POLITECNICO DI TORINO

Department of Mechanical and Aerospace Engineering

M. Sc. Thesis
in Automotive Engineering

Aggressive driving detection through Iterative DBSCAN labeling and supervised pattern recognition

Tutor

Prof. Eng. Angelo Bonfitto

Candidate

Alessandro Mauro Danusso

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF SYMBOLS

Symbol	Meaning
AAA	American Automobile Association
AD/ADAS	Autonomous Driving / Advanced Driver Assistance Systems
API	Application Programming Interface
AUC	Area Under the Curve
BN	Bayesian Network
CNN	Convolutional Neural Network
DWT	Discrete Wavelet Transform
EDB	Elementary Driving Behavior
F	Function corresponding to the Bayesian Regularization objective
FOV	Field Of View
GMM	Gaussian Mixture Model
I-DBSCAN	Iterative Density-Based Spatial Clustering of Applications with Noise
IMU	Inertial Measurement Unit
k	Number of clusters of k-means
k*	k-Star
k-NN	k-Nearest Neighbors
L	Buffer overlap
M_0	Output buffer size
minPts	Minimum number of neighbors for DBSCAN
MLP	Multilayer Perceptron
MP	Monitoring Periods
MSE	Mean Squared Error
NHTSA	National Highway Traffic Safety Administration
NN	Neural network
normPercent	Minimum percentage of data composing the normal driving cluster
$p(\mathbf{x} \mid \omega_i)$	Class-conditional probability density function of **x**
$p(\omega_i)$	A priori probability
$p(\omega_i \mid \mathbf{x})$	A posteriori probability
PCA	Principal Component Analysis
PCR	Principal Components Regression
PLSR	Partial Least Squares Regression

Q_1	25^{th} percentile
Q_2	50^{th} percentile
Q_3	75^{th} percentile
RF	Random Forest
RMS	Root Means Squares
RMSE	Root Mean Squared Error
RMSW	Root means square of weights w
ROC	Receiver Operating Characteristic
RSS	Residual Sum of Squares
SHM	Shared Memory
STD	Standard Deviation
SVM	Support Vector Machine
SVR	Support Vector Regression
T_{si}	Input sample period
w	Weights of the neural network
$\mathbf{x}$	Observation vector
Y	Output of the neural network
Y_p	Expected target output
γ	Parameter of F
ε	Radius of neighborhood for DBSCAN
Λ	Loss matrix
λ	Parameter of F
λ_{ij}	Cost of assigning an object $\mathbf{x}$ to a class ω_i
ω_i	Classes

ABSTRACT

Detection of driver aggressiveness is a significant method which helps to ensure safe driving. Aggressive driving behavior is the cause every year of a vast number of traffic accidents. The traffic accidents which result from this type of conduct are cause of mortality, severe damage and high economical cost.

The goal of this thesis is to design an algorithm for aggressive driving detection, capable of recognizing the driver's behavior. The proposed method is based on sensor features to characterize related driving sessions and to decide whether the session involves aggressive driving behavior.

Driving simulators are being increasingly used in recent years by automotive manufacturers and researchers. There are several advantages with using this system, including the increased safety enhanced repeatability, helping researchers significantly reduce time and cost. They play a key role in studies of the driver's behavior in unstable vehicle conditions and maneuvers. It is exactly this instrument that is employed to collect kinematic data of the vehicle (such as speed, acceleration and heading of the vehicle) in different scenarios. Features are extracted from every observation, which are then labeled by means of unsupervised learning and used to train a pattern recognition neural network. The algorithm is then tested in real-time in the driving simulator.

1 INTRODUCTION

Driver aggressiveness is one of the major contributing factors to fatal traffic accidents. “An individual commits a combination of moving traffic offenses so as to endanger other persons or property” is the definition of aggressive driving provided by the National Highway Traffic Safety Administration (NHTSA) [1]. The American Automobile Association (AAA) conducted a study in 2009 reporting that 56 percent of fatal crashes from 2003 to 2007 were caused by aggressive driving and speeding was the most critical behavior. This study was based on data collected by NHTSA’s Fatal Accident Reporting System. Speeding was also the principal driving behavior related to fatal crashes in 2019 (17.2 percent) [2]. Aggressive driving behavior is a psychological concept that does not have a quantitative measure. However, according to NHTSA [2] aggressive driving is characterized by the following behaviors:

- “Following improperly
- Improper or erratic lane changing
- Illegal driving on road shoulder, in ditch, or on sidewalk or median
- Passing where prohibited
- Operating the vehicle in an erratic, reckless, careless, or negligent manner or suddenly changing speeds
- Failure to yield right of way
- Failure to obey traffic signs, traffic control devices, or traffic officers, failure to observe safety zone traffic laws
- Failure to observe warnings or instructions on vehicle displaying them
- Failure to signal
- Driving too fast for conditions or in excess of posted speed limit
- Racing
- Making an improper turn”

Detection of driver aggressiveness can be an important method to increase the safety on the road.

1.1 Thesis' goal

The main goal of this thesis is to develop an algorithm capable of recognizing the behavior of the driver, distinguishing between a normal and an aggressive style. Vehicle data are collected by means of a driving simulator, features are extracted through statistical functions and a dataset is created. An unsupervised learning algorithm called I-DBSCAN is adopted to divide the still unlabeled observations in two classes: normal and aggressive behavior. A pattern recognition algorithm is then trained and employed to classify the behavior in real time.

1.2 Hardware and software components

SCANeR Studio is the chosen software capable of running the driving simulations and it is paired with a Logitech G920 steering wheel (Figure 1.1) which provides the driving inputs and can render the steering feel through the force feedback. SCANeR communicates with Matlab and Simulink by means of an API to collect the vehicle data. SCANeR Studio is a software which allows users to configure, prepare and run simulations or analyze results. It is capable of integrating 3D graphics, AD/ADAS systems and physics based sensors with the vehicle dynamics model.

Figure 1.1: Driving setup for simulation

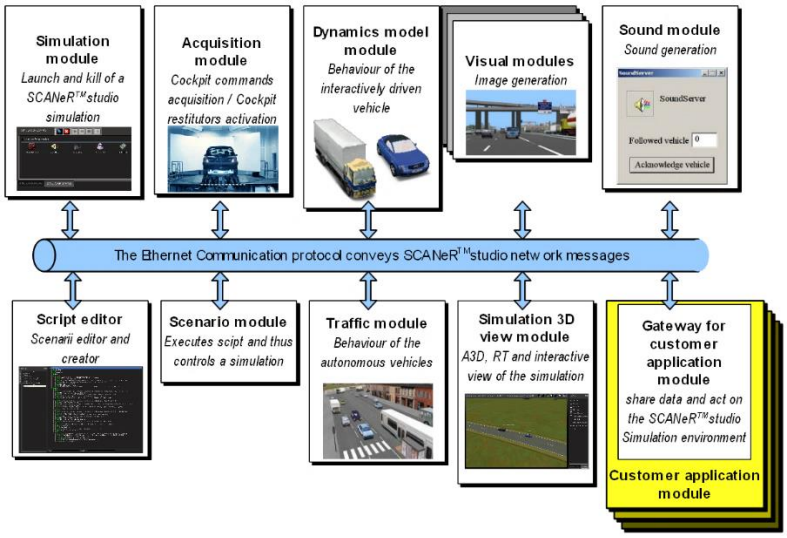

Figure 1.2: SCANeR modules architecture

The SCANeR Studio environment is populated by several modules which use a common communication protocol (Figure 1.2) [3]. The modules employed in this thesis will be (a) the simulation module, fundamental part to launch the entire driving simulation; (b) the acquisition module used to drive the interactive vehicle managing input driver commands; (c) the dynamics model module which defines

the behavior of vehicles in 3 dimensions; (d) visual, sound and simulation 3D view modules for image and sound generation to provide a more faithful simulation; (e) the scenario module used to control the simulation; and (f) the traffic module which controls the behavior of the autonomous vehicle employed to populate the scenario.

The dataset is handled and labeled with Matlab, software also used for the training of the neural network.

1.3 State of the art of aggressive driving detection

The detection of aggressive driving behaviors can be divided in three main categories: (a) anomaly detection based approaches; (b) threshold based approaches; and (c) machine learning classifier based approaches [4].

1.3.1 Anomaly detection based methods

These types of approaches build a profile of the driver's normal behavior and detect the aggressive behavior as a deviation from the normal profile.

1) Discrete Wavelet Transform: DWT is a technique which, given a time signal, divides it in time series described by coefficients representing the evolution in time of a signal [5]. For this approach the DWT used is the Daubechies wavelet to detect aggressive driving [6] [7]. This method is based on the idea that accelerometer signals of aggressive driving behaviors can be decomposed in a normal acceleration signal and some noise applied to it, which represents abrupt changes in acceleration. After the signal is decomposed, the original signal is reconstructed removing the normal signal and keeping only the component. If the distance between the original signal and the reconstructed one is greater than a certain threshold, it is marked as aggressive driving.

2) Gaussian Mixture Model: GMM is a model which performs soft clustering, it tells the probability of a point belonging to each of the possible clusters, and it takes in account the variance of the distribution. A possible application is to

compare true values of acceleration with predicted ones, in order to cluster driving data detecting aggressive behaviors. The distance of the cluster center from each point is computed and compared with a threshold. If it results greater, it is likely to indicate an abrupt change of the driver' behavior [8].

3) Partial Least Squares Regression: PLSR is a statistical method which has some similarities with principal components regression (PCR). It projects the predictors and the actual values to a new space and finds a linear regression model. It is employed to find fundamental relations between two set of data. In this case the accelerometer data is used to predicts acceleration values. Their distance with the actual values is computed and if it is significant the behavior is classified as aggressive [8].

4) Support Vector Regression: SVR is a regression algorithm which applies support vector machine to regression. While the SVM can predict discrete labels during classification, SVR is capable of predicting continuous variables. The goal of linear regressors is to minimize the error rate, instead SVR uses a hyperparameter which tries to keep the error inside a given threshold. It has been used for the forecasting and estimation of driving fatigue from electroencephalography data [9]. A different approach proposes SVR to detect asymmetry in car following behavior [10]. In [8] SVR predicts sensor data based on a previous reading and classifies the behavior as aggressive when the difference of the predictor from the actual value is greater than a given threshold.

1.3.2 Threshold based methods

These types of approaches classify driving events as aggressive when the value of specific variables exceeds a predefined threshold.

1) Thresholds on Acceleration Data: the idea of this approach is simple, when the value of acceleration read by a sensor overcomes a given threshold once or multiple times, the event is labeled as aggressive. The difficult aspect is the

identification of the optimal value of the threshold and different ones are proposed across the literature. One of the ways of taking advantage of this approach is to compute the residual sum of squares (RSS), which is a technique that measures the amount of variance in the dataset, of the distance between sensors readings at each instant and zero. Given a time window, when more than a predefined number of elements' RSS value exceeds a threshold, the entire time window is labeled as aggressive [8].

2) Jerk Evaluation: jerk is another variable useful to detect aggressive driving. It is the acceleration rate of change, in other words how fast the acceleration is changing over time and the aggressive behavior is detected when the jerk value becomes larger than a threshold [11].

3) Variable Thresholds: in [12] it is proposed to utilize a variable threshold that changes its value depending on the driving speed, since a fixed value could not provide an accurate result for different driving conditions.

1.3.3 Machine Learning based methods

Machine learning approaches take advantage of classifiers to recognize aggressive driving behaviors.

1) Parametric Classifiers: among the most used classification algorithms there are: (a) Multilayer Perceptron (MLP); (b) Convolutional Neural Network (CNN); (c) Random Forest (RF); and (d) Bayesian Network (BN). A parametric classifier is a model that approximates a dataset with a fixed number of parameters independent from the size of the training set. An example is described in [13] where the author, after extracting features from the accelerometer, applied a Random Forest to the dataset and he managed to distinguish safe from unsafe driving with an accuracy of 95.5%. In [14] the authors trained a CNN with data coming from inertial measurement unit (IMU) and GPS that was able to outperform conventional Machine Learning algorithms (for example SVM and k-NN).

2) Non-parametric Classifiers: some examples of non-parametric machine learning algorithms are: (a) Support Vector Machine (SVM); (b) k-Nearest Neighbors (k-NN); and k-Star (k*). A non-parametric classifier is a model which builds a mapping function that best fits the training data, but without losing the capacity to generalize new data. In [15] the authors compared the performance of SVM, k-NN, k*, NB, DT, RF and Artificial Neural Network applied to the classification of harsh events and this analysis showed a higher accuracy of the k* algorithm.

2 COLLECTION OF DATA AND FEATURE SELECTION

The aim of this section is to describe the process of data collection which consists of two phases, both carried out in SCANeR Studio: the creation of the scenario and the execution of the simulation [3].

2.1 Scenario definition

A scenario is defined as the aggregation of elements:

- Terrain
- A set of objects such as vehicles and pedestrians
- A set of parameters such as initial conditions and record settings
- A storyboard necessary for managing events such as situations or accidents

The scenario is created in order to fit the driving experience needed.

2.1.1 Terrain selection

The first step is the selection of the terrain. The terrain represents the 3D synthetic environment where the simulation is carried out. It is possible to use or modify an existing one or to create a new one. To choose the right terrain is important to consider: (a) the type of environment: highway, city, country; (b) the road infrastructures: traffic lights, roundabouts, crossroads, highways, secondary roads, driveroads, barriers, bridges; (c) the 3D objects such as road signals and eventually for decoration: trees, buildings, advertising boards.

For this work was chosen an existing terrain from the provided library. It is called Community (Figure 2.1) and it is a complex terrain which includes three environments: city, village and country; the city is provided with 78 traffic lights, sidewalks and pedestrian crossings. The terrain is compliant with the Physics module which handles 3D collisions and physics behavior of simulation objects (interactive vehicle, autonomous vehicle, pedestrian, bicycle, infrastructure objects,

crash barriers) and in this case sends feedback to the driver through the steering wheel

Characteristics	Community
Km covered by roads	15.9
Driving side	RHT LHT
Number of traffic lights	78
Number of barriers	0
PHYSICS compliant	yes

Table 2.1: Community characteristics

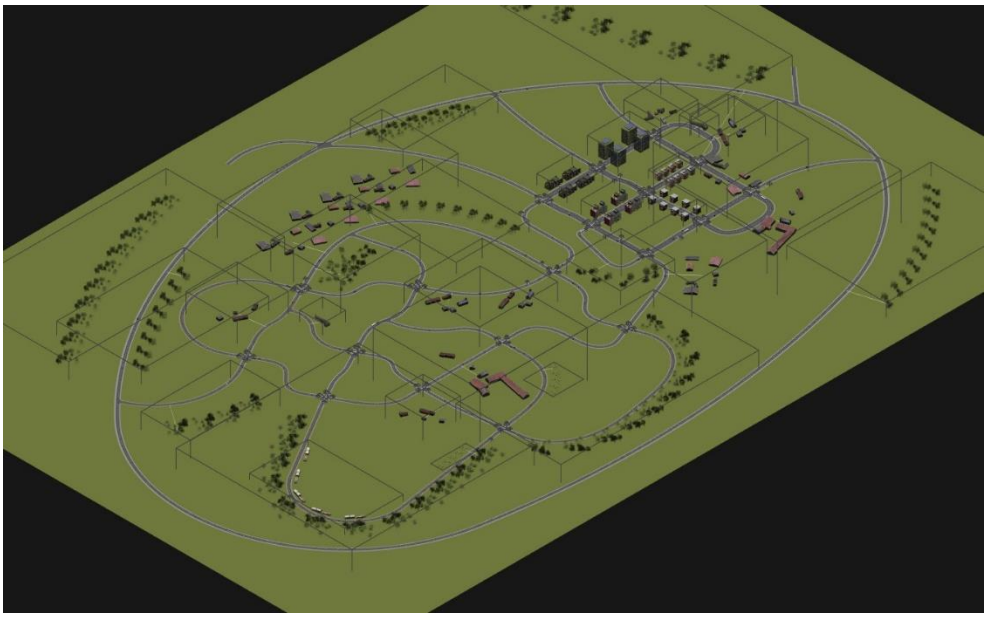

Figure 2.1: Community orthographic view

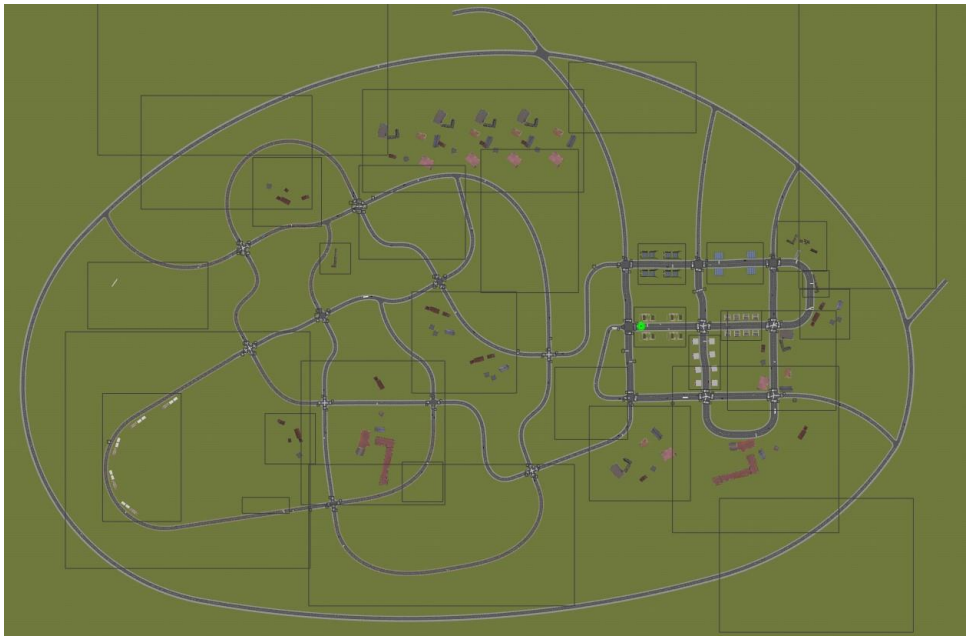

Figure 2.2: Community top view

2.1.2 Resources selection

As explained before, a scenario is an aggregation of elements including resources such as vehicles and pedestrians.

Regarding the vehicles we have to differentiate between interactive and autonomous vehicle. The former represents in this case the car interacting with the driver through the physical input while the latter are the vehicles around the subject, driven by the traffic module.

There are several dynamics models available to model the behavior depending on the needs, their complexity level depends on how they are used in the simulation.

The simple model describes tire, suspension and steering not sufficiently in detail to have an appropriate response to subtle things. Different parameters of the vehicle can be edited and several configurations are already provided. For its simplicity the most common use is the modeling of the autonomous vehicle for traffic, as happens in this study. The model characteristics are the following:

- Bi-axle (2 wheels by axle)

- position of the vehicle is computed with 1 road picking
- terrain following
- engine
- transmission
- braking
- steering.

The intermediate model is similar to the simple one but with some key differences:

- multi-axles (N wheels by axle)
- position of the vehicle is computed with 4 road picking
- suspensions

The simple vehicle dynamic model only manages 2 axles, for this reason to create a traffic vehicle with more than 2 axles, the intermediate is necessary.

Callas is the most complete and accurate dynamic model and enables the user to create co-simulations with Matlab and Simulink. It is built with a parametric approach and has high correlation with test track data.

The terrain is populated by the needed resources: one Callas model for interactive vehicle and a number of simple and intermediate models depending on the level of traffic required for the simulation.

The chosen Callas model is called ExecutiveCar (Figure 2.3) with the characteristics listed in Table 2.2.

Figure 2.3: Callas model - ExecutiveCar

Engine	
Aspiration	Gasoline
Max Power (hp / kW)	320 / 240
At Engine RPM (rpm)	5700
Max Torque (daN*m)	45
At Engine RPM (rpm)	5000
Transmission	
Transmission Type	Rear Wheel Drive
Front Gear Ratio Number	6
Rear Gear Ratio Number	1
Dimensions	
Length (mm)	4620
Width (mm)	1860
Height (mm)	1450
Weight (kg)	1834
Wheelbase (mm)	2810

Frame	
Anti-Block Brake system	yes
Active yaw control	yes
Traction control	yes
Front suspension	Independent McPherson
Rear suspension	Independent Multilink
Performances	
Max speed (km/h)	250
0-100 km/h (s)	6,4

Table 2.2: ExecutiveCar Technical datasheet

It is equipped with a Long Range Radar Sensor (Figure 2.4) in order to detect the distance to collision and is capable of detecting mobile obstacles (for example cars, pedestrians, bicycles, motorbikes). The sensor is positioned in the car front at a distance from the ground of 0.6m (Figure 2.5). It is characterized by a maximum beam range of 150m, a horizontal FOV ranging from -5° to 5° and a vertical FOV ranging from -10° to 10°.

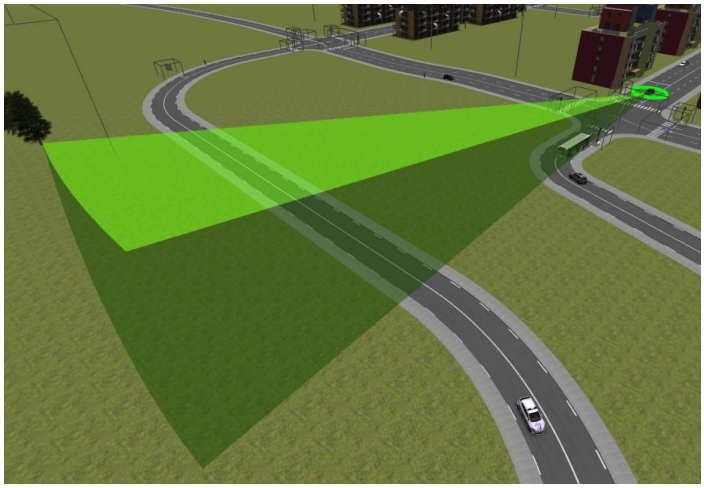

Figure 2.4: Long Range Radar Sensor

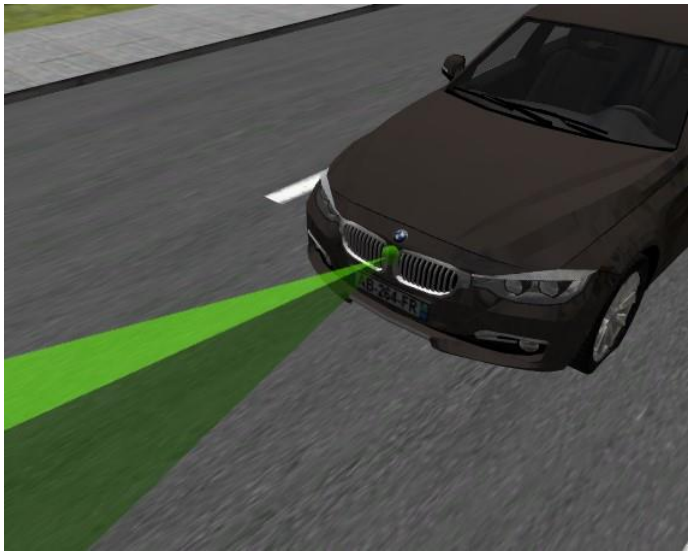

Figure 2.5: Long Range Radar Sensor positioning

The traffic tools are used to populate the terrain with autonomous vehicles. There are a total of one hundred forty-two vehicles with the distribution expressed in Table 2.3, and ten pedestrians. Of the total vehicles 90% is set with a normal behavior, 5% with a cautious behavior and 5% with an aggressive behavior.

Vehicle	**Vehicle distribution (%)**
Cars	65
Buses	5
Motorbikes	10
Bikes	10
Trucks	5
Trailer assembly	5

Table 2.3: Autonomous vehicles distribution

2.2 Simulink model

Identification of the variables which best characterize the driver's behavior is a fundamental step for an effective aggressive driving detection. The chosen variables are listed in Table 2.4.

Variables	Description	Unit
cdgSpeed_x	Longitudinal speed of the vehicle	m/s
cdgAccel_x	Longitudinal acceleration of the vehicle	m/s^2
cdgAccel_y	Lateral acceleration of the vehicle	m/s^2
Engine Speed	Speed of the engine	rad/s
Accelerator	Throttle pedal position (range: from 0 to 1)	
Brake	Brake force	N
SteeringWheel	Steering wheel angle	rad
SteeringWheelSpeed	Steering wheel speed	rad/s
distanceToCollision	Distance to collision	m

Table 2.4: Variables collected from interactive vehicle

A Simulink model is necessary to carry out a co-simulation with Matlab, with the objective of collecting the data. For this application there are two types of blocks needed: controller and input.

The controller (Figure 2.6) allows Simulink to be detected as a SCANeR module.

Figure 2.6: Controller block

The input blocks (Figure 2.7) are used to retrieve data from SCANeR simulation and they are in turn divided in two categories: SHM and Network [3].

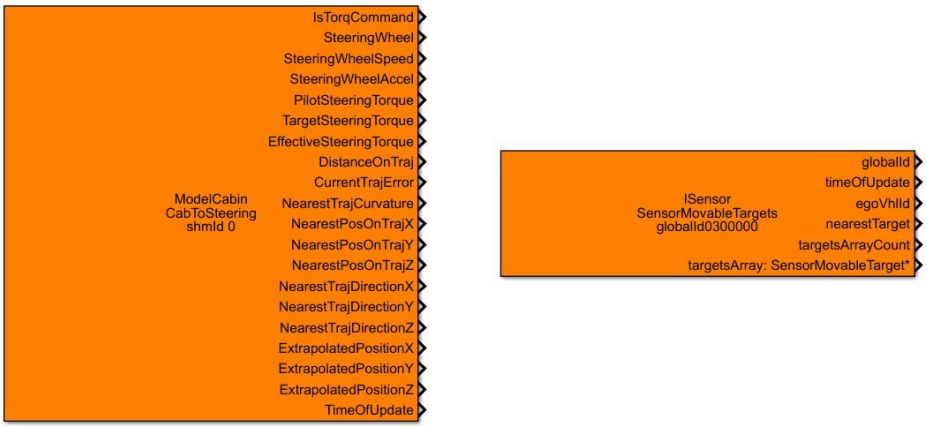

Figure 2.7: SHM and Network input blocks

The SHM blocks allow a high performance communication of several modules with the Modelhandler, reading and writing in the shared memory. Modelhandler is the module employed to handle the interaction between the road surface and the vehicle dynamic model. The SHM blocks necessary for this simulation are three: (a) VehicleOutput is necessary to read the data generated by the vehicle; (b) CabToModel outputs the inputs of the driver (accelerator and brake pedal); (c) CabToSteering reads the steering inputs of the driver (steering angle and steering wheel speed).

The network blocks allow the communication with different SCANeR modules other than the Modelhandler, in this case with the Sensors module necessary to retrieve the distance to collision.

Each block is characterized by an index representing the desired vehicle (SHM blocks) and desired sensor (Network blocks), allowing the retrieval of data from different sources simultaneously.

The signals outputted from each SCANeR block are merged in a bus creator and sent to the workspace to be furtherly processed (Figure 2.8).

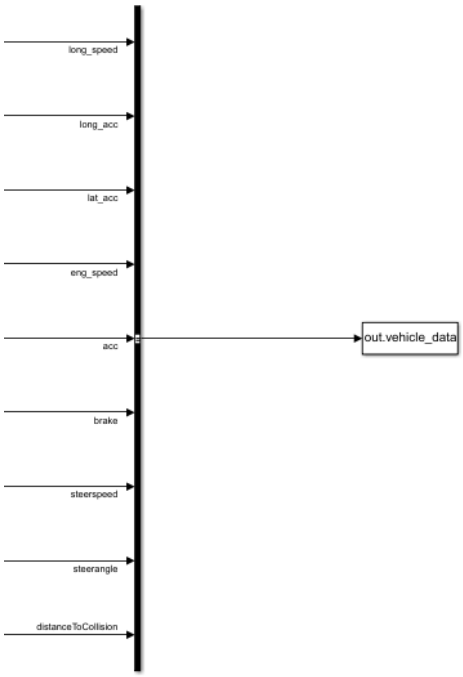

Figure 2.8: Merging of output signal to be sent to Matlab workspace

The final model used for simulation is depicted in Figure 2.9.

Figure 2.9: Simulink model for data collection

2.3 Simulation

The simulation is the fundamental step necessary to collect driving data. Before starting to drive the interactive vehicle, the co-simulation has to be launched both on SCANeR and Simulink. With the launch of the SCANeR simulation also the needed modules have to be started [3]. The modules needed for this simulation are the following:

a) Traffic: it is needed to control autonomous vehicles actions and movements and the animated road signs.
b) WalkerTraffic: it is a dedicated traffic module to control movements and actions of pedestrians.
c) Visual: it is used to show the point of view of the driver displaying: terrain, moving vehicles, pedestrians and animated road signs (Figure 2.10).
d) Dashoboard: it is used to display counters, replicating a real dashboard (Figure 2.11).
e) Sound: it is dedicated to the creation of sounds coming from the environment; in the vehicle it is possible to hear accurate vehicle sounds (for

example: engine, car horn, indicators sirens) coming from the interactive vehicle and the traffic vehicles.

f) Acquisition: It acquires inputs coming from the driver through the keyboard or steering wheel and consequently generate inputs for the vehicle model.
g) Physics: it gives to the driver feedbacks of collisions and physics behaviors.
h) ModelHandler: it manages the interaction of the interactive vehicle model with the road surface.
i) Sensors: it simulates different sensors with detection features (radar, ultrasonic, camera, light sensor).
j) Simulink: it allows the co-simulation with Simulink through SCANeR API.

Figure 2.10: Visual module

Figure 2.11: Dashboard module

Once the co-simulation is started, the interactive vehicle has to be driven freely around the terrain, trying to maintain a behavior as close as possible to the real world. If the driver commits some errors, for example hitting other vehicles or going off-road. The simulation is stopped and the data relative to that event are erased. While driving it is possible that the driver will behave in an aggressive way for limited periods, this are the events that will be detected as outliers by the I-DBSCAN. A total of thirty minutes of data have been collected which result in more than one thousand three hundred of 5 seconds observations.

Some examples of collected data are depicted in Figure 2.12. The image shows the lateral acceleration, accelerator pedal and steering angle signals.

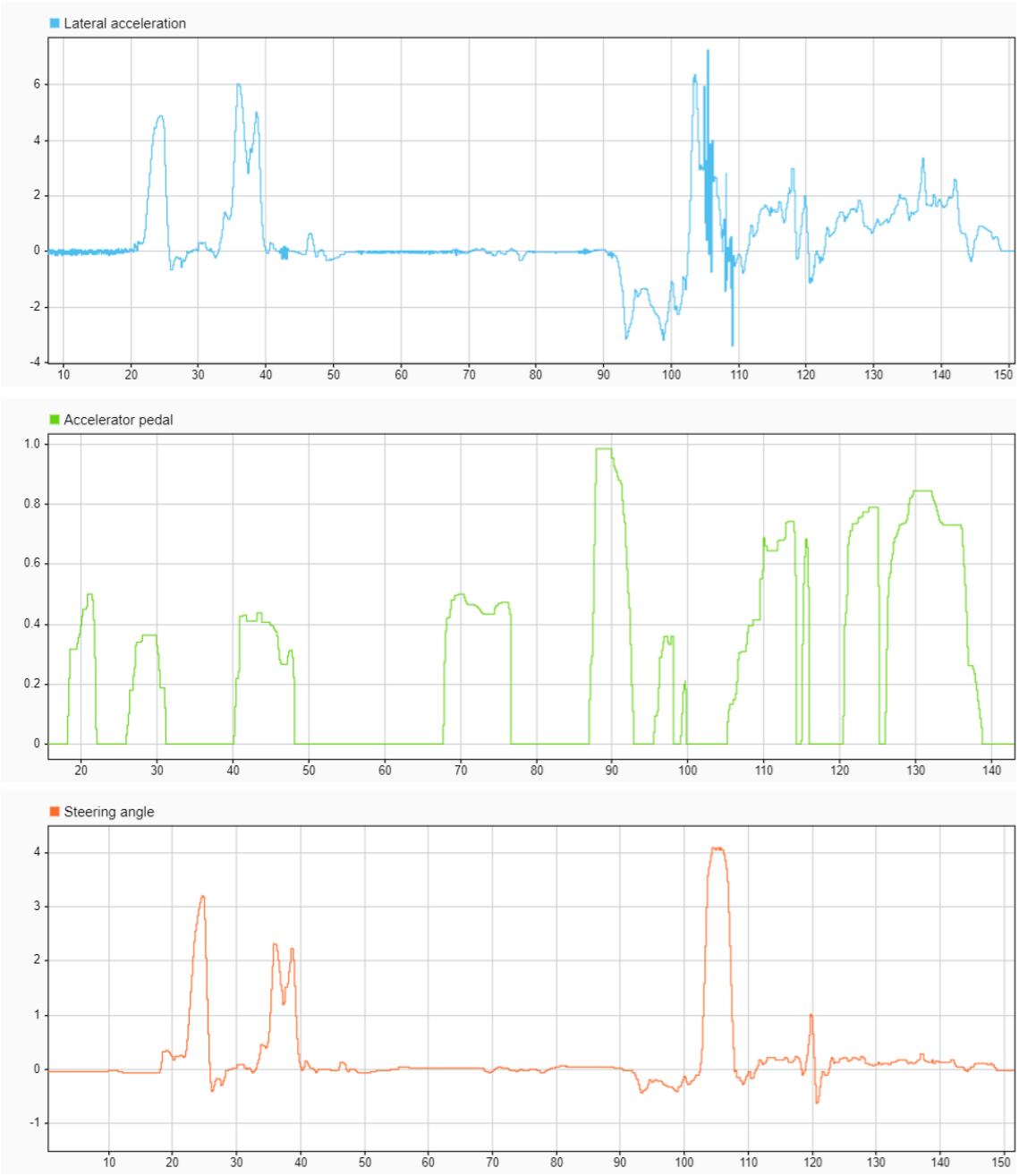

Figure 2.12: Collected signals of lateral acceleration, accelerator pedal and steering angle

2.4 Data processing and feature selection

The first step once the simulation is complete, is to format the data collected. It is necessary to divide each observation into monitoring periods (MP) of the same length which in this case is 5 seconds and occur at 1 second intervals. Since the sample rate of the simulation is 10 Hz, every MP contains 50 time steps (Figure 2.13). Each MP is now a 50x9 matrix, however the pattern recognition neural network needs a one-dimensional vector as input. For this reason, it is necessary to calculate features which characterize each period. To compute these features, the statistical functions shown in Table 2.5 have been applied to every signal of the dataset. In this way every MP is represented by 90 features.

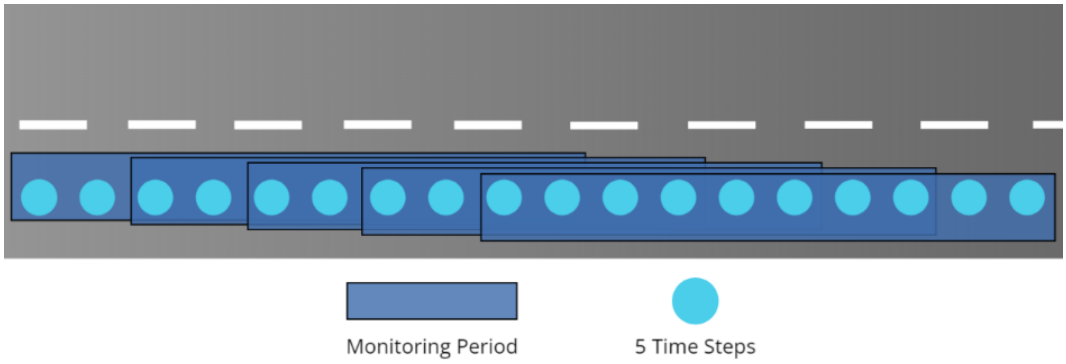

Figure 2.13: Converting time steps into monitoring periods

#	Function	Description
1	Mean	Mean of a signal
2	Min	Minimum value of a signal
3	Max	Maximum value of a signal
4	Variance	Square of the standard deviation of a signal
5	STD	Standard deviation of a signal
6	RMS	Root mean square
7	Q_1	25^{th} percentile
8	Q_2	50^{th} percentile
9	Q_3	75^{th} percentile
10	Peak Amplitude	Difference between the maximum and minimum value of the signal

Table 2.5: Selected features

3 LABELING OF THE DATASET WITH ITERATIVE DBSCAN

Looking at the literature, since there is not a unique definition of aggressive driving, it is difficult to measure and accurately quantify this behavior.

This thesis follows an approach which employs unsupervised learning to label aggressive driving behavior and uses I-DBSCAN to achieve this goal [16]. The objective of this approach is to find a small subset of observations, which represents "abnormal driving behaviors". They can in turn be identified as potential aggressive behaviors.

3.1 Creation of elementary driving behaviors

The first step is the creation of subsets of the dataset which are indicated as "elementary". This approach is based on the logic that the behaviors of a driver can be grouped into fundamental maneuvers (accelerating on a straight, making a right or left turn, merging, etc.) and each one is characterized by a data profile. Even with a similar data profile, the risky behavior will outlie the profile of an average behavior. These subsets are called elementary driving behaviors (EDB) and to establish them the k-means algorithm has been utilized.

3.1.1 K-means clustering

K-means is a type of unsupervised learning which clusters the data into a known number of groups, denoted as "k". The objective of this algorithm is to find the point of the cluster which minimizes the variance within the cluster; these points are called centroids.

For each cluster we choose, from all the points that should belong to that cluster, the centroid which minimizes the Euclidian distance between the cluster center and the remaining points of the cluster. This point is the mean of the cluster.

```
Initialise Cluster Centers
for each iteration l do
    Compute r_nk:
    for each data point x_n do
        Assign each data point to a cluster:
        for each cluster k do
            if k == argmin||x_n - μ_k^(l-1)|| then
                r_nk = 1
            else
                r_nk = 0
            end if

        end for
    end for
    for each cluster k do
        Update cluster centers as the mean of each cluster:
        μ_k^l = (∑ r_nk x_n) / (∑ r_nk)
    end for
end for
```

Algorithm 1: K-means clustering pseudocode

At the start k centroids are randomly initialized and each data point is assigned to the centroid which minimizes the Euclidian distance. Then the centers are updated by computing the mean of each cluster. This procedure is iterated until convergence of the centroids [17].

In this study case the dataset has been divided by speed and change in steering angle. K-means is run with 3 as the number of clusters (k), using only the average longitudinal speed in order to generate three distinct subsets representing low, medium and high speed (Figure 3.1). The same process has been carried out in previous studies [18] [19].

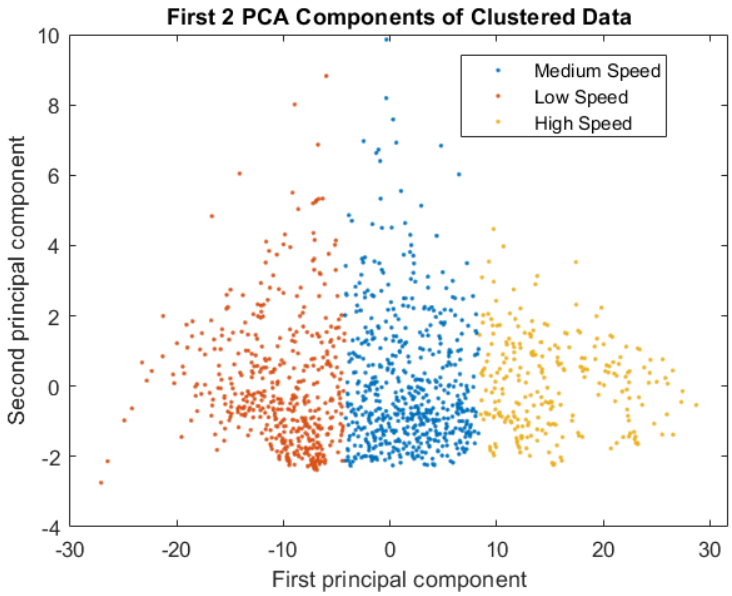

Figure 3.1: K-means clustering results

Subsequently the subsets are furtherly divided into five groups based on the change in steering angle. The first is the characterized by a change in steering angle smaller than 10 degrees (0.17 rad), representing the vehicle going on a straight. Following that there are slight left and right curves with a change in steering angle between 10 and 45 degrees (0.17 and 0.79 rad), and lastly the curves with a change in steering angle greater than 45 degrees (0.17 rad).

The entire dataset contained 1320 observations, 594 of which representing low speed, 460 representing medium speed and 266 representing high speed observations, furtherly divided depending on the change of steering angle during the period.

3.2 Iterative DBSCAN

Once the EDB have been created, the following step consists in identifying the potentially risky driving behaviors utilizing the density-based spatial clustering of applications with noise (DBSCAN) in an iterative way [16].

DBSCAN is based on a model which uses a minimum density level estimation. It creates clusters bases on a minimum number of neighbors, called “minPts” that are

enclosed within a radius ε. Objects are considered "core points" when they have more than minPts neighbors within a neighborhood of radius ε. With this idea DBSCAN tries to find areas with higher density which satisfies this threshold of minimum density and considers those with lower density as noise. If a point is found within a radius ε of a core point, it is considered to be part of the same cluster of the core point. If this point does not have similarly at least minPts neighbors within a radius ε, it is considered a non-core point or "border point". The remaining points are considered noise.

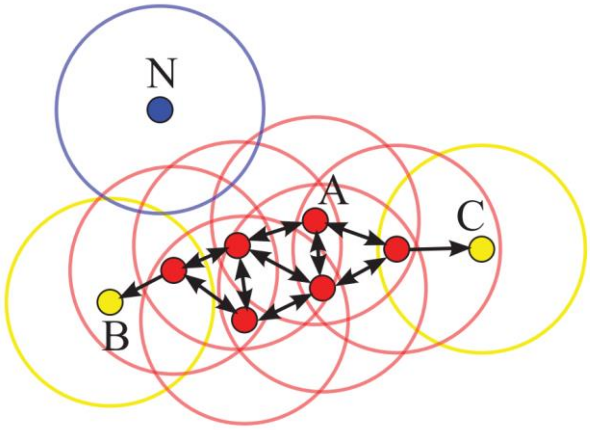

Figure 3.2: Illustration of the DBSCAN cluster model

In the example of Figure 3.2 A and the red points are core points, and since B and C are found within a radius ε from a core point, they are considered border points. These points belong to the same cluster and are deemed density reachable. Since N is not density reachable, it is labeled as noise.

The DBSCAN algorithm follows the previous model, linearly scanning the database for new points to process. When border points are found, they are considered noise and whenever a core point is discovered its neighbors are added to the cluster [20].

```
Input: DB: Database
Input: ε: Radius
Input: minPts: Density threshold
Input: dist: Distance function
Data: label: Point labels, initially undefined
1  foreach point p in database DB do
2      if label(p) ≠ undefined then continue
3      Neighbors N ← RangeQuery(DB, dist, p, ε)
4      if |N| < minPts then
5          label(p) ← Noise
6          continue
7      c ← next cluster label
8      label(p) ← c
9      Seed set S ← N \ {p}
10     foreach q in S do
11         if label(q) = Noise then label(q) ← c
12         if label(q) ≠ undefined then continue
13         Neighbors N ← RangeQuery(DB, dist, q, ε)
14         label(q) ← c
15         if |N| < minPts then continue
16         S ← S ∪ N
```

Algorithm 2: DBSCAN pseudocode

When working with high-dimensional dataset the clustering algorithms perform worse as the dimensionality increases and this phenomenon is called “curse of dimensionality”. For this reason, depending on the number of variables of the dataset it is possible to run a principal component analysis (PCA) as preliminary step to reduce dimensionality [21].

The steps of each iteration of the I-DBSCAN are the following [16]:

1. identifying DBSCAN inputs minPts and ε.
2. defining normPercent as minimum percentage of data composing the normal driving cluster, running DBSCAN on the dataset and ensuring that the percentage of normal driving observation is greater than normPercent.
3. extracting the normal cluster, any additional cluster and the noise.
4. deciding whether the procedure is complete or repeat the cycle only on the normal driving cluster.

For this case study, the Principal Component Analysis is used, since the dataset has a high number of features and the number of observations comprising a few EDB is low. Running the PCA it has been found over 90% of the variation of the variables was explained by 4 components.

The first step is the choosing of the input parameters. After the results of the Principal Component Analysis, the minPts parameter has been set to 8 as twice the dataset reduced dimensionality [22]. A way of choosing the ε parameter is by employing the "elbow" method. This method gives some insights on the density distribution of the dataset. For a given k, which in this case is set to minPts, the distance of each point to its k-th nearest neighbor is computed and a plot is constructed sorting the points in descending order depending on their k-distance value. This graph is called sorted k-dist (Figure 3.3) graph and the ε parameter can be chosen by identifying the "elbow" or "valley" of this plot. This procedure is repeated for every EDB since the characteristics of the database are different and consequently the k-dist graph is changing. The ε values found ranged from 516 – 2466.

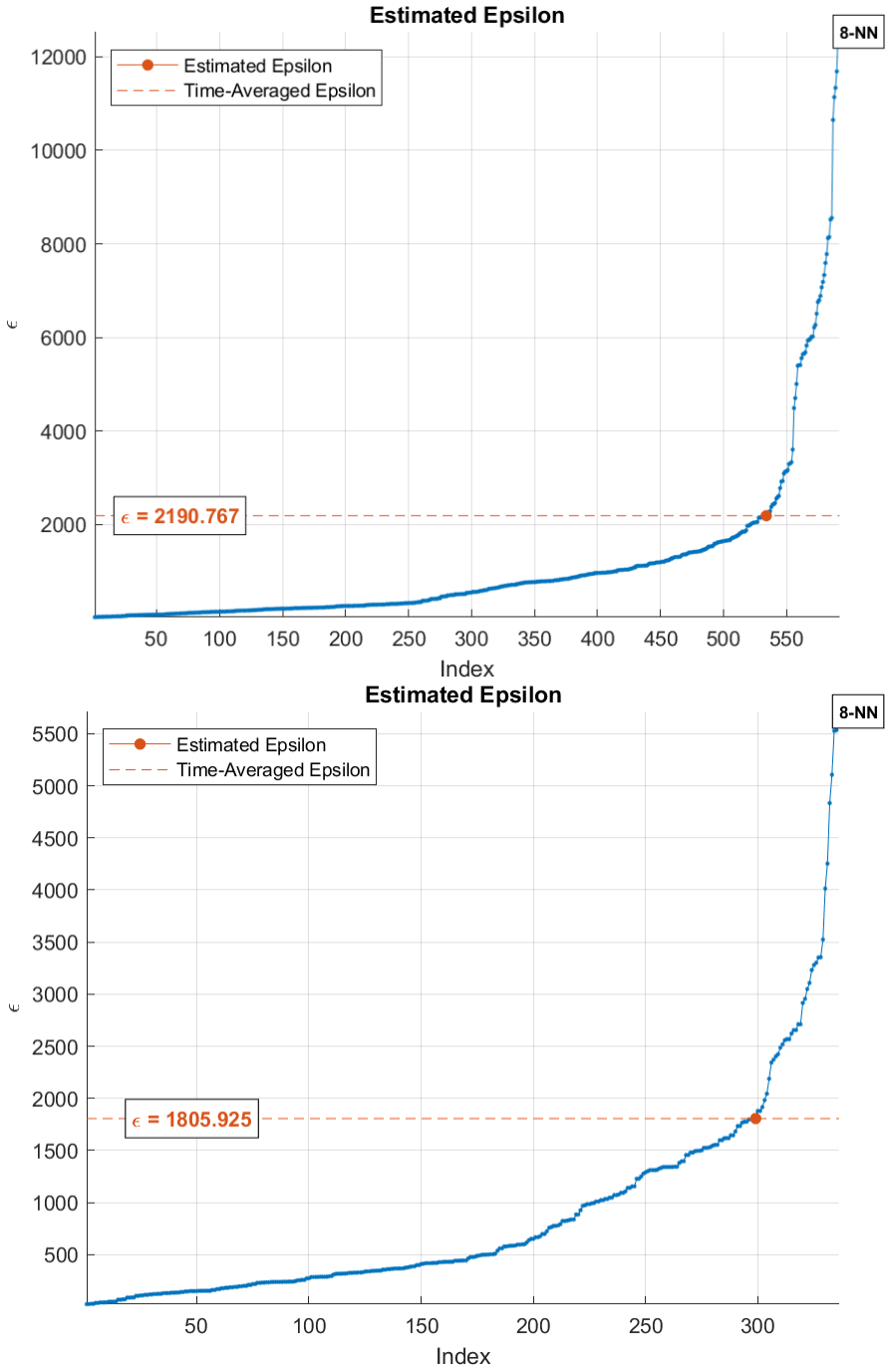

Figure 3.3: K-dist plot examples

The second step consists in running DBSCAN on each EDB. The normPercent value is set to a default value of 90%. If there is not a cluster containing at least

normPercent points, three options are available: (1) ε is increased which causes the number of clusters identified to decrease; (2) normPercent is decreased; (3) returning to the creation of elementary driving behaviors and ensuring that the data represents each EDB.

For the third step a new dataset is created from the normal cluster and this is the dataset that will be used if a new iteration of I-DBSCAN is run.

The final step consists in determining if I-DBSCAN can be terminated or another iteration has to be performed. If the normPercent threshold is not exceeded even after an adjustment of the parameters, I-DBSCAN should be terminated and the creation of the EDB has to be performed once again. For this application, after adjusting the parameters, the DBSCAN was run. At the end of the first iteration, since multiple EDB had a percentage of normal driving below normPercent (Figure 3.4), a new iteration was run with an adjusted threshold of 80%. After the second iteration, the normPercent threshold was exceeded, therefore the algorithm is terminated. Two classes result from this operation, the first representing the normal driving behavior and the second, corresponding to the noise, representing the aggressive driving behavior. The two classes are now labeled either as 0 or 1, 0 corresponding to the normal behavior class and 1 corresponding to the aggressive behavior class.

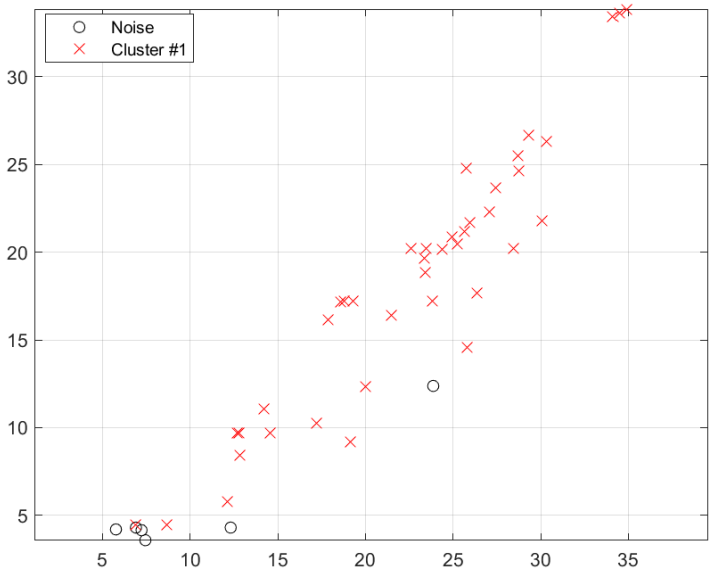

Figure 3.4: EDB with 86% of points belonging to normal profile

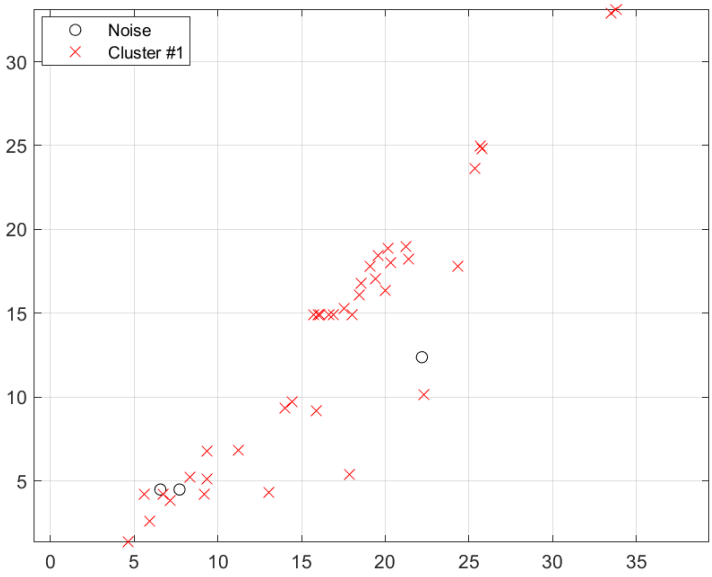

Figure 3.5: EDB with 92% of points belonging to normal profile

To test the consistency of these results, the average speed, longitudinal acceleration and lateral acceleration have been computed (Table 3.1). It is possible to notice that the values relative to the normal profile are smaller than the noise values, which correspond to a potential aggressive behavior. This result alone is not sufficient to prove the effectiveness of this clustering, therefore the average maximum and minimum value the previous variables is calculated. The absolute values

corresponding to the potential aggressive behavior are again greater than the normal values. Another important indicator is the standard deviation (expressed in parentheses in Table 3.1) that denotes an increase in variability of the noise values with respect to the normal ones. This proves that the normal profile is characterized by more homogeneous observations as opposed to the noise, which is characterized by more variability.

	Normal	**Potential Aggressive**
Average Speed (m/s)	15.66 (6.47)	18.73 (7.50)
Average Longitudinal Acceleration (m/s^2)	0.060 (1.08)	0.44 (2.03)
Average Lateral Acceleration (m/s^2)	0.008 (1.21)	0.11 (2.10)
Max Speed (m/s)	20.51 (6.19)	21.20 (7.68)
Max Longitudinal Acceleration (m/s^2)	1.79 (1.36)	3.43 (2.37)
Max Lateral Acceleration (m/s^2)	0.70 (1.17)	1.99 (2.85)
Min Speed (m/s)	10.85 (6.62)	15.99 (7.30)
Min Longitudinal Acceleration (m/s^2)	-0.64 (1.49)	-3.21 (3.23)
Min Lateral Acceleration (m/s^2)	-0.65 (1.86)	-1.83 (3.09)

Table 3.1: Statistical indicators of consistent labeling

4 CLASSIFICATION ALGORITHM FOR PATTERN RECOGNITION

This study requires a pattern recognition approach which in this case involves a supervised learning algorithm, considering that the class labels are already available. Supervised learning can be divided in three categories: statistical, structural and syntactic, but for this application the statistical pattern recognition is applied.

4.1 Statistical classification

A classifier is a pattern recognition algorithm which, given a dataset of observations described by a set of features as input, assigns to each of these objects a class label. To perform, this algorithm has to be trained beforehand with a labeled dataset where each observation is already characterized by a class label. In this way the classifier is able to learn the patterns defining each class and consequently to assign a class to an unlabeled object.

The Bayes decision theory is a principle at the foundation of statistical pattern recognition. The dataset is made of objects belonging to "c" different classes which are called ω_i with i = 1, 2, …, c, and each class occurs with the a priori probability $p(\omega_i)$. Given an "observation vector" **x**, the goal is to assign an object of the dataset to a class ω_i. The a posteriori probability that the object **x** belongs to the class ω_i is computed as

$$p(\omega_i|\mathbf{x}) = \frac{p(\mathbf{x}|\omega_i)\, p(\omega_i)}{p(\boldsymbol{x})} \tag{1}$$

expressed as function of the a priori probability and $p(\mathbf{x}|\omega_i)$, the class-conditional probability density function of **x.** This is known as the Bayes' rule for minimum error

since it assigns the object to a class with the greater a posteriori probability and consequently with the lower probability of making an error.

There is also a rule which minimizes the expected loss or risk which is measured by the cost of assigning an object **x** to a class ω_i when **x** belongs to ω_j (λ_{ij}). These costs are the components of a loss matrix $\Lambda = [\lambda_{ij}]$. The Bayes' decision rule for minimum risk is expressed by the equation:

$$r^* = \int_{\boldsymbol{x}} \min_{i=1,\dots,c} \sum_{j=1}^{c} \lambda_{ji} \; p(\omega_i|\mathbf{x})\, p(\boldsymbol{x})\, d\boldsymbol{x} \tag{2}$$

where r^* is the Bayes' risk.

In some cases, it is difficult to obtain the expected risk or the a posteriori probability, therefore for each class, a new function is evaluated and it has the name of discriminant function. One example are the neural networks which try to find the best discrimination boundary between classes, instead of estimating the probability density function [23] [24].

4.2 Bayesian regularized neural network

The chosen neural network is a fully connected feed-forward network which contains three layers: one input layer, one hidden layer and one output layer as shown in Figure 4.1. The input layer has 90 neurons such as the length of the input vector containing the observation features. It is observed the computation time increases significantly for a hidden layer with a large number of nodes, while the accuracy of the neural network is not improving significantly, therefore 10 is the chosen number of nodes for the hidden layer. The output layer has dimension 1 since the neural network outputs a single value.

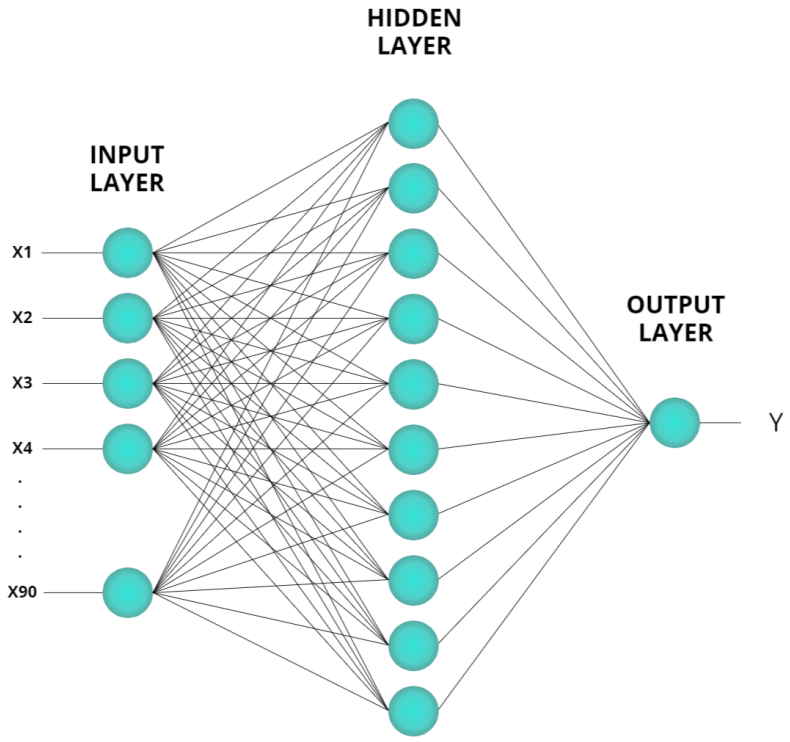

Figure 4.1: Neural network architecture

A problem of neural networks is the risk of overfitting, with this term is indicated the behavior of the NN of not generalizing well the training dataset, therefore the neural network will perform well on the training data and poorly with the test data. There are two ways to avoid overfitting: (1) the extension of the dataset and (2) the use of regularization. The latter is the easiest to use and the less time consuming, therefore it is one of the most used methods for improving the algorithm's performance and avoid overfitting.

In a simple neural network, the sequence of training is the following:

- Using an optimal state to initialize the vector of the weights
- Each object of the training dataset is run through the network
- At each iteration a loss function is computed as the squared difference between the expected output and the actual output

- Backpropagation is employed to modify the weights at each iteration in order to minimize the cost function, which is the average of the loss functions computed on all the training data

In this study the Bayesian regularization training algorithm is used in order to optimize the weight and bias values, updating them according to Levenberg-Marquardt optimization [25]. This aims to minimize the root mean squared error (RMSE) which is expressed as:

$$RMSE = \sqrt{\frac{1}{m}\sum_{j=1}^{m}\left[Y_p(j) - Y(j)\right]^2} \quad (3)$$

with m is the number of predictions, Y_p is the expected target output and Y is the output of the neural network.

The root means square of the n number of weights w can be written as:

$$RMSW = \sqrt{\frac{1}{n}\sum_{j=1}^{n} w_j^2} \quad (4)$$

The function which expresses the Levenberg-Marquardt optimization can be computed as:

$$f(x) = \frac{1}{2}\sum_{j=1}^{n}\left[f_j(x)\right]^2$$

(5)

The function corresponding to the Bayesian Regularization objective can be expressed as:

$$F = \gamma RMSW + \lambda RMSE$$

(6)

Where γ and λ are parameters of the function.

The steps carried out by the Bayesian Regularization optimization are the following:

1. Before the first training step, the network weights, γ and λ have to be initialized. After the first iteration, these values will be obtained from the previous iteration.
2. The Levenberg-Marquardt optimization function (equation 5) is used in order to minimize the Bayesian Regularization objective equation F (equation 6).
3. Estimation of γ and λ.
4. Iteration of the previous steps until convergence of the network.

4.3 Training and results of the neural network

The training of the neural network has been carried out in Matlab. The data have first been divided into two datasets: (a) a training dataset containing 80% of the data, (b) a testing dataset containing the remaining 20% of the dataset. The division function choses the indices for the two partitions randomly.

To build the neural network the Matlab function *patternnet* which requires the input arguments listed in Table 4.1.

Input	Input argument	Description
hiddenSizes	10	Size of the hidden layers of the network, expressed as a row vector
trainFcn	*'trainbr'*	Network training function, in this case Bayesian regularization backpropagation
performFcn	*'mse'*	Mean squared normalized error performance function

Table 4.1: patternnet function inputs

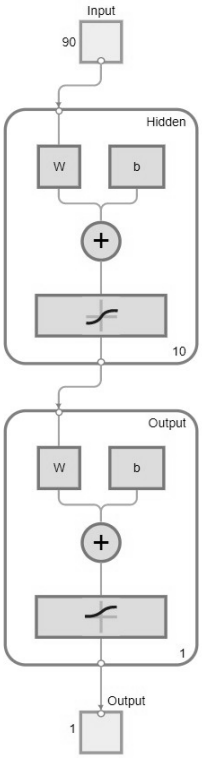

Figure 4.2: Neural Network model

The dataset and the classes are fed separately to the neural network which is then trained. The training stopped at epoch 79 and gave the results listed in Table 4.2.

Unit	Initial value	Stopped value
Epoch	0	79
Performance (MSE)	0.758	0.00223
Gradient	0.89	0.00092
Mu	0.005	500
Effective # Parameter	921	93.2
Sum Squared Param…	36.9	83.9

Table 4.2: Neural Network training results

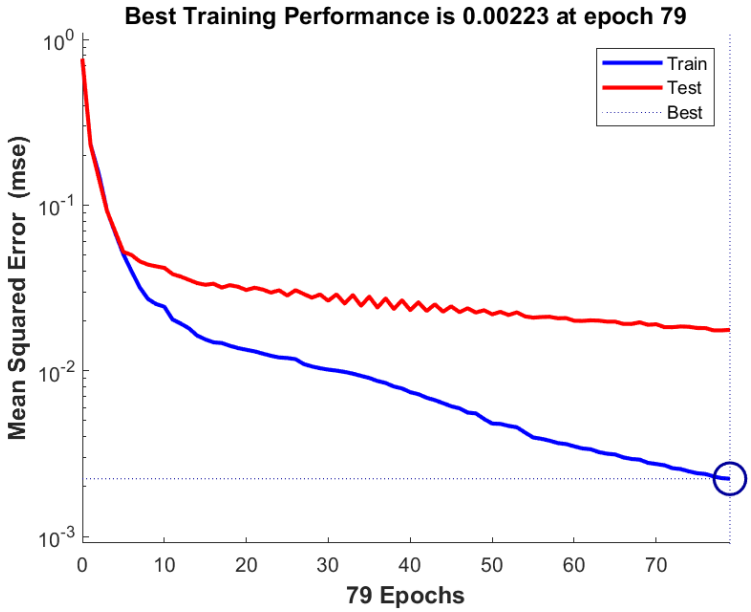

Figure 4.3: Neural Network training performance plot

It is now necessary to verify the if the results achieved by the training algorithm are satisfactory. The plot in Figure 4.3 represents the performance of the neural network, measured in term of mean squared error. It shows a rapid decrease for both training and test set as the neural network was trained, reaching a minimum at the last epoch. Unlike other training algorithms, the Bayesian regularization backpropagation does not need a validation set. The reason is that the use of validation is itself of form of regularization but, as the name suggests, this algorithm applies its own form of regularization.

There are other plots and parameters necessary to evaluate the training performance of the neural network, one of them is the confusion matrix (Figure 4.4)

A confusion matrix is an *n x n* matrix, where n is the number of target classes, which compares the results of the classification predicted by the neural network with the actual target values. The cells of the diagonal contain the percentage of the cases correctly classified and the off-diagonal cells contain the cases not correctly classified.

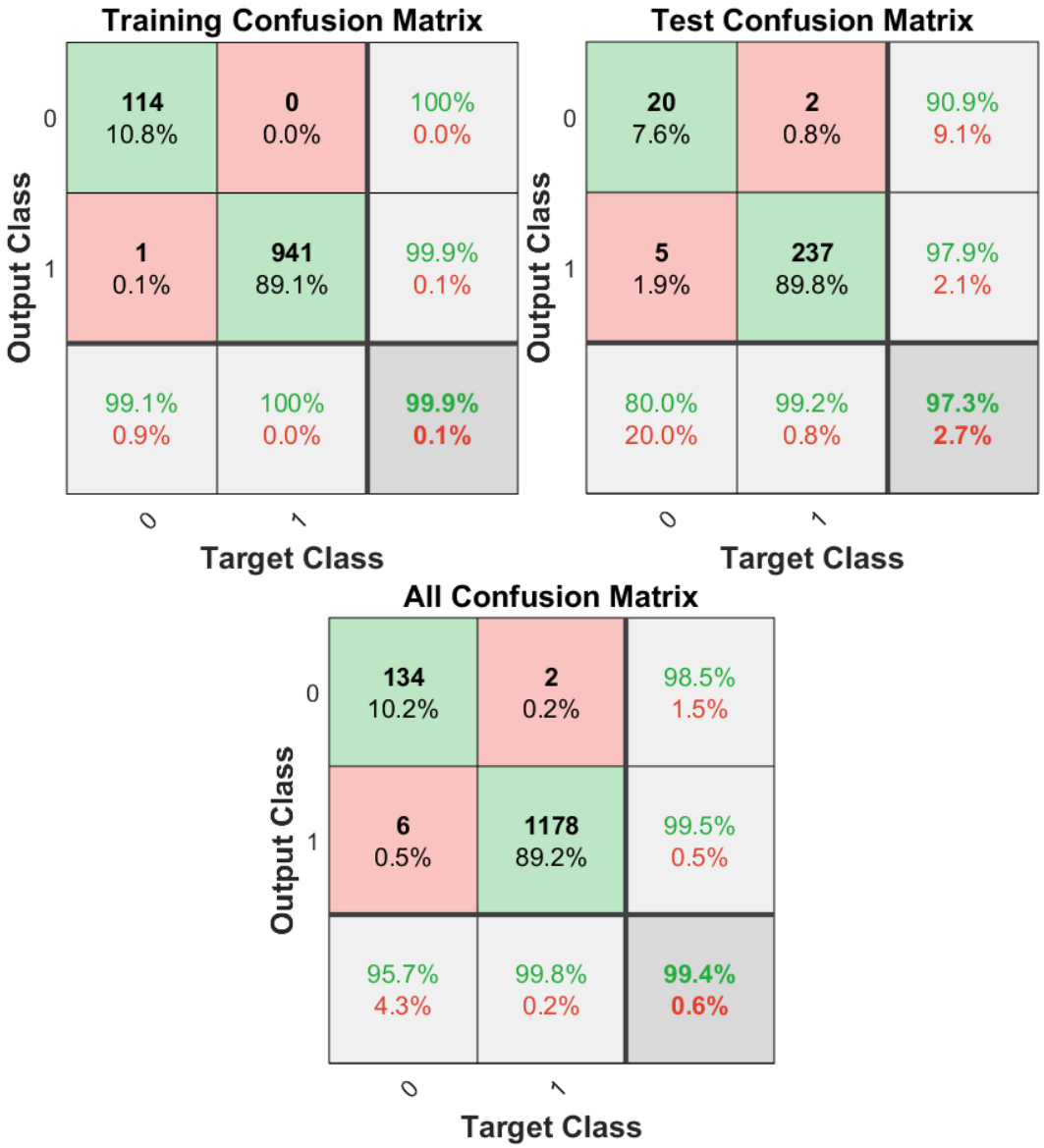

Figure 4.4: Neural Network confusion matrices

It can be noticed that the training confusion matrix has reached an accuracy of nearly 100%. It means that the weights of the neural network are perfectly

calibrated to classify the training data, but this is not a certainty that it will classify new data with the same accuracy. As a matter of fact, the test confusion matrix shows that the accuracy reached for new data is smaller with a tendency to misclassify the class 0 (normal driving behavior), maintaining however an acceptable accuracy.

The next useful plot is represented by the error histogram (Figure 4.5). It shows the distribution of the errors committed by the neural network on the testing instances. It has a normal distribution with the majority of errors near zero.

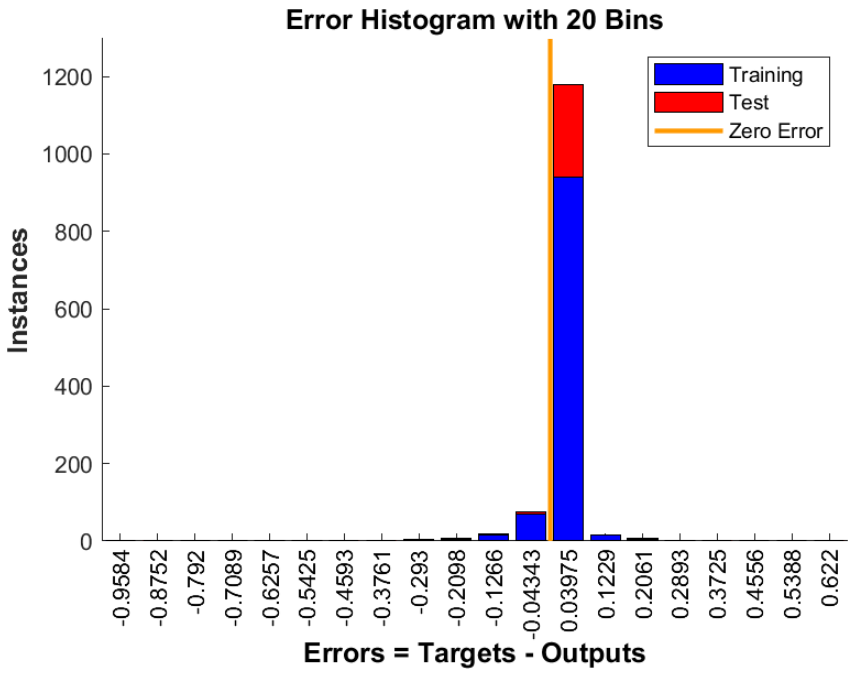

Figure 4.5: Neural Network error histogram

The regression plot (Figure 4.6) is another indicator of how well the trained neural network has fit the data, where the outputs of the neural network and the corresponding targets are plotted. The most important parameter computed with this analysis is the correlation coefficient R and indicates how strong the linear relationship between outputs and targets is. The closer its value is to 0, the weaker the correlation is. The closer its value is to 1, the stronger the correlation is. This relationship can be graphically seen by how close the fit line is to the diagonal dashed line.

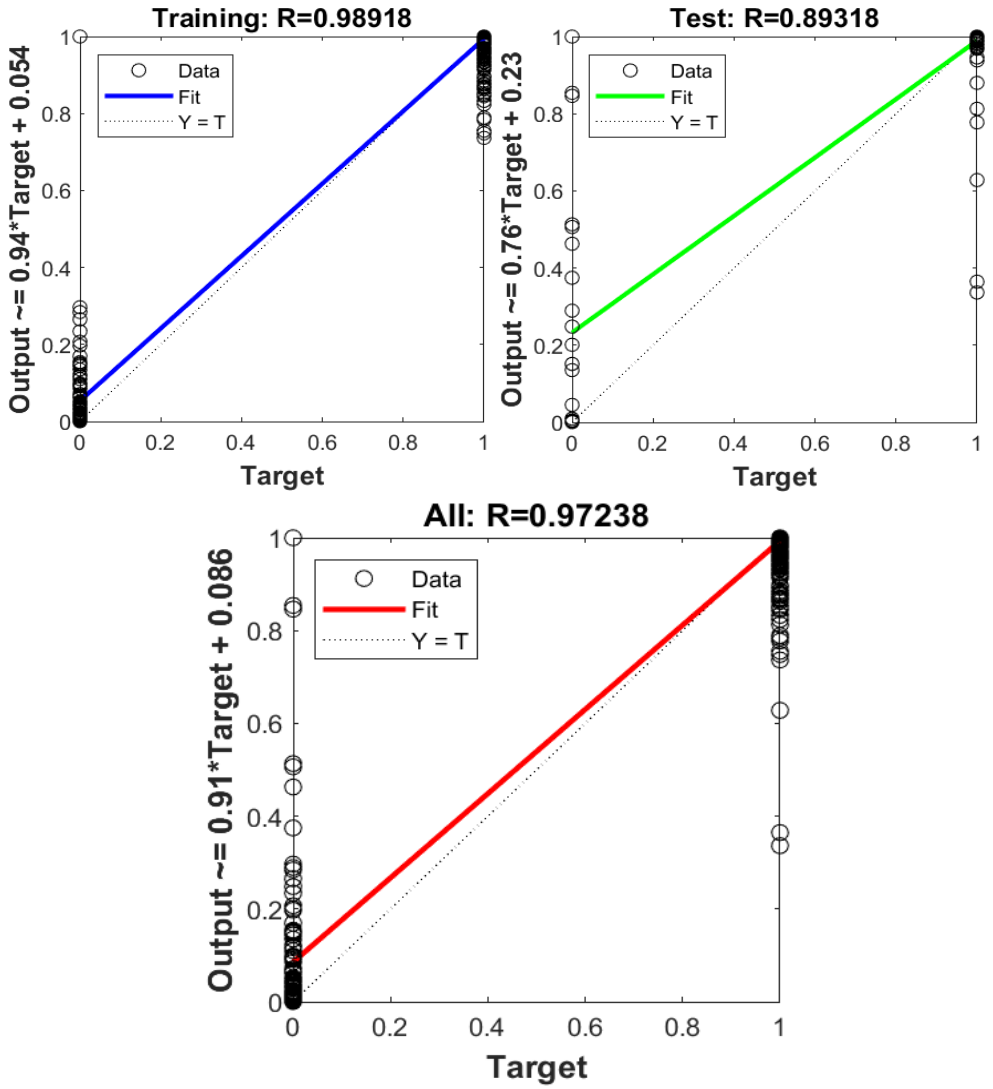

Figure 4.6: Neural Network regression plots

The ROC (Receiver Operating Characteristic) curve (Figure 4.7) is a plot with false positives rate on the X-axis and true negatives rate on the Y-axis for every classification threshold. The area under the curve (AUC) measures the capacity of discrimination. the more the AUC is close to 1, the more accurate the classification is.

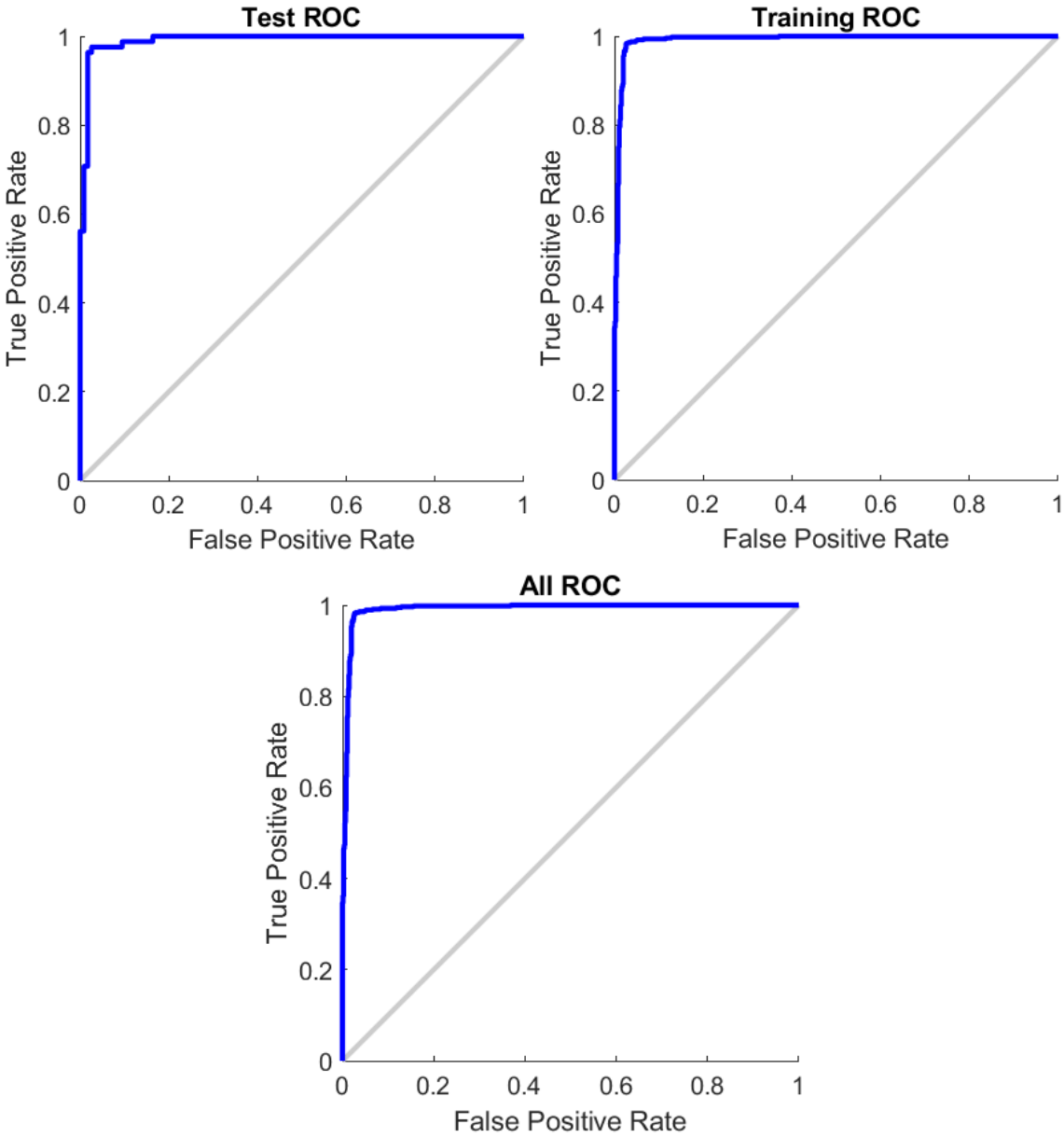

Figure 4.7: Neural Network ROC plots

The last plots (Figure 4.8) represent the progression of the training algorithm parameters during the training. In conclusion the performance results after the neural network training show an acceptable accuracy, therefore it can now be tested in real-time in the driving simulator.

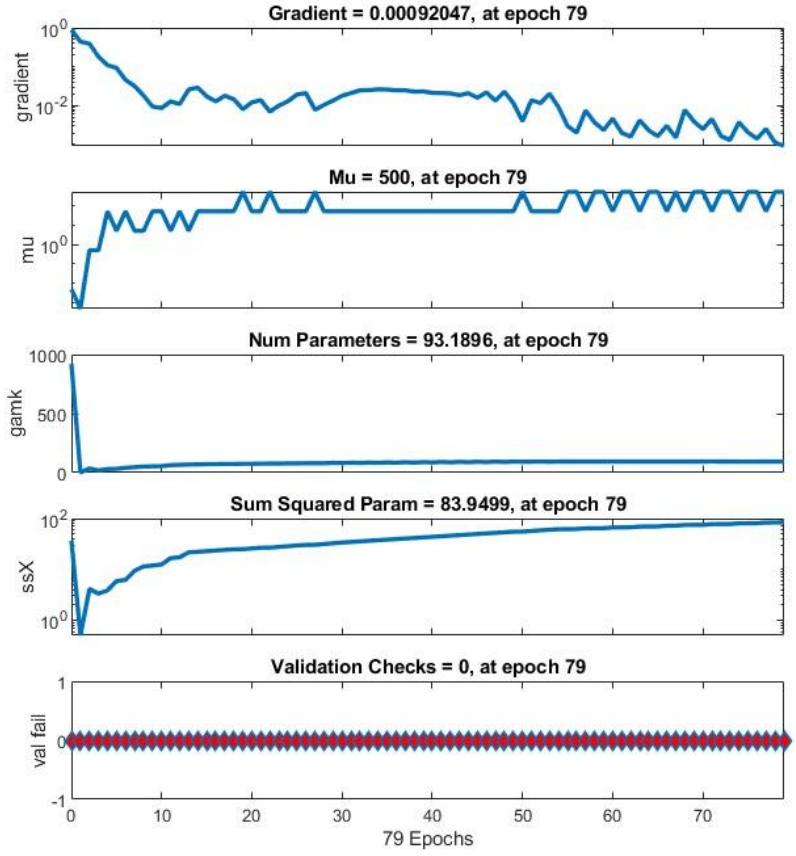

Figure 4.8: Progression of Neural Network training parameters

5 TESTING OF THE CLASSIFIER IN A REAL-TIME SIMULATION

The final test consists in a co-simulation with SCANeR and Simulink. The interactive vehicle is driven in SCANeR and the data is collected with Simulink in the same way as the first simulation.

To be analyzed by the neural network, the input must be in the same form of the training data, therefore the signals coming from SCANeR have to be divided into 5 seconds MP and the features have to be computed for every period. To achieve this objective a technique called sliding window is employed. The Simulink block used is the buffer block, where the output buffer size (M_0) and the buffer overlap (L) are set to 50 and 40 respectively. The buffer L samples from the previous output, it adds M_0 – L new samples from the input signal and propagates it as the new output. The resulting output frame period is computed with the equation

$$Output\ Frame\ Period = (M_0 - L)\ T_{si}$$

with T_{si} as the input sample period. Since in this case the sample period is 0.1s, the resulting output frame period will be 1s. This means that every second a new frame will be outputted, which includes 40 timesteps from the previous frame and 10 new timesteps.

Every signal corresponding to a different variable coming from the vehicle, is connected to a buffer. From each output frame coming from the buffer, the statistical features are computed (Figure 5.1). The resulting features of every variable are joined together to create a vector.

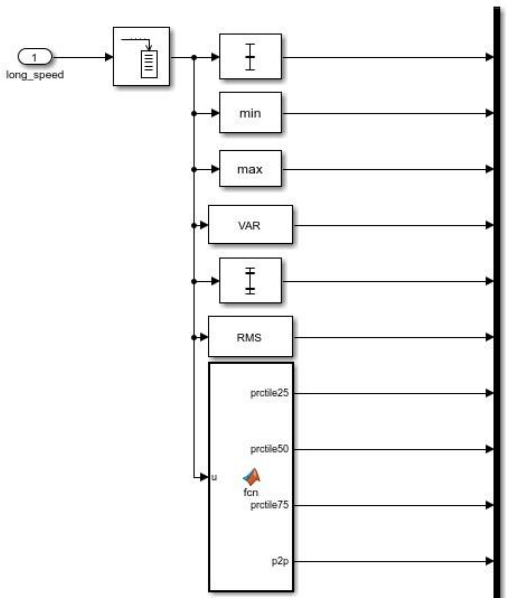

Figure 5.1: Simulink model for sliding window and feature calculation

The resulting vector of size 90 is fed to the trained pattern recognition neural network (Figure 5.2). The NN outputs a value ranging between 0 and 1 which is connected to a switch and if it exceeds a threshold of 0.85, the assertion block will display a message and a red signal explaining that an aggressive behavior has been detected.

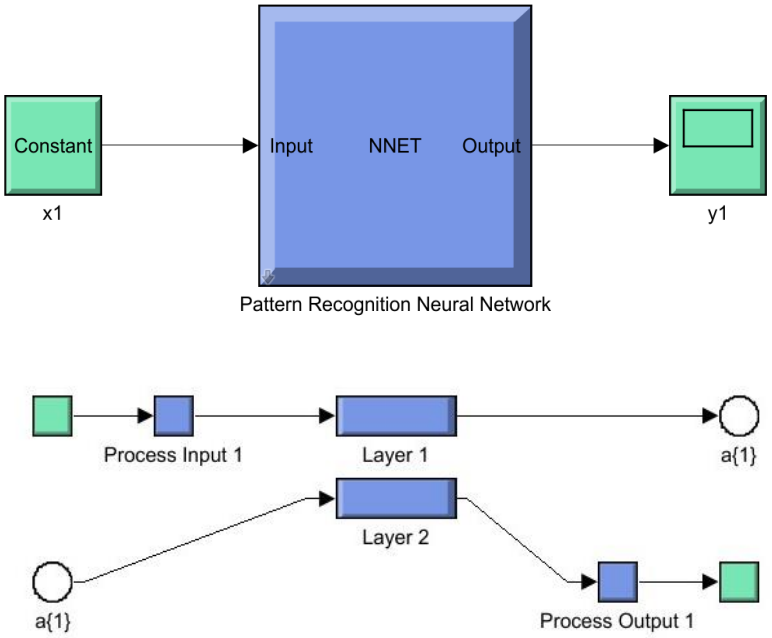

Figure 5.2: Neural Network Simulink model

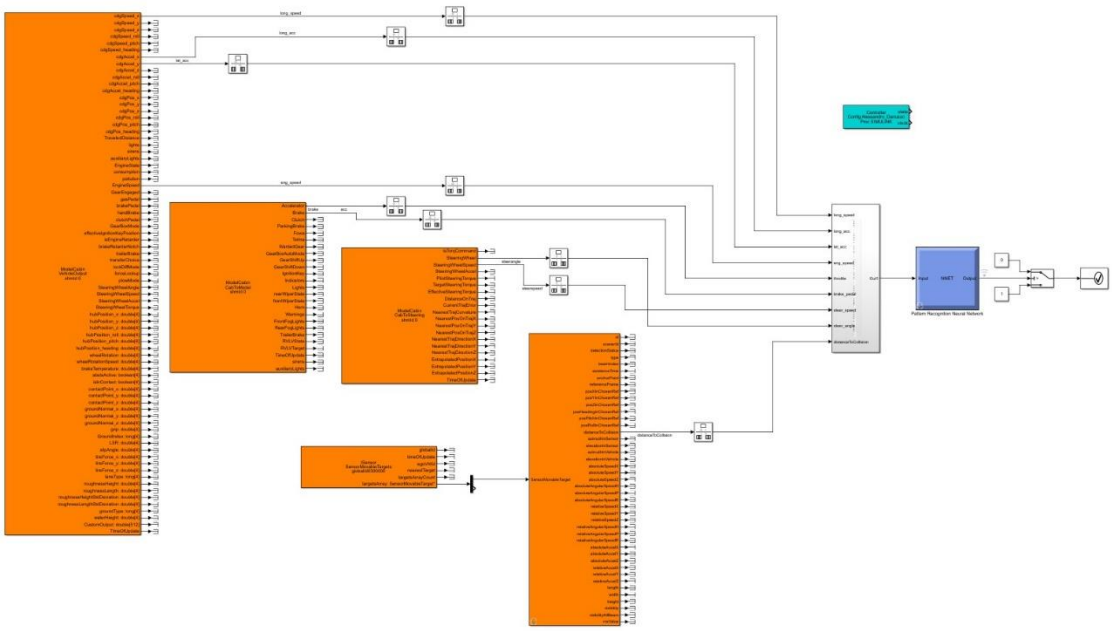

Figure 5.3: Simulink model for real-time classification

For this simulation the same scenario as the one for data collection is kept.

The SCANeR modules are started, the Simulink and SCANeR simulations are run and the interactive vehicle can now be driven freely.

Utilizing Simulink's data inspector is possible to display the output of the neural network and the assertion signal (Figure 5.4). Comparing the data of longitudinal acceleration with the results from the NN (Figure 5.5), it can be noted that for the highest peaks of this variable the neural network recognizes an aggressive behavior. Another confirmation of the NN effectiveness comes from the comparison of its output with the brake force (Figure 5.6), it classifies hard braking as aggressive behavior.

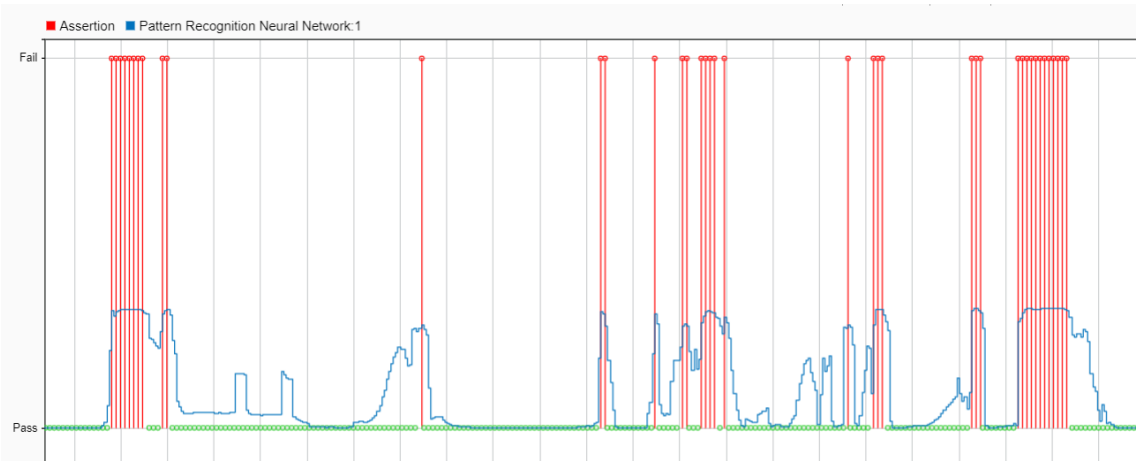

Figure 5.4: Comparison between Neural Network output and assertion block signal

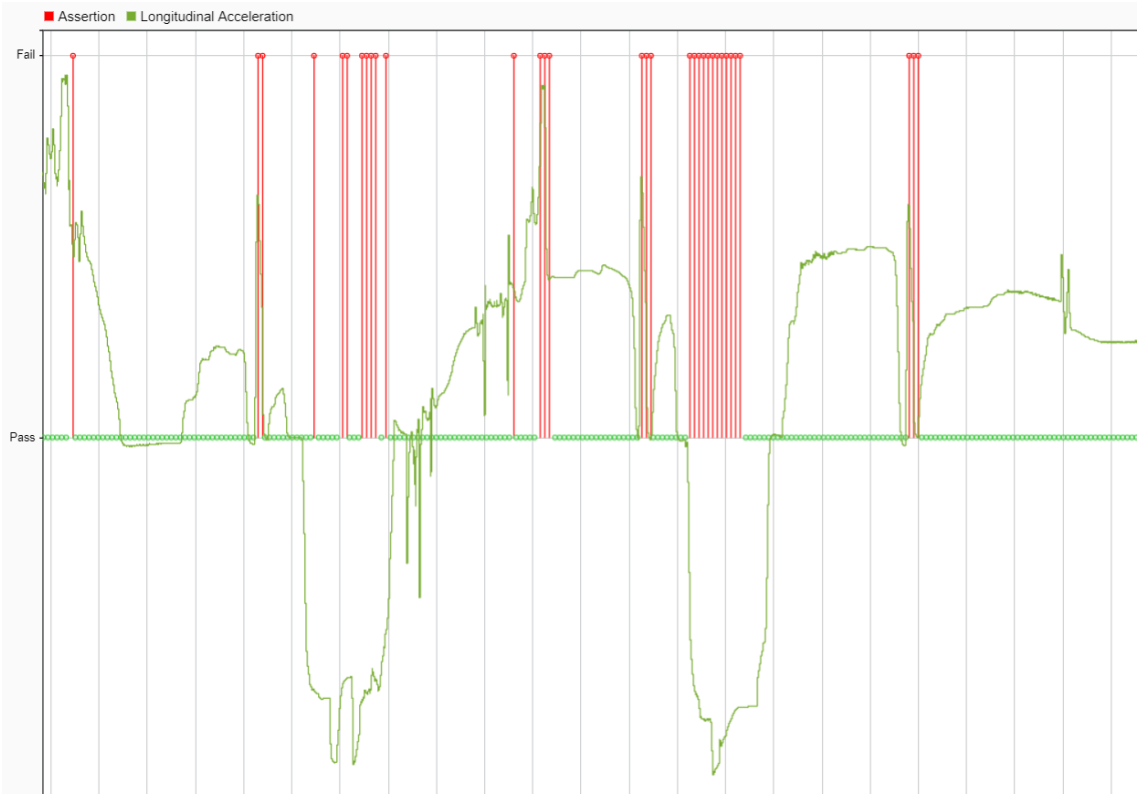

Figure 5.5: Comparison between lateral acceleration signal and assertion block signal

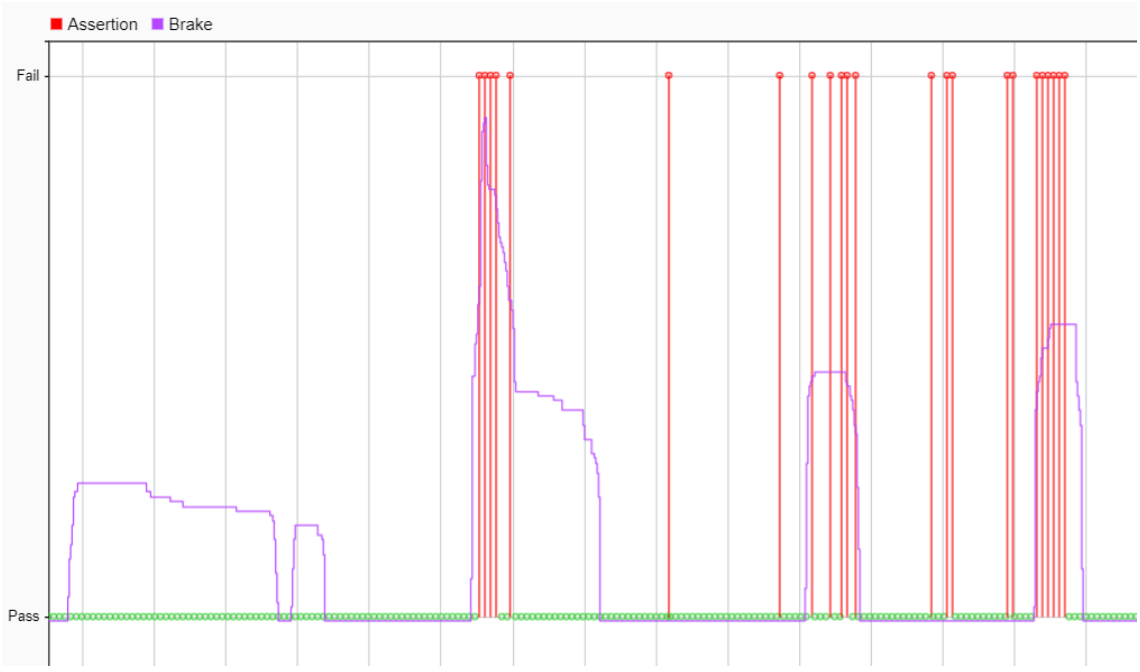

Figure 5.6: Comparison between brake force signal and assertion block signal

6 CONCLUSIONS AND FUTURE WORK

The aim of this thesis was to drive freely an interactive vehicle in a custom scenario using the driving simulator SCANeR. Timeseries of nine chosen kinematic variables are collected with Matlab/Simulink and a dataset is created. In order to train a pattern recognition algorithm to classify normal and aggressive behavior, the dataset is divided into 5 seconds windows with 4 seconds of overlap and for every window, ten statistical features are computed. Every 5 seconds observation is now characterized by ninety values and needs to be labeled. For the labeling the Iterative DBSCAN approach is adopted. The observations have to be divided into elementary driving behaviors and subsequently assigned to one of two classes, the first representing the normal profile characterizing an EDB and the second representing the outlier observations considered as aggressive driving. The labeled dataset is used to train a pattern recognition neural network with Bayesian regularization backpropagation in Matlab environment. The trained neural network is finally tested in real-time with SCANeR driving simulator and Simulink.

For what concerns the dataset labeling with unsupervised learning, the collected statistical indicators confirm the consistency of the results. The average and maximum speed and acceleration are significantly greater for the potential aggressive behavior with respect to the normal one. Another indicator is the standard deviation that explains the variability of the data. As expected, the variability is greater for the aggressive driving since it represents a more heterogeneous category than the normal profile which is a unified cluster.

The final results coming from the testing in real-time of the neural network show an acceptable accuracy in recognizing an aggressive behavior. Comparing the NN output and variable signals such as brake force and acceleration points out the

relationship between aggressive driving events and peaks of acceleration and braking force.

Despite the achieved results, there are different aspects of this work that could be further explored or improved, for example, a more advanced driving setup can be utilized in the phase of data collection in order to best replicate real driving conditions. Another improvement relative to the same phase would be an increased amount of data collected and an employment of different drivers to train the algorithm on different driving profiles. A comparison between different classifier could also be conducted in order to analyze the performance of different algorithms.

REFERENCES

[1] National Highway Traffic Safety Administration, «Aggressive Driving Enforcement: Evaluation of Two Demonstration Programs,» March 2004.

[2] AAA Foundation for Traffic Safety, «Aggressive Driving: Research Update,» April, 2009.

[3] AVSimulation, «SCANeR™studio's documentation».

[4] V. Gatteschi, A. Cannavò, F. Lamberti, L. Morra e P. Montuschi, «Comparing algorithms for aggressive driving event detection based on vehicle motion data,» in *IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY*, 2021, pp. 53-68.

[5] M. Hosseinzadeh, «4 - Robust control applications in biomedical engineering: Control of depth of hypnosis,» in *Control Applications for Biomedical Engineering Systems*, Ahmad Taher Azar, 2020, pp. 89-125.

[6] H. R. Eftekhari and M. Ghatee, «Hybrid of discrete wavelet transform and adaptive neuro fuzzy inference system for overall driving behavior recognition,» *Transp. Res. Part F, Traffic Psychol. Behav.,* vol. 58, p. 782–796, 2018.

[7] S.-R. G. Christopoulos, S. Kanarachos, and A. Chroneos, «Learning driver braking behavior using smartphones, neural networks and the sliding correlation coefficient: Road anomaly case study,,» *IEEE Trans. Int. Transp. Syst.,* vol. 20, n. 1, pp. 65-74, 2019.

[8] Y. Ma, Z. Zhang, S. Chen, Y. Yu, and K. Tang, «A comparative study of aggressive driving behavior recognition algorithms based on vehicle motion data,» *IEEE Access,* vol. 7, p. 8028–8038, 2018.

[9] Y.-T. Liu, Y.-Y. Lin, S.-L. Wu, C.-H. Chuang, M. Prasad e C.-T. Lin, «EEG-based Driving Fatigue Prediction System Using Functional-link-based Fuzzy Neural Network,» in *International Joint Conference on Neural Networks (IJCNN)*, 2014.

[10] D. Wei e H. Liu, «Analysis of Asymmetric Driving Behavior Using a Self-Learning Approach,» *Transportation Research. Part B: Methodological,* vol. 47, pp. 1-14, 2013.

[11] G. Castignani, T. Derrmann, R. Frank, and T. Engel, «Smartphone-based adaptive driving maneuver detection: A large-scale evaluation study,» *IEEE Trans. Intell. Transp. Syst.,* vol. 18, n. 9, p. 2330–2339, 2017.

[12] L. Eboli, G.Mazzulla, and G. Pungillo, «Combining speed and acceleration to define car users' safe or unsafe driving behaviour,» *Transp. Res. Part C, Emerg. Technol.,* vol. 68, pp. 113-125, 2016.

[13] G. Zylius, «Investigation of route-independent aggressive and safe driving features obtained from accelerometer signals,» *IEEE Intell. Transp. Syst. Mag.,* vol. 9, n. 2, p. 103–113, 2017.

[14] M. U. Ahmed and S. Begum, «Convolutional neural network for driving maneuver identification based on inertial measurement unit (IMU) and global positioning system (GPS),» *Front. Sustain. Cities,* vol. 23, n. 1, p. 34, 2020.

[15] A. Yuksel and S. Atmaca, «Driver's black box: A system for driver risk assessment using machine learning and fuzzy logic,,» *J. Intell. Transp. Syst.,,* vol. 25, n. 5, pp. 482-500, 2021.

[16] C. Marks, A. Jahangiri e S. G. Machiani, «Iterative DBSCAN (I-DBSCAN) to identify aggressive driving behaviors within unlabeled real-world driving data,» *Proceedings of the 22nd Intelligent Transportation Systems Conference,* 2019.

[17] D. Unzueta, «Unsupervised Learning: K-Means Clustering,» 5 April 2022. [Online]. Available: https://towardsdatascience.com/unsupervised-learning-k-means-clustering-6fd72393573c.

[18] X. Wang, A. J. Khattak, J. Liu, G. Masghati-Amoli, and S. Son, «What is the level of volatility in instantaneous driving decisions?,» *Transportation Research Part C: Emerging Technologies,* vol. 58, p. 413–427, 2015.

[19] A. Jahangiri, S. G. Machiani, and V. Balali, «Big data exploration to examine aggressive driving behavior in the era of smart cities,» in *Data Analytics For Smart Cities*, Boca Raton, FL, USA, CRC Press, Taylor & Francis Group, 2019, p. 163–182.

[20] J. S. M. E. H. P. K. a. X. X. Erich Schubert, «DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN,» *ACM Trans. Database Syst.,* vol. 42, n. 3, 2017.

[21] K. Beyer, J. Goldstein, R. Ramakrishnan e U. Shaft, «When Is "Nearest Neighbor" Meaningful?,» in *Database Theory - ICDT'99*, Jerusalem, Israel, 1999.

[22] J. Sander, M. Ester, H.-P. Kriegel e X. Xu, «Density-based clustering in spatial databases:,» in *Data Mining and Knowledge Discovery volume 2*, 1998, p. 169–194.

[23] L. Kuncheva e C. Whitaker, «Pattern Recognition and Classification,» in *Wiley StatsRef: Statistics Reference Online*, 2015, pp. 1-7.

[24] A. R. Webb, K. D. Copsey e G. Cawley, in *Statistical Pattern Recognition*, John Wiley & Sons, Incorporated, 2011.

[25] N. A. Khafaf e A. El-Hag, «Bayesian regularization of neural network to predict leakage current in a salt fog environment,» *IEEE Transactions on Dielectrics and Electrical Insulation,* vol. 25, n. 2, pp. 686-693, 2018.

[26] T. Toledo, H. Koutsopoulos e M. Ben-Akiva, «Integrated driving behavior modeling,» *Transp. Res. C Emerg. Technol.,* pp. 96-112, 15 2007.

AKNOWLEDGEMENTS

A conclusione di questo elaborato, desidero menzionare tutte le persone, senza le quali questo lavoro di tesi non esisterebbe.

Ringrazio il mio relatore Angelo Bonfitto, che in questi mesi di lavoro, ha saputo guidarmi, con suggerimenti pratici, nelle ricerche e nella stesura dell'elaborato.

Ringrazio di cuore tutta la mia famiglia, mia mamma Giulia, Nino, che è stato per me come un padre; le mie sorelle Martina; Marisa e Cinzia; mio nipote Nicolò e Stefano; le nonne Giovanna e Rita. Grazie per avermi sempre sostenuto e per avermi permesso di portare a termine gli studi universitari.

Vorrei ringraziare la mia fidanzata Giulia, che con il suo continuo amore e supporto mi ha permesso di compiere questo lungo percorso.

Un ringraziamento particolare va a Shailesh Hegde che ha contribuito con le sue idee alla riuscita di questo elaborato.

Nella mia vita ho avuto la fortuna di avere accanto una seconda famiglia, quella composta dai miei amici. Non importa se siamo cresciuti insieme, o ci conosciamo da poco tempo perché ognuno di loro a modo suo ha avuto un peso determinante nella mia crescita. E se sono potuto arrivare fin qui, lo devo anche a tutte le esperienze che abbiamo condiviso. Dunque, il minimo che posso fare è dire grazie.

Ed in fine voglio rivolgere un ultimo ringraziamento al gruppo FKSKK per avermi supportato e sopportato in questo percorso universitario.