to the low amount of collected data is not possible to

extract more precise information about stress indexes and events: further data acquisition may clarify this aspect by finding a reliable stress index.

6.1 Future Works

Here we list some possible future works that can be useful to increase and improve the information obtained from the acquisition.

Since in an autonomous car we do not have the driver anymore but only passengers, a useful work is to measure stress levels on the passenger, namely a person without control on the car: this study could also give information about the differences of stress level when a person is a driver and when is a passenger. To spot these differences, after the driver had complete the experiment can seat as a passenger still paying attention to the road and wearing sensors. This stress evaluation can be also done in an operative fully autonomous car thus testing if the planner act as desired with manoeuvres that avoid stress to increase.

A way to increase the information gathered from external cameras is to perform image segmentation. Actually, we tried this method but with a lot of misclassification, probably due to network overfitting since with test images the performance was very good while with our acquired images were awful. So by finding a better NN with good performances with the acquired images, it is possible to identify lanes and crosswalks and eventually find new stress conditions like overtakes or sudden crossing pedestrian thus improving our knowledge.

A possible future work, with the goal of simplifying the data acquisition, is to implement a stress classifier. The main idea of this future work is to recognize stress situations in an unobtrusively way: after having collected enough data from physiological sensors to identify many stress conditions, the further step is to recognize these conditions by only analysing images processed with YOLOv3 or external sensors such as LiDAR. The critical part is data acquisition for the supervised learning required to train a NN able to recognize environmental stress conditions: to build a robust classifier we need a lot of data about stressing events.

Due to insurance restrictions, we had a limitation in the number of subjects that could drive the car: it could be interesting to repeat the experiment with people confident in driving the electric car in an urban environment to see if there are more differences between average stress level, and unforeseen event stress reaction. With a lower average stress level it could be possible to spot better peaks of stress levels thus having a more reliable measure.

There are several body reactions to a stressing event one may have and due to the high individuality of this factor, an interesting and useful future work may be to verify if the stressing indexes present in the literature could be used in the real world. In our work, in the end, we used only GSR but we cited other works that say also BVP could be used by exploiting the already mounted camera or respiration rate may add further information combined with other stress indexes: individuality may show some indexes more reliable with respect to the ones we used here and could be useful also to find human features that affect an index to be more reliable.

Once an autonomous car is finally implemented, we could use internal camera to monitor the situation inside the car. This can be a good way to debug and check if the behaviour of the car is improving the user experience or not by seeing their reactions, how much they check the car's actions or if they are able to do different tasks without paying attention to the road. This noninvasive monitoring could measure stress levels also in the in-market autonomous cars so continuously improving this technology.

The used eye-tracking headset has the possibility to acquire images from a different camera beyond the default one installed on the headset: this could increase the quality of information got from the eye tracker since we do not frame the cockpit, we can perform a better image segmentation removing noise and also get gaze point directly on those images without rototranslating gaze point coordinates.

Another possible future work could exploit Deep Learning on point cloud got from LiDAR in order to classify objects surrounding the car with their distances; we can exploit this last information to see if distances also influence stress on the passenger. This will not substitute YOLOv3 elaboration used in this work since it will be impossible to detect what object the driver is focused on but will give more information on the phase of testing the autonomous car behaviour, capable to find a possible critical distance below which the driver feels uncomfortable.

There are a lot of factors that can influence the stress level on the driver, one of these can be how the driver is confident with the car. For this reason, could be useful to give a survey before the experiment in which the subject gives information about its routines trying to understand how much driving our car on a busy road could stress the driver. A post-experiment survey may also be useful to validate acquired data, asking in general how stressing was the overall experiment and if there were some particular stressing situations, verifying then if the software reveals orienting responses on those situations.

With enough computational power, is possible to perform real-time object detection; this future work could help to reduce storage usage. The

idea is to perform YOLOv3 detection while acquiring and store only the information about bounding boxes, having also this information just after the experiment ended. This could also be used for real-time gaze-related stress event detection, so the driving planner will exploit this information to promptly react to reduce driver stress level.

Having a way to monitor throttle-brake trend during the driving task may further improve available information since we can also understand which is the action performed whenever a stress level overcome a threshold or a stressing event occur: this kind of knowledge could simplify the planner development knowing which behaviour should adopt autonomous car in case of danger. However, the autonomous car should not behave human-like because we had already seen in literature, papers stating that the car should drive in a safer way with respect to the usual driver driving style so this is just a general guideline for planner developing.

Bibliography

Christer Ahlstrom and Katja Kircher. A Generalized Method to Extract Visual Time-Sharing Sequences From Naturalistic Driving Data. *IEEE Transactions on Intelligent Transportation Systems*, 18(11):2929–2938, November 2017. ISSN 1524-9050, 1558-0016. doi: 10.1109/TITS.2017.2658945. URL http://ieeexplore.ieee.org/document/7865902/.

Riccardo Barbieri, Ing Luca Cerina, Romano Bruno, Sortino Dario Maria, and Togninalli Francesco. Determinazione dello stato di attenzione alla guida: un approccio contactless. page 57, 2018.

Chandrayee Basu, Qian Yang, David Hungerman, Mukesh Singhal, and Anca D. Dragan. Do You Want Your Autonomous Car To Drive Like You? In *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction - HRI '17*, pages 417–425, Vienna, Austria, 2017. ACM Press. ISBN 978-1-4503-4336-7. doi: 10.1145/2909824.3020250. URL http://dl.acm.org/citation.cfm?doid=2909824.3020250.

Hanna Bellem, Thorben Schönenberg, Josef F. Krems, and Michael Schrauf. Objective metrics of comfort: Developing a driving style for highly automated vehicles. *Transportation Research Part F: Traffic Psychology and Behaviour*, 41:45–54, August 2016. ISSN 13698478. doi: 10.1016/j.trf.2016.05.005. URL https://linkinghub.elsevier.com/retrieve/pii/S136984781630064X.

George E. Billman. The LF/HF ratio does not accurately measure cardiac sympatho-vagal balance. *Frontiers in Physiology*, 4, February 2013. ISSN 1664-042X. doi: 10.3389/fphys.2013.00026. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3576706/.

Moran Cerf, E. Paxon Frady, and Christof Koch. Using semantic content as cues for better scanpath prediction. In *Proceedings of the*

2008 symposium on Eye tracking research & applications - ETRA '08, page 143, Savannah, Georgia, 2008. ACM Press. ISBN 978-1-59593-982-1. doi: 10.1145/1344471.1344508. URL http://portal.acm.org/citation.cfm?doid=1344471.1344508.

Daniel J. Fagnant and Kara Kockelman. Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations. *Transportation Research Part A: Policy and Practice*, 77:167–181, July 2015. ISSN 09658564. doi: 10.1016/j.tra.2015.04.003. URL https://linkinghub.elsevier.com/retrieve/pii/S0965856415000804.

Donald L. Fisher, Anuj K. Pradhan, Alexander Pollatsek, and Michael A. Knodler. Empirical Evaluation of Hazard Anticipation Behaviors in the Field and on Driving Simulator Using Eye Tracker. *Transportation Research Record: Journal of the Transportation Research Board*, 2018(1): 80–86, January 2007. ISSN 0361-1981, 2169-4052. doi: 10.3141/2018-11. URL http://journals.sagepub.com/doi/10.3141/2018-11.

Anna-Katharina Frison and Andreas Riener. Fostering User Acceptance and Trust in Fully Automated Vehicles: Evaluating the Potential of Augmented Reality. page 17, 2018.

R. Girshick, J. Donahue, T. Darrell, and J. Malik. Region-Based Convolutional Networks for Accurate Object Detection and Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(1):142–158, January 2016. ISSN 0162-8828. doi: 10.1109/TPAMI.2015.2437384.

Yanming Guo, Yu Liu, Ard Oerlemans, Songyang Lao, Song Wu, and Michael S. Lew. Deep learning for visual understanding: A review. *Neurocomputing*, 187:27–48, April 2016. ISSN 09252312. doi: 10.1016/j.neucom.2015.09.116. URL https://linkinghub.elsevier.com/retrieve/pii/S0925231215017634.

J.A. Healey and R.W. Picard. Detecting Stress During Real-World Driving Tasks Using Physiological Sensors. *IEEE Transactions on Intelligent Transportation Systems*, 6(2):156–166, June 2005. ISSN 1524-9050. doi: 10.1109/TITS.2005.848368. URL http://ieeexplore.ieee.org/document/1438384/.

Tove Helldin, Göran Falkman, Maria Riveiro, and Staffan Davidsson. Presenting system uncertainty in automotive UIs for supporting trust calibration in autonomous driving. In *Proceedings of the 5th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*

- *AutomotiveUI '13*, pages 210–217, Eindhoven, Netherlands, 2013. ACM Press. ISBN 978-1-4503-2478-6. doi: 10.1145/2516540.2516554. URL http://dl.acm.org/citation.cfm?doid=2516540.2516554.

Moritz Kassner, William Patera, and Andreas Bulling. Pupil: An Open Source Platform for Pervasive Eye Tracking and Mobile Gaze-based Interaction. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*, UbiComp '14 Adjunct, pages 1151–1160, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-3047-3. doi: 10.1145/2638728.2641695. URL http://doi.acm.org/10.1145/2638728.2641695.

Kanwaldeep Kaur and Giselle Rampersad. Trust in driverless cars: Investigating key factors influencing the adoption of driverless cars. *Journal of Engineering and Technology Management*, 48:87–96, April 2018. ISSN 09234748. doi: 10.1016/j.jengtecman.2018.04.006. URL https://linkinghub.elsevier.com/retrieve/pii/S0923474817304253.

Jeamin Koo, Jungsuk Kwac, Wendy Ju, Martin Steinert, Larry Leifer, and Clifford Nass. Why did my car just do that? Explaining semi-autonomous driving actions to improve driver understanding, trust, and performance. *International Journal on Interactive Design and Manufacturing (IJIDeM)*, 9(4):269–275, November 2015. ISSN 1955-2513, 1955-2505. doi: 10.1007/s12008-014-0227-2. URL http://link.springer.com/10.1007/s12008-014-0227-2.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

Jesse Levinson, Jake Askeland, Jan Becker, Jennifer Dolson, David Held, Soeren Kammel, J. Zico Kolter, Dirk Langer, Oliver Pink, Vaughan Pratt, Michael Sokolsky, Ganymed Stanek, David Stavens, Alex Teichman, Moritz Werling, and Sebastian Thrun. Towards fully autonomous driving: Systems and algorithms. In *2011 IEEE Intelligent Vehicles Symposium (IV)*, pages 163–168, Baden-Baden, Germany, June 2011. IEEE. ISBN 978-1-4577-0890-9. doi: 10.1109/IVS.2011.5940562. URL http://ieeexplore.ieee.org/document/5940562/.

Matteo Matteucci and Alessandro Gabrielli. I.DRIVE: Progetto e sviluppo di un data logger multisensoriale per studi di interazione veicolo conducente. page 114, 2015.

Matteo Matteucci, Giulio Fontana, and Iacopo Galimberti. I.DRIVE: Elaborazione e analisi di segnali siologici per la valutazione dello stress alla guida. page 136, 2016.

Marco Pedrotti, Mohammad Ali Mirzaei, Adrien Tedesco, Jean-Rémy Chardonnet, Frédéric Mérienne, Simone Benedetto, and Thierry Baccino. Automatic Stress Classification With Pupil Diameter Analysis. *International Journal of Human–Computer Interaction*, 30(3):220–236, March 2014. ISSN 1044-7318. doi: 10.1080/10447318.2013.848320. URL `https://doi.org/10.1080/10447318.2013.848320`.

Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, Las Vegas, NV, USA, June 2016. IEEE. ISBN 978-1-4673-8851-1. doi: 10.1109/CVPR.2016.91. URL `http://ieeexplore.ieee.org/document/7780460/`.

Svenja Scherer, André Dettmann, Franziska Hartwich, Timo Pech, Angelika C Bullinger, and Gerd Wanielik. How the driver wants to be driven - Modelling driving styles in highly automated driving. page 6, 2015.

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, January 2015. ISSN 08936080. doi: 10.1016/j.neunet.2014.09.003. URL `https://linkinghub.elsevier.com/retrieve/pii/S0893608014002135`.

Fred Shaffer, Rollin McCraty, and Christopher L. Zerr. A healthy heart is not a metronome: an integrative review of the heart's anatomy and heart rate variability. *Frontiers in Psychology*, 5, September 2014. ISSN 1664-1078. doi: 10.3389/fpsyg.2014.01040. URL `http://journal.frontiersin.org/article/10.3389/fpsyg.2014.01040/abstract`.

Hardeep Singh, J. S. Bhatia, and Jasbir Kaur. Eye tracking based driver fatigue monitoring and warning system. In *India International Conference on Power Electronics 2010 (IICPE2010)*, pages 1–6, New Delhi, India, January 2011. IEEE. ISBN 978-1-4244-7883-5. doi: 10.1109/IICPE.2011.5728062. URL `http://ieeexplore.ieee.org/document/5728062/`.

Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, M. N. Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, Michele Gittleman, Sam Harbaugh, Martial Hebert, Thomas M. Howard, Sascha Kolski, Alonzo Kelly, Maxim Likhachev, Matt McNaughton, Nick Miller, Kevin Peterson, Brian Pilnick, Raj Rajkumar, Paul Rybski, Bryan Salesky, Young-Woo Seo, Sanjiv Singh, Jarrod Snider, Anthony Stentz, William "Red" Whittaker, Ziv Wolkowicki, Jason Ziglar, Hong Bae, Thomas Brown, Daniel Demitrish, Bakhtiar Litkouhi, Jim Nickolaou, Varsha Sadekar, Wende Zhang, Joshua Struble, Michael Taylor, Michael Darms, and Dave Ferguson. Autonomous driving in urban environments: Boss and the Urban Challenge. *Journal of Field Robotics*, 25(8): 425–466, August 2008. ISSN 15564959, 15564967. doi: 10.1002/rob.20255. URL `http://doi.wiley.com/10.1002/rob.20255`.

María Viqueira Villarejo, Begoña García Zapirain, and Amaia Méndez Zorrilla. A Stress Sensor Based on Galvanic Skin Response (GSR) Controlled by ZigBee. *Sensors*, 12(5):6075–6101, May 2012. ISSN 1424-8220. doi: 10.3390/s120506075. URL `http://www.mdpi.com/1424-8220/12/5/6075`.

Adam Waytz, Joy Heafner, and Nicholas Epley. The mind in the machine: Anthropomorphism increases trust in an autonomous vehicle. *Journal of Experimental Social Psychology*, 52:113–117, May 2014. ISSN 00221031. doi: 10.1016/j.jesp.2014.01.005. URL `https://linkinghub.elsevier.com/retrieve/pii/S0022103114000067`.

H. Weisser, P.J. Schulenberg, H. Gollinger, and T. Michler. Autonomous driving on vehicle test tracks: overview, implementation and vehicle diagnosis. In *Proceedings 199 IEEE/IEEJ/JSAI International Conference on Intelligent Transportation Systems (Cat. No.99TH8383)*, pages 62–67, Tokyo, Japan, 1999. IEEE. ISBN 978-0-7803-4975-9. doi: 10.1109/ITSC.1999.821028. URL `http://ieeexplore.ieee.org/document/821028/`.

Zutao Zhang and Jia-shu Zhang. Driver Fatigue Detection Based Intelligent Vehicle Control. In *18th International Conference on Pattern Recognition (ICPR'06)*, pages 1262–1265, Hong Kong, China, 2006. IEEE. ISBN 978-0-7695-2521-1. doi: 10.1109/ICPR.2006.462. URL `http://ieeexplore.ieee.org/document/1699439/`.

Appendix A

Software installation

Below a detailed description of the software system, including all third-party Python modules.

A.1 Requirements

To execute the script correctly, we need the following software requirements to be satisfied:

- Ubuntu Linux operating system version 16.04 LTS
- ROS framework *Kinetic* version
- Python 2.7 version (compatible with ROS version)
- Pupil Labs® software available at **https://github.com/pupil-labs/pupil/releases/tag/v1.11**
- YOLOv3 available at **https://pjreddie.com/darknet/yolo**

A.2 Python modules

Below all the non-standard Python v2.7 modules that had to be installed: modules concerning rosbag and YOLOv3 elaboration are automatically installed when installing YOLOv3 and ROS:

- *Pymongo* for MongoDB elaborations
- *OpenCV* v3.4.3 for image elaboration
- *Pandas* for data elaboration through data frames

- *Mathplotlib* for Matlab-like plot methods
- *Scipy* for *mat* file import and data filtering
- *Padasip* and *Adaptfilt* for data filtering and elaboration
- *tzlocal* for timestamp synchronization with GPS data

All above modules were installed using command:

pip install *module-name*

A.3 Custom ROS messages

In this section we will describe ROS message file creation, used to store information in rosbag files. Basically, message files contain variables or structures of them: the most common structure is the Header of a message, a built-in structure used to store information about the message itself like timestamp or the id of the message; a list of the type of variable can be found at **http://wiki.ros.org/msg**. Concerning the messages containing the acquired video we used a function of OpenCV module that convert a single frame object into a ROS Image message, so without creating customized message. All other information needed a customized message, made up of the Header part and the data part: the messages containing YOLOv3 information contained also an enumerated type variable, to ensure the sub-classes of detected objects. These *msg* files are located in the ROS workspace:

Home > catkin_ws > src > building_bag > msg

The name "building_bag" is mandatory because to import messages structure into python modules we refer to that directory: the final step is to compile ROS with the new messages by running in the catkin workspace the command **catkin_make**. Remember to put into *.bashrc* the command

source /catkin_ws/devel/setup.bash

In this way the script will be able to retrieve custom messages.

```
Header header
uint8 OTHER=0
uint8 CAR=1
uint8 TRUCK=2
uint8 BUS=3
uint8 PERSON=4
uint8 BICYCLE=5
uint8 TRAFFIC_LIGHT=6
uint8 CELLPHONE=7
uint8 MOTORBIKE=8
uint8 object
float64 confidence
float64 relative_center_x
float64 relative_center_y
float64 relative_width
float64 relative_height
```

Figure A.1: Yolo message: The variables with assigned values implements the enumeration.

```
Header header
string eye
float64 confidence
float64 diameter
float64 model_confidence
float64 diameter_3d
float64 norm_pos_x
float64 norm_pos_y
```

Figure A.2: Eyes message: variable "eye" indicate which eye the message refer.

A.3.1 Custom messages content

- **yolo.msg (Figure A.1):** This file contains the information about YOLOv3 detection so the 9 classes, the detection confidence, center, width and height of the bounding box.
- **eyes.msg (Figure A.2):** This file stores information about each eye, the diameter of the pupil in pixel and millimetres and the coordinates of pupil position.
- **blinks.msg (Figure A.3):** Here we store the beginning and final timestamp and frame of the detected blink.
- **fixation_on_surf.msg (Figure A.4):** Contains the start and end frames, the positions about fixation on a surface and the name of the surface.
- **gaze_on_surf.msg (Figure A.5):** Store information about gaze coordinates with respect to a surface: the "on_surface" variable con-

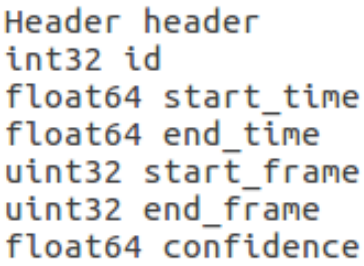


Figure A.3: Blinks message with start and end timestamp and frame.

```
Header header
int32 id
string surface
float64 duration
int32 start_frame
int32 end_frame
geometry_msgs/Pose2D norm_pos
geometry_msgs/Pose2D pos_scaled
string on_surface
```

Figure A.4: In this message we exploit the built-in structure Pose2D for storing fixation position.

```
Header header
float64 gaze_time
geometry_msgs/Pose2D gaze_pos
geometry_msgs/Pose2D gaze_pos_scaled
string surface
string on_surface
float64 confidence
```

Figure A.5: Gaze on surface message.

```
Header header
int32 index
float64 confidence
float64 norm_pos_x
float64 norm_pos_y
geometry_msgs/Point gaze_point_3D
geometry_msgs/Point right_eye_center
geometry_msgs/Point right_eye_gaze_normal
geometry_msgs/Point left_eye_center
geometry_msgs/Point left_eye_gaze_normal
```

Figure A.6: Gaze position message.

tains "true" or "false" values; the choice of string and not a Boolean was for simplicity of implementation because the value in *csv* file is stored as a string.

- **gaze_pos.msg (Figure A.6):** Store information about gaze position in the world frame: for better visibility, we exploit 3D points to store all gaze position information.

- **surface_position.msg (figureA.7):** Contains the number of iden-

```
Header header
string surface
geometry_msgs/Point[3] translation_to_screen
geometry_msgs/Point[3] translation_from_screen
uint8 num_of_markers
```

Figure A.7: Surface position message.

```
Header header
string surface
uint8 OTHER=0
uint8 CAR=1
uint8 TRUCK=2
uint8 BUS=3
uint8 PERSON=4
uint8 BICYCLE=5
uint8 TRAFFIC_LIGHT=6
uint8 CELLPHONE=7
uint8 object
string event
```

Figure A.8: Surface event message.

tified markers for a surface and the roto-translation matrix from the screen reference system to the surface and vice-versa: since matrices are not allowed in ros messages, we built an array of 3D point of dimension 3 since we need a 3x3 matrix.

- **surface_event.msg (Figure A.8):** Store information on when the gaze enters in a surface and what detected object is observing. The event state if the gaze was entering or exiting the corresponding surface: adding the enumeration variable we can match the object and the surface, hence object and gaze position with respect to the car orientation.

A.4 Python scripts download

All scripts are stored online on bitbucket and can be downloaded after registration and login with the command:

git clone
https://username@bitbucket.org/airlab-polimi/idrive_ros.git

By default, Pupil Capture software store recordings in the directory with the following structure:

*Home > recordings > *date_ of_ recordings* > *three digits counter**

After download, scripts must be ran under the "recordings" directory, otherwise will not be able to retrieve the required files for elaboration.

A.5 YOLOv3 setup

Although YOLOv3 has a website with a dedicated guide for installation, still there are some operations to consider in order to make this framework work properly. First of all, you have to download the proper weight file and store in a "weight" folder inside "darknet" directory; then if the script rises the exception "Cuda error: memory out of bound" you may have to change a bit the configuration. To do that, inside "*darknet/cfg*" you can find "yolov3.cfg" file; modify it by changing the first recurrence of variable "subdivisions" from 16 to 32 (or 64 if the error rises again). Last modification we had to do was the python "*darknet.py*" script: once open, user must paste:

from os.path import expanduser
home = os.expanduser("∼")

thus defining variable home as path to home directory; then find and uncomment line:

lib = CDLL(home+"/darknet/libdarknet.so", RTLD_GLOBAL)

and comment the next line; that command should retrieve "libdarknet.so" file so in case of different location, the user should place the correct path. After this, YOLOv3 should work without problems.

A.6 MongoDB installation

In this section we describe all steps to correctly install MongoDB for the correct execution of the scripts:

- Download MongoDB community edition by following instruction in the page. **https://docs.mongodb.com/manual/tutorial/install-mongodb-on-ubuntu/**
- Create folder with read permission by using command **sudo chown -r mongodb:mongodb *folder path***
- Modify **/etc/mongodb.conf** by open it with root privilege and change **dbpath** with the previous created folder.

- Restart MongoDB with command **sudo service mongodb restart**

Now it is possible to import data acquired from the sensors into MongoDB.

Appendix B

Scripts documentation

In this section, we describe in detail the developed software. The whole software system is developed with the idea of maintaining good modularity: the main reason is that this allows the user to resume pipeline execution in case of error or to redo partial execution in case of new data processing, avoiding to execute the whole pipeline thus reducing execution time.

For each script we describe the main function, required input, the implemented function, the output of the script and the time complexity; finally, we describe the possible parameters that could be passed and which is the effect on the execution.

B.1 Data elaboration.py

This is the first script that has to be executed after the acquisition ended. Its main function is to extract frames from the videos acquired by the three eye tracker's cameras, so make sure that all videos are correctly recorded. To increase automation of this script still maintaining a good level of modularity, is possible, according to the given parameters (details in next section), to execute other scripts in the processing pipeline.

This video extraction is an important part because both YOLOv3 and rosbag files take as input images and not videos. To perform this frame extraction we used an available function in OpenCV that reads the whole acquired video and then iterate it to extract all frames that are then stored: so the time complexity of this algorithm is linear in the length of the eye tracker's cameras video.

B.1.1 Parameters

- **path: required**
  Recording date in the format YYYY_MM_DD corresponding also to the recording directory.

- **vals: required**
  In the case of multiple acquisitions on the same date is possible to select which recording to process; two integers are requested indicating the first and the last acquisition to process. If only one acquisition has been done, just pass "1 1".

- **output: required**
  Destination folder in which store rosbag files.

- **-f: optional**
  Enable the frame extraction functionality; mandatory the first time you run this script to generate input files for the other scripts.

- **-y: optional**
  Enable YOLOv3 detection by running "mutidarkent.py" script.

- **-b: optional**
  Enable *bag* file creation by running "CSVtoBAG.py" script.

- **-l: optional**
  Enable light amount estimation by running "Light.py" script.

- **-t: optional**
  Enable timestamp shift by running "timestamps.py" script (Section B.2).

- **-debug: optional**
  Parameter passed to "multidarknet.py" script that save images with printed bounding boxes on detected objects.

B.2 timestamp.py

This script computes Unix time shift for blinks, pupil positions, and gaze positions by reading the info.csv file of a recording that contains initial timestamp in Unix format: to avoid problems that could derive from overwriting existing *csv* files, the script create new file with the same name of the original but suffixing "_aligned" to distinguish the new file. The main purpose of

this elaboration is to plot graphs of gaze position and pupil diameter aligned with GSR graph; Knowing the files name and timestamp column position, we decided to hardcode those information in the script. In case one needs to compute Unix time shift on other *csv* files, it can just call the internal function with the name of the file without its extension and the timestamp column position.

This script changes the timestamp of pupil and gaze position so time complexity is linear in the rows of those *csv* files that also correspond to the number of eye frames, hence is linear in the eye frames.

B.2.1 Parameters

- **path: required**
  Recording date in the format YYYY_MM_DD corresponding also to the recording directory.

- **vals: required**
  In the case of multiple acquisitions on the same date is possible to select which recording to process; two integers are requested indicating the first and the last acquisition to process. If only one acquisition has been done, just pass "1 1".

B.3 Light.py

This script estimates the light amount by computing the average RGB value in some hardcoded ROI chosen according to the eye image: since the frame rate of the eye images is double with respect to the world image, the script performs the average between pairs of images getting a 1:1 correspondence with world image hence a light estimation for each world frame. For different acquisitions with different subjects, chosen ROIs could not be suitable and should be adjusted. The result of the computation are stored in "*recording date > Output > Light_ *num of acquisition**". Trivially, this script is linear in the number of eye frames since the ROIs are fixed so independently from the input amount.

B.3.1 Parameters

- **path: required**
  Recording date in the format YYYY_MM_DD corresponding also to the recording directory.

- **vals: required**
  In the case of multiple acquisitions on the same date is possible to select which recording to process; two integers are requested indicating the first and the last acquisition to process. If only one acquisition has been done, just pass "1 1".

- **- -debug: optional**
  Store images in a sub-folder "r_ROI" or "l_ROI" according to the processed eye and shows eye frames with the relative ROIs so to check if can fit to the subject or must be changed.

B.4 Correlation.py

The role of this script is to compute the PCC in order to understand which physiological features could be used as stress index; our hypothesis is that GSR is the most reliable so we look for other features with high correlation with it.

At the first release of this script, the computation was not optimized hence the execution time was pretty long: to overcome this problem we implemented the script such that at each execution checks if there are stored results, otherwise starts a new computation and after the most work-intensive part, it stores partial results in:

YYYY_MM_DD/Output/Correlation

In that location you can find *csv* files named *"[Left/Right]_aligned_GSR *acquisition number*.csv"* so further executions of the script will read directly those files thus reducing execution time: in case one needs to recompute partial results, files must be deleted manually. In a second time we optimized this script reducing considerably the execution time of the first part but maintaining the partial result storage for future works that could focus on those data computing something more than simply correlation.

Going into details, the first part reads data from "tonic_phasic.mat" file, containing GSR row data, tonic and phasic features and "pupil_positions _aligned.csv" file, reading timestamp, confidence and diameter in millimeters. Then it stores all of this into a pandas data frame, join it with light estimation of both eyes reading values from files, compute oversampling of GSR by interpolating it thus having the same amount of data and finally stores data.

The second part filters GSR, pupil diameter and light estimation data and compute PCC: we tried different filtering methods, left as comments in

the script in case one want to try them. Methods are moving average and Butterworth filter, with low-pass or high-pass configuration depending on the computation to perform; after this first operation have been applied we tried to extract stress features from eyes pupil diameter by filtering signal with NLMSE and RLSE and compare those results with GSR features. Finally results concerning PCC are print on terminal while plots about diameter, light and GSR, RLSE are shown.

Concerning time complexity, since we have to compute interpolation in order to guarantee the same amount of data between ones collected from eye tracker and GSR electrodes, we iterate among GSR data, interpolate between 2 points and compute new data for each timestamp of the eye tracker data. So the overall complexity is the GSR amount, times the ratio between eye frames and GSR; so linear in the amount of eye frames.

B.4.1 Parameters

- **path: required**
  Recording date in the format YYYY_MM_DD corresponding also to the recording directory.
- **vals: required**
  In the case of multiple acquisitions on the same date is possible to select which recording to process; two integers are requested indicating the first and the last acquisition to process. If only one acquisition has been done, just pass "1 1".
- **time: required**
  Accept values “mean”, “begin” or “end” and is designed for continuous stress level analysis: in our case was used to determine if LF/HF was a proper stress index but the problem was to align it correctly with GSR since computed after about 2 minutes of acquired data. This parameter shift the signal at beginning, end or the centre of this time interval, computed at run-time so adapting to different time intervals.
- **filter_type: required**
  Select the type of applied filtering: “low” for low pass filtering, “high” for high pass filtering; passing “no” avoids filtering of data.

B.5 multidarkent.py

This script computes YOLOv3 detection by importing the functions from module “darknet.py” installed together with YOLOv3: after load network

architecture and weights and classification data, we run the detecting function storing the results in a *txt* file in which each row represent different detected object with related confidence, dimension and centre coordinates of the bounding box. In case of -d parameter, a directory "frame_debug" is created storing all resulting frames with the drawn bounding boxes: to compute the position and coordinates of bounding boxes, you have to run this script without the - -no_yolo parameter to generate *txt* file with all required information; this to avoid recompute object detection every time.

YOLOv3 detect method allow also to choose in case of multiple GPUs, in which of them to load the CNN and execute: simply we have to give as parameter the index of the GPU we want to use ('0' in case of a single graphic card as in our case); this index is passed as parameter and can be retrieved by running on terminal the command:

nvidia-smi

Each object detection is performed individually on each frame, so the overall complexity depends on the amount of world camera frames.

B.5.1 Parameters

- **directory: required**
  Working directory in the format *YYYY_MM_DD/*acquisition number**.
- **GPU: required**
  The GPU index you want to use for YOLOv3 detection.
- **-d: optional**
  enable the generation of images with objects bounding boxes drawn on them; available only if *txt* files of detected objects are generated.
- **- -no_yolo: optional**
  Avoid execution of YOLOv3 allowing to create images without recomputing object detection; this parameter implies the presence of *-d* parameter otherwise the script terminates immediately.

B.6 draw_gaze_point.py

This simple script reads YOLOv3 processed images with bounding boxes drawn and plot a fuchsia circle in the gaze position read from the *gaze_position.csv* file. The resulting images could be used for a presentation video showing the

available information and to check eventually correctness of processed data. Although the elaboration modifies world camera frames, gaze position is based on eye data so the complexity of this script is linear in the eye frames.

B.6.1 Parameters

- **path: required**
  Recording date in the format YYYY_MM_DD corresponding also to the recording directory.

- **vals: required**
  In the case of multiple acquisitions on the same date is possible to select which recording to process; two integers are requested indicating the first and the last acquisition to process. If only one acquisition has been done, just pass "1 1".

B.7 CSVtoBAG.py

This script takes as input the frames acquired from the eye tracker's cameras plus all *csv* files exported from Pupil Player software, computing automatically time alignment. According to the processed information, a ROS message is created, assigning a topic and the values read from the *csv* file: these values are selected basing on our work, in case for future work there is the need of process different data, one should change ROS message files and how this script process them. Moreover file positions are hardcoded, assuming that only one data exportation from Pupil Player software is performed and that the user do not modify files: in case of multiple exports, the script processes only export number "000" so need to rename folders for a correct execution.

In case the bag file was already created, the script waits for user input to overwrite the existing bag file or not; in case of negative response, the script ends the execution immediately.

The time complexity depends on the amount of collected data and the largest amount is collected by the eye frames so time complexity is linear (but with very large constant) on eye frames.

B.7.1 Parameters

- **date: mandatory**
  The directory that contains data we want to store in rosbag file, in the form of *recording_date*/00*number of recording*: this format is the

result of the intention of parallelizing the rosbag file creation so many recordings in the same day could be executed in parallel to save time.

- **work_dir: mandatory**
  The name of the output rosbag file, in recording_date_*number of acquisition*.

- **output_dir: mandatory**
  Destination directory to store rosbag files; we choose to use this parameter since rosbag are often large file so user can select the proper file location.

B.8 orienting_responses.py

This script computes orienting responses features of GSR; as previously mentioned , the orienting response is a sudden increase of skin conductance, easily attributable to a stressing event and they extracted four features:

- Number of orienting responses
- Sum of duration $\sum \mathrm{O_D}$
- Sum of magnitude $\sum \mathrm{O_M}$
- Sum of estimated area $\sum \frac{1}{2}\mathrm{O_D} * \mathrm{O_M}$

These features are computed within a time range so by dividing the acquisition into regular time intervals, is possible to compare these features and spot the stressing events.

In Healey and Picard (2005) actually they did not specified parameters about the computation of the orienting responses such as minimum slope or duration to consider an orienting response as valid so we had some trials in order to find the combination that shows orienting responses at least when we were sure about a stressing event: the final threshold we used concerned slope greater than 0.5. The slope was simply computed with linear interpolation between pairs of GSR points, then we compute the gradient of the line linking these two points.

Finally we filter results by excluding all orienting responses with a magnitude lower than 0.2; we tried first to filter result basing on energy but in that case, quick reactions were excluded from the final results, even with high magnitude. Once data have been computed, *orienting_responses.csv* file with the result is created in the directory specified by parameters, and a plot with GSR and the estimated area of all orienting responses is displayed:

this feature shows high readability for stressing moment identification, having relatively large values near the well-known stressing event. Input file *Skin conductance.csv*, got from the sensor, is retrieved by the parameters so must be placed in the correct directory *"YYYY_MM_DD/00*acquisition number*"*. Time complexity of this script depends only on the dimension of GSR data.

B.8.1 Parameters

- **path: required**
  Recording date in the format YYYY_MM_DD corresponding also to the recording directory.

- **vals: required**
  In the case of multiple acquisitions on the same date is possible to select which recording to process; two integers are requested indicating the first and the last acquisition to process. If only one acquisition has been done, just pass "1 1".

- **- -show: optional**
  This parameter shows plots about GSR and the orienting responses for debug purpose in order to find the slope that best describes trend of signal.

B.9 plot_bag.py

This is the script that have to be executed at last, since is developed to read data from different *csv* files, rosbag files and *mat* files and save results into images; file locations must correspond to the acquisition one so:

*YYYY_MM_DD/00*acquisition number**

As mentioned before, *bag* files are stored in a different location so it must be passed as parameter to correctly retrive them. Moreover this script implements a function that reads from "gps_positions.csv" the coordinates of the beginning of each road section, this allow to draw on plots vertical lines in the exact moment in which the road section changes, obtaining path segmentation. Since the number of graphs to show can be customized, the worst case is when the user wants to plot all graphs, at most 7: in that case the complexity is still linear in the input data and more precisely on *bag* file amount of messages.

B.9.1 Parameters

- **rec_date: required**
  Recording date in the format YYYY_MM_DD corresponding also to the recording directory: in this case is used to store result images.

- **vals: required**
  In case of multiple acquisition in the same date is possible to select which recording to process; two integer are request indicating the first and the last acquisition to process. If only one acquisition have been done, just pass "1 1": in this case is used to rename result images avoiding overwriting.

- **bag_path: required**
  Location in which retrieve rosbag files generated by previous scripts, assuming are saved as *YYYY_MM_DD_*acquisition number*.bag*.

- **- -segmented: optional**
  This parameter enables segmented elaboration, creating plot images for all recognized road section giving a more precise view of the data in that context. Without this parameter, the script generates a single image per acquisition but dividing the whole graph per road sections with vertical red lines for a general overview of stress levels.

- **- -default: optional**
  This parameter let the user to store and plot default data that are:

  - "pupil_eyes__left_diameter"
  - "pupil_eyes__right_diameter"
  - "pupil_gaze__norm_pos_x"
  - "pupil_gaze__norm_pos_y"
  - "pupil_driver_gaze__object"
  - "pupil_yolo__object"
  - "LF/HF"
  - "GSR_tonic"
  - "GSR_phasic"
  - "GSR_raw"

  So by default a plot image contains 4 graphs summarizing all previous listed information. Without the parameter, the script prints the *bag* topics available and the user must select which one he wants to plot: the result is an image with only the desired graphs.

- **- -GSR: optional**
  This parameter extract the GSR data and features (tonic and phasic) from the corresponding *mat* file and generate a new plot saved in the same image.

- **- -ECG: optional**
  This parameter extract LF/HF feature from heart *mat* file plotting the result on a new plot, still in the same image for a more compact view.

- **- -orienting: optional**
  This last parameter read orienting responses data from the corresponding *csv* and create a scatter plot for a better view of the features of the orienting responses

The script can be run with any combination of the optional parameters; pay attention that files are written always in "plots > segmented_*acquisition number* " so multiple runs will overwrite the previous generated images: this memory efficient design choice aim to automatically delete images with uninteresting plots.