UNIVERSITÀ
DEGLI STUDI
FIRENZE

Scuola di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea in Informatica

TITOLO IN ITALIANO

TITLE IN ENGLISH

TERROSI FRANCESCO

BONDAVALLI ANDREA
STRIGINI LORENZO

Anno Accademico 2018-2019

ABSTRACT

Abstract

INDICE

ELENCO DELLE TABELLE

ELENCO DELLE FIGURE

1

INTRODUZIONE

Sistemi informatici ormai ovunque (Cosa sono, esempi)

1.1 CYBER-PHYSICAL SYSTEMS OF SYSTEMS

1.2 DEPENDABILITY AND SAFETY

- Dependability

- Safety

- fault tolerance - fail safe

- Considerazioni generiche sul perche' della tesi

2

AUTOMOTIVE - STATE OF ART

Self driving cars are one of the hottest topics of the decade. Artificial Intelligences specifically trained to drive with machine learning techniques demonstrated that it's possible for a computer to drive cars. However, a failure in these kind of systems may have very serious consequences that could result in people being injuried, or killed. At the same time, it is a problem to certify the ultra-high dependability requirements of these systems. In this chapter, today's problems regarding the safety issues related to self driving cars are reviewed, for that it was decided to conduct this study.

2.1 AUTONOMOUS CARS AS CPS

In order for a car to be able to drive by itself, suitable hardware and software are required. This makes autonomous cars cyber physical computer systems, and the possible catastrophic consequences that a failure in/of these systems can cause, make them fall under the set of critical systems.

To sense and map the surrounding environment, the system collects data from multiple sensors. Some of the most important sensors and their purposes are listed here:

- GPS
  - → High precision GPS sensors are used to estimate the exact position on the vehicle in the world

- Odometry & IMU sensors
  - → These sensors are worth for detecting changes in the position of the car and of the objects in the environment over time

- Cameras

$\rightarrow$ Cameras are literally the *eyes* of the system. Images captured are usually processed with image recognition software

- Lidars & Radars

$\rightarrow$ Lidars can be seen as the evolution of conventional radars. Data combined from these sensors serve the purpose of mapping the environment and detect obstacles and objects around the car

Outputs from these sensors are combined and given to the car's control system. An abstraction of the software architecture is shown in this figure:
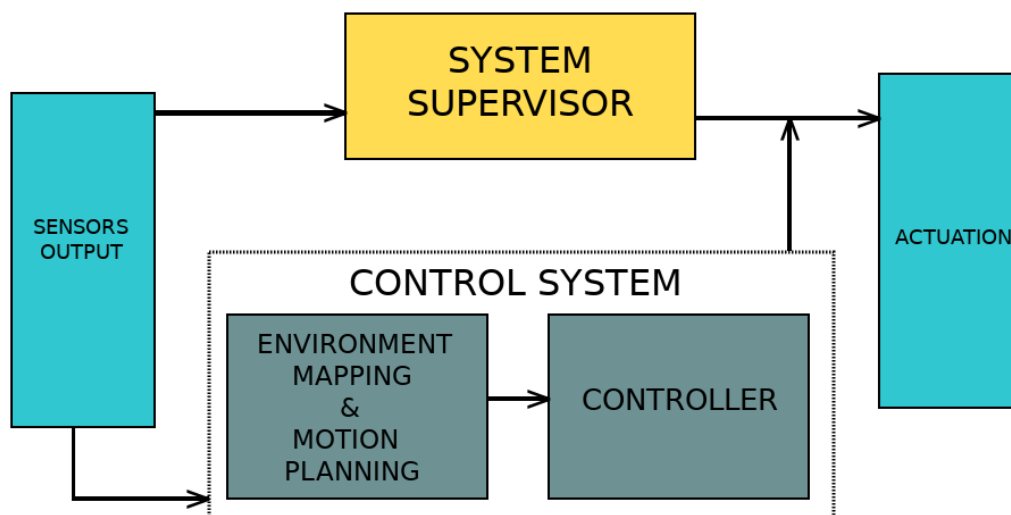
Figura 1: High-level abstraction of the system's software architecture

Data from sensors are inputs of the control system, here simplified as compose by two constituent systems: one in charge of collecting data directly from the sensors, process them in order to build an *occupancy grid*[1] to map the surrounding area and to create a physical model of

---

[1] A matrix mapping the environment, the cells $a_{i,j}$ are flagged with 0 if there's no object at coordinates i, j, 1 if occupied.

the environment in order to follow the correct route to the destination without crashing. The Controller (usually composed of a Velocity Controller[2] and a Steering Controller[3] uses these data to adjust the values on the actuator controlling the movements of the car: throttle, brake and steer.

Due to the criticity of their task, it's mandatory to have a System Supervisor, a System in charge of detecting possible hardware failures or wrong outputs[4] from the Control System and, if needed, activate a corrective routine.

The System Supervisor is the main failure avoidance component of such systems. Of course there may be specific checks when data are processed, but the last decision is up to this system's monitor and the underestimation of its importance can lead to dramatic consequences, such as the incident in Arizona in 2018, where a woman was killed by a self-driving car.[3] Further inspections showed that the car's radar and lidar data detected the victim almost 6 seconds before the impact and it took 4 seconds circa to infer that there was an obstacle on the road and that an emergency brake was needed. However, this safety-checker was disabled during tests for "smoother rides", causing the incident.[5]

The extreme complexity of these systems raise concerns among the experts: the way te system's safety is studied and tested must be looked from a new point of view and to sensitise about safety culture.[6]

2.2   SAFETY IN THE AUTOMOTIVE

The level of autonomy of a car ranges from 0 to 5. Level 0 means no autonomy: a human driver just drives the car, level 5 means that there's no need of human intervention at ll and the car is not only capable of driving safely on the road, but it must be able to avoid catastrophic failures that may seriously harm (or kill) people. The more autonomous the car is, the higher the dependability requirements are for it to be put on public roads. It is well known that demonstrating a system's dependability is not an easy task for itself, it gets even harder with ultra-high dependability systems such as these are. In adition to the problem itself, demonstrating autonomous cars' dependability has two more problems to deal with:

---

2 Controller in charge of adjusting the vehicle's speed
3 Controller that determines the steering angle
4 With *"wrong"* is intended not only outputs out of the domain space but also outputs that would cause the system to fail (e.g. causing a crash)

how to safely and effectively test the system and the need for neural networks to achieve the task.

Lots of studies demonstrated that it's unthinkable to just test cars on the roads. One of these, that we will refer to as the RAND Study, answers the question of how many mils of driving would it take to demonstrate autonomous vehicles' reliability using classical statistical inference, saying that if autonomous cars fatality rate was 20% lower than humans', it would take more than 500 years with *"a fleet of 100 autonomous vehicles being test-driven 24 hours a day, 365 days a year at an average speed of 25 miles per hour"*.[2] It's just impossible to demonstrate cars' dependability by observing them driving and hope they won't fail. Not just for the quantitative results of the impossibility of it, but also because failures and crashes are (should be) very rare failure events, therefore it could take too much time for one of them to happen, or it may not happen at all during tests.

Validating these requirements is a hard task already. Things are made harder by the fact that these cars are driven by neural networks.
In these years there is a huge interest in the *machine learning* sector, and this has made that a lot of progress was done in the research. It's also thanks to these progresses that autonomous cars now seem like something we can achieve, since these AIs gave surprising results with their skills and big names such as *Uber* and *Tesla* are putting more and more efforts in AI research.

If neural networks gave promising results on one hand, and they seem the only way to achieve goals such as autonomous cars, thanks to their ability to handle situations which they were not specifically programmed to handle, on the other hand the lack of official regulations and certifications of these kinds of software raised some concerns, especially after dramatic events such as the death of a woman in Arizona[3], and consciousness is growing on the topic, asking for more regulations on companies developing advanced AIs.[4]

- Perche' le neural network sono un problema per la safety e perche' e' difficile validarla per questi sistemi | citazioni paperz (RAND study, koopmann, high-dependability systems. . .)

2.3 NEURAL NETWORK CHE GUIDANO

2.4 CONTROLLER - CHECKER PROBLEM

===

Qui descriverei il nostro problema. Per adesso ho solo copincollato quello che avevo scritto in precedenza, userei questa sezione per giustificare cosa facciamo

===

As the network learns, we expect the area covered by the Primary to grow. With a relatively simple Monitor, in relatively simple scenarios, there will potentially be no overlapping between the hazard areas covered by the two. In this phase, the safety gain provided by the use of a (correctly implemented) safety checker will be remarkable, since the Primary is still learning to handle "easy" demands. As pointed in the previous sections, our main goal is to observe and study the variation of the dependability provided by the monitor when the network is trained to handle "hard" demands, since there are no guarantees on the Monitor's performance in the long period.

As noted in [1] the probability of a failure for a *system* composed by a *Primary Component* and a *Safety-Monitor* on a random demand X is:

$$P_{fp}(1 - \text{Coverage}_\sigma) - \text{covariance}_Q(\theta(X), C_\sigma(\sigma, X)) \tag{2.1}$$

where:

- $P_{fp}(1 - \text{Coverage}_\sigma)$ is the probability of a failure in the Primary Component (P_{fp}) that is **not detected** by the Safety Monitor (the term $1 - \text{Coverage}_\sigma$ is exactly the probability of having a false negative/positive)

- $\text{covariance}_Q(\theta(X), C_\sigma, X)$ given a demand profile $Q = <x, y>$ (i.e. the pair <demand, output), measures the correlation between:

$\theta(X)$ - The expected probability that the Primary will fail when processing demand X

$C_\sigma(\sigma, X)$ - the term identifying the *Coverage Factor* of the **Monitor**, on the specific demand X

This formula highlights the deep connection between the safety levels of the Controller and the Monitor, when it comes to the global safety of

the system. It is clear from the equation that the probability of observing a failure in the system is also depending on the specific demand X, that's .

****

TO REVIEW:

****

The formula points the fact that to have the probability of observing a failure in the system depends not only on *all the possible demands* in the *demand space* but also on how the controller and the monitor react to them.

3

SYSTEM ANALYSIS METHOD

In this chapter is presented a method to study the safety level of an autonomous car over time, observing the emergence resulting from the interactions of a neural network controller and a safety monitor in a simulated environment.
A neural network was trained, tested and trained again several times with and without the safety monitor, to collect data about the emergence of these components.

3.1 EXPERIMENTS PREPARATION AND MEASURES

The goal of this work is to develop and to assess the feasability of an experimental method for the safety assessment of an autonomous vehicle. Due to the system being composed by two constituent systems: the Controller and the Checker, we think that a point of view based on the emergent behaviour resulting from the interaction of these systems can improve the quality of the assessment.

The proposed framework is designed with particular attention on studying the emergence resulting from the interaction of these two consitutent systems. The main aspects we are interested in are:

- How the coverage of the system changes when the same monitor is applied to different stages of a network

- Changes in the safety gain provided by the same monitor when applied to different networks

- What features of the monitor determine an improvement (or worsening) to the safety of the system

- What impacts the use of a safety monitor has on the training of a neural network

- Possible behaviours of the neural network having an impact on the monitor usefulness

At the system level, we are interested in the probability of a safety-failure (e.g. a crash) and how to minimize it, or in the safety of the system as a result of the training of a neural network, the Controller and a fault tolerance mechanism, the Monitor. As long as the network is trained properly, we expect that $P_{C_i}(failure) \geqslant P_{C_{i+t}}(failure)$ where C_i represents a neural network controller trained for i epochs. At the same time we want our safety monitor to provide at least the same level of safety if the same network is trained again for t epochs.

The analysis must start with the definition of $n_{l=0...h}$ safety cases, where n is the effective number of cases and h is used to describe the *"level of difficulty"* of the simulated environment. C_i is tested in all the scenarios, starting with conditions somehow *"favorable"* to the system ($h = 0$) and then increasing the difficulty (e.g. increasing the traffic in the scenario and/or simulating a bad weather). This method allows to observe the *Time To Failure* of the Controller under diverse points of view:

MISURE PER ORA MESSE QUI, DA RIGUARDARE

- TTF_{i,j_l} as the time in which the controller at stage i fails in the ${j_l}^{th}$ scenario

- $MTTF_{i,l}$, mean time to failure when the system performs at level of difficulty l

- $MTTF_{Controller_i} = \frac{1}{n} \sum_{k=0}^{n} MTTF_{i,k}$

The controller is then trained again and re-tested in the same scenarios.

Since we are working in a simulated environment, the time to failure was computed using simulation steps, without loss of generality.

————————————————————————————————— FATTO CHE PROB INCIDENTE SOMMI A 1. COME LO ESPRIMO?

One of the main problem realated to assessing AVs' safety is the execution time. The probability of a failure increases monotonically over time, but the increasing factor should be lower the more the network is trained, so variation on the hardness of scenarios must be designed accurately. This property could result in very long simulations, but it also gives a useful hint for checking whether the system's safety is improving or not.

Once trained neural networks become essentially black boxes and even a small variation on the training parameters could result in totally different behaviours during test phase, therefore it can not be assumed that the same software (the monitor) will provide the same level of safety.

The monitor must be tested in the same scenarios in which the controller was tested and should not intervene during the simulation, but at the rigth time needed to prevent the failure event if the controller failed the scenario. The simulations previously recorded are now repeated with the monitor activated and the following measures are extracted:

- True Positive Rate (or *Sensitivity)*)

    - Rate of successful detections where the controller failed.

- False Positive Rate (or *Fall-out*)

    - Rate of the events in which there's no failure but Monitor raises alarm. It is important that the fault-tolerance mechanism is not activated or the simulation will be compromised

- False Negative Rate (or *Miss Rate*)

    - Rate of failures in the controller not detected by the monitor

Of course we desire the monitor's detections to be the most accurate possible. For this reason *Sensitivity*[1] and False Negative rate were chosen as measures of interest.

While in most of the cases a false positive will result in a state of degraded service, since a self-driving car is a safety-critical system performing in a dynamic environment, a false positive could put the system in an unsafe state (imagine performing an emergency brake for no reason on the highway), so False positive rate was taken into account as well.

However, these measures themselves aren't sufficient to detect changes in the interaction between the two CSs, so it's useful to record data such as the vehicle's speed and the controller's throttling/steering and to combine them with data recorded from the monitor to detect possible correlations between the behaviours. These values must be recomputed not only when the Monitor is improved (either by implementing a more sophisticated detection method or by improving the quality of sensors) but also when the network is trained because/due to...

---

1 True Positive rate: the proportion of safety measures applied by the monitor when actually needed

===
GIUSTIFICARE QUI IL DISCORSO FAMOSO O RIMANDARE A CAPI-
TOLO PRECEDENTE SU LETTERATURA? =====================================
   After collecting the data we need, performances can be easily compared
to assure that the use of the same monitor with 2 different networks (or
the same network in 2 different epochs) doesn't become detrimental with
more expert Controllers.

3.2 EXPERIMENTAL METHODOLOGY

In this section we propose and describe the methodology developed in
this work.

   The study consists of several experiments in a simulated realistic envi-
ronment, in which we observe how the coverage of the safety-monitor
(i.e. the probability of raising an alert if there really is a safety-hazard)
varies with respect to a neural network *in different stages of training*.
The definition of the n_0 safety cases are the first step to perform the ana-
lysis. Each scenario should be different in terms of initial conditions for
the System, but the environment should be identical. For example, given
that in a situation of regular traffic the amount of cars is at most m, all
the n_0 scenarios should spawn m cars, but the initial state of the system
should change (e.g. by changing the starting point). Now, after defining
h hardenings of the, $n_{1...h}$ variations of the scenarios are generate (note
that here n is fixed). It is important that everything else but the variation
specifically designed is identical. For such reason each stage $j_{k \neq 0}$ all the
parameters of the simulation should be identical to those in the stage j_0,
such as *seeds* used in pseudo number generators and in general all the
parameters not changed by the k_{th} variation.
The design and definition of the safety cases and the levels of difficulty
are subjected to a lot of factors, and it is impossible to cover all the
situations that may happen in a real environment. By repeating the same
safety cases in worse conditions, we have a better idea of the system's
ability to handle unattended situations than creating a specific safety case
for the bad condition itself because in such way it's possible to study
possible correlations between failures and environmental conditions.

After the definition of the scenarios, the constituent systems are tested separately to get statistics about the interactions between the two, these data are then compared to those in which the Controller and the Monitor cooperate to fullfill the assigned task.
The testing phase is organized in the following steps:

1 Controller testing phase

2 Monitor testing phase

3 Data Analysis

- ==

? System testing phase

? Data Analysis

3.2.1 *Controller Testing*

In step 1, the controller is tested alone in the $(h+1) \cdot n$ scenarios. This testing phase was developed with the idea of comparing the results of 2 different neural networks (or again, the same neural network but in different stages of training) in mind.

In order to have a more complete picture of the networks' behaviour, in addition to the TTFs as defined above additional data are collected. This data are used to enrich the understanding of the mistakes in the control system. Examples of this data are the vehicle's speed, the controller's action (throttling, steering or braking), if there was a crash: was it with another vehicle? With a pedestrian?
Since the design and especially the testing of a safety-critical system is a circular process and can not end after an acceptance test phase, informations collected in this first step can be used to guide the hardening factors in novel experiment suites.
One of the most critical parameters when designing the scenarios in which the controller will be tested, is the definition of alting criteria: we don't know how long the car must drive before eventually crashing. However this problem is still unsolved and it's let to the user to define *reasonable* criteria to stop an execution. However, we think that whenever a collision is detected the scenario should be considered concluded.

==
Credo che questa cosa sia importante. Visto che lavoriamo nell'assunzione (almeno noi) che la macchina si schianti, merita dire di trovare i

parametri che rendano "abbastanza" difficile? —-> si ricollega anche al fatto di uno goal?

==

It is important that in this step, the *exact* behaviour is recorded for repeatability, as a sort of black box. This is because a neural network tested in two executions of the *same* scenario with the s a m e initial condition could take slightly different decisions that would change the rest of the environmental conditions.

In

===

DISEGNINI PRIMA FASE

===

In the second step the positives/negatives rates of the Monitor are studied. Our goal is to measure the coverage factor of this component with respect to the failures of the Controller.
The *executions* of the simulations previously recorded are repetead in all the $j_l; j = 1 \dots n, l = 0 \dots h$ scenarios, attaching the Monitor to the system. For each scenario, the alerts raised by the Monitor are recorded but the fault-tolerance mechanism is not activated until the instant of time z needed to prevent the failure event. The choice of z is a sensitive issue for the accuracy of the measuresements: if the Monitor is activated too late, the failure will not be prevented and it's not possible to record just the alert raised (if any), since the efficacy of the safety procedure is an indicator of the Monitor's performance. At the same time, if the Monitor is activated too early, a premature alert will change the conditions in which the failure occurred. In this case the system may avoid the failure event, but it's impossible to say if the system was "good enough" to avoid it or if an early stopping of the car caused by a safty brake prevented it.
Using this approach, if the simulation environment and the hardware are powerful enough, it's possible to have a better understanding on which cases are easily handled by the Controller and not by the monitor, and vice versa. Given a Controller C_i and a Monitor M, we calculate the followings:

$$FPR = \frac{FP}{FP + TN} \tag{3.1}$$

$$TPR = \frac{TP}{TP + FN} \tag{3.2}$$

$$FNR = \frac{FN}{FN + TP} = 1 - TPR \tag{3.3}$$

As pointed before, any alert raised before time z are recorded as False Positives. This allows to effectively count the number of false alarms in a given scenario, which can then be easily aggregated with the results from all the simulations to compute a good estimate of the False Positive Rate. With a good simulator it's possible to check in every frame if a crash happened and, possibly, with what the car collided. With this information, the time t in which there was a crash can be easily extracted. Assuming a *proper choice* of z, allowing the Monitor to intervene in the time interval $t - z$ allows us to check whether the Monitor correctly detected the hazard or not.

These values calculated over the sets of the n scenario, for all the h difficulties, are useful to understand once again what kind of situations could put the system at risk, looking at the interactions between these two components. Specifically, with this approach the coverage of the 2 constituent systems is studied in two different steps but with in the same environment(s), therefore is easier to look at similarities between failures of the controller (crashes) and failures of the Monitor (false negatives), if any.

It is reasonable to think about different difficulty levels $0 \ldots h'$ for the Safety Monitor as well. A value of $h' = 0$ may be considered as using the Monitor as it is intended to be mounted on the system. Defining n' as $(h + 1) \cdot n$, the number of scenarios in which the Controller was tested autonomously, the monitor is tested under h' operating conditions, creating $(h' + 1) \cdot n'$. Examples of perturbations on the monitor may include downgrading of the sensors' quality or noise injection in the data collected.

METHOD IMPLEMENTATION AND RESULTS

In this chapter the tools used, the infrastructure and method implementation and the results collected during the analysis are reviewed.

A Neural Network was trained to drive in an urban environment. Checkpoints of the network's state during the training were recorded for comparison. These stages of the network were then tested with and without a simple safety-monitor in order to provide a new point of view to study AV's behaviours.

4.1 TOOLS AND SOFTWARE

4.1.1 *Carla Simulator*

In order to have a realistic environment, with accurate physics simulation and data sensors, the open-source simulator CARLA[7], developed by researchers at the University of Barcellona, was used. This simulator was developed with the purpose of offering an environment where AI agents can be trained to drive, with high control of the simulation parameters and the simulation of realistic sensor, which can be tuned to increase or decrease data quality, or to inject faults.

CARLA is developed with a client-server architecture in mind. The *server* is basically a game, developed with *Unreal Engine 4* in C++. C++ performances are with no doubts essential to the functionality of the server: not only the environment must be simulated (inlcuding movements of pedestrians/vehicles, weather simulation. . .), but also all the data needed from the sensors attached to the system.

==

IMMAGINE CARLA

==

CARLA is currently at version 0.9.7 and huge improvements are done at every release, gaining more attention from the experts for its realisticity. Unfortunately, when this study started, CARLA 0.9 was recently released and the tools needed for our work couldn't be found online. Thanks to the quantity of work done for the last *stable* version of CARLA, 0.8.4 was used at first.

Versions prior to 0.9 have some limitations on the control one has of the simulations parameters and on the data collectable from it. This doesn't impede our study, but of course limited in some way the informations on the environment and system. Some of these problems are still present in later versions of the simulator, but most of them were solved in the transition from 0.8 to 0.9.

One of the main problem found was with the coordinate systems. Before version 0.9, developers were using UE4's default coordinates system which is left-handed, while the standard is considered to be right-handed. This looks like not a big deal since things could be easily solved by applying a transformation matrix. However, due to performance issues (a Python client should do the real-time processng of *loads* of data at each timestep, resulting in considerable slowdowns as a result of all the processes running at the same time), it was decided to stick with the developers' decision and convert the data during analysis phase.

All the sensors available in CARLA 0.8 were used during the experiments. These can be easily accessed via the Python APIs provided:

- Cameras
    - The *"scene final"* camera provides a view of the scene (just like a regular camera)
    - The *"depth map"* camera assigns RGB values to objects to perceive *depth* of the environment
    - A *"semantic segmentation"* is used to classify different objects in the view by displaying them in different colors, according to the object's class

- Ray-cast based Lidar

The three cameras were used during the training phase of the network. Three *"scene final"* cameras are attached to the car to actually *see* the environment (one on the front and 1 per side). These cameras, combined with the *"depth map"* camera allows not only the car to see, but also to

perceive distances from objects in the scenario. The *"semantic segmentation"* provides image classification features by querying the server for ground-truth values. This is with no doubt a semplification of a real system, where the most powerful image-classification softwares are essentially neural networks. A misclassification could lead to catastrophic failures but we are not interested in their accuracy: the system's safety monitor, combining data from all the available sensors, will not "correct" the misclassification but it must react fast and safely to the consequences of it.

- CARLA

- Nervana Systems - coach (Intel)

- Reti neurali su git

- Monitor

4.2 METHOD IMPLEMENTATION

Dedicare una sezione alle decisioni prese?

4.3 RESULTS

- Interazione rete-monitor

- Safety Monitor Implementation - obstacle detection

- Come vengono raccolti i dati

- Come vengono preprocessati

5

CONCLUSIONS

BIBLIOGRAFIA

[1] Peter Popov, Lorenzo Strigini - *Assessing Asymmetric fault-tolerant Software* (Cited on page 15.)

[2] Nidhi Kalra, Susan M. Paddock - *Driving to Safety: How Many Miles of Driving Would It Take to Demonstrate Autonomous Vehicle Reliability?* (Cited on page 14.)

[3] Arizone 2018 Uber Incident - *https:en.wikipedia.orgwikiDeath_of_Elaine_Herzberg* (Cited on pages 13 and 14.)

[4] Elon Musk declarations - *https:techcrunch.com20200218elonmusksays-alladvancedaidevelopmentshouldberegulatedincludingattesla* (Cited on page 14.)

[5] Uber incident Preliminary Report - *https://www.ntsb.gov/investigations/AccidentReports/Reports/HWY18MH010-prelim.pdf* (Cited on page 13.)

[6] Philip Koopman - *http:safeautonomy.blogspot.com* (Cited on page 13.)

[7] CARLA - *http:carla.org*