


Doctoral Dissertation
Doctoral Program in Electrical, Electronics and Communications Engineering
(31^{st} cycle)

Smartphone Based Applications for Road Traffic Telematics

Waqar Hassan

* * * * * *

Supervisor
Prof. Guido Albertengo

Doctoral Examination Committee:
Prof. Andrea Fumagalli, Referee, University of Texas at Dallas
Prof. Renato Lo Cigno, Referee, Università degli Studi di Trento
Laura Coconea, PhD, Swarco Mizar
Prof. Paolo Giaccone, Politecnico di Torino
Prof. Maurizio Munafò, Politecnico di Torino

Politecnico di Torino
November 6, 2019

I hereby declare that the contents and organisation of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

. . . . . .

Waqar Hassan

Turin, November 6, 2019

Summary

This research is concerned with Intelligent Transportation System (ITS), with two major points of focus. The first point is the collection and processing of data from road traffic using smartphones and other devices (such as On-Board Units (OBUs)). It aims at using smartphones that are ubiquitous and host various sensors whilst providing several communication interfaces, to collect data and to investigate the possible beneficial applications of such data for traffic management, awareness and safety. The collected data is of high value for relevant authorities such as city management, public transportation system, traffic police, vehicle insurance companies, etc. The second point is the use of machine learning techniques to predict the intensity of traffic using crowd-sourced data from a small segment of the traffic. The processed information about traffic intensity can improve the accuracy of Advanced Traffic Management System (ATMS) and reduce the costs incurred due to the use of dedicated traffic sensing hardware.

A number of facts motivated this work, some of which are as follows. First, the high demand for mobility: whereby more than 70% of all journeys are made by a car in the European Union (EU). Specifically in Italy, where the number of vehicles per 1000 inhabitants is 625, which places it the 2nd highest across the EU. Second, the cost of traffic congestion: one consequence of high motorisation rates is traffic congestion, which costs about € 100 billion in the EU every year. Third, the growing adoption rates of smartphones: smartphones are omnipresent and well-connected; almost 80% of Internet users in the EU surfed via a mobile device in 2016; the average Penetration Rate (PR) of smartphones in the EU is quite high at about 67.3% of its population. Fourth, the versatility of high-resolution and high-quality mobile sensor data: smartphones have high computational power, high capacity connectivity, and low-power Inertial Measurement Unit (IMU) which enable them to be orientation-aware with minimal power consumption. Nearly every single smartphone produced in the last decade has a 6 or 9-axis IMU built-in. And finally, cloud-based data crowdsourcing trend: thanks to affordable Internet connectivity, cloud-based crowdsourcing for data has opened new doors for providing rich services to users without any dedicated hardware for data collection.

To further elaborate on the first point, ITS applications that require data related to vehicle dynamics (e.g. acceleration, yaw angle, etc.) usually have low PR

due to physical constraints on the placement of the smartphone. This work proposes a procedure, which is the first of its kind, to convert any measure taken in real-time by a smartphone sensor into the vehicle coordinate system called Vehicle Dynamics Data Acquisition (VDDA). It uses information from Global Positioning System (GPS) and low-power IMU (accelerometer and magnetometer). The results are reasonably accurate with a very high increase in usability which is a factor of paramount importance for customer-oriented applications. This allows the use of ITS related applications by drivers/passengers without any constraints on the placement of their devices, which significantly improves the PR of such applications. This approach is embedded into a highly modular and customisable vehicle data acquisition and processing system called Vehicle Data Acquisition Platform (VDAP). Using VDAP and VDDA, an Android application called Driving Style Analysis (DSA) was implemented to collect sensor data in real-time then to process it to provide driving behaviour information to users. It does so by recognising driving events such as left/right turns, accelerations, decelerations, lateral accelerations, and stops by a freely-placed smartphone in the vehicle.

For what concerns the second point, Adaptive Traffic Control Systems (ATCSs) are crucial for smart cities; the data source for these control systems has mainly been conventional induction loops which are expensive to instal and maintain. In this work, a software-based mechanism for real-time road traffic sensing called Virtual Induction Loop (VIL) was devised to replace or complement real induction loops providing a nearly perfect accuracy assuming 100% PR of the technology. The feasibility of the approach was demonstrated along with a practical integration scheme to allow Urban Traffic Control (UTC) systems to benefit from VIL. Extensive tests on real traffic patterns in the city of Turin showed that Deep Learning algorithm can be used to forecast the intensity of traffic with a higher accuracy (approx. 95%) and lower complexity as reported in the literature. The results are not only accurate, but they have significant applications including optimisation of traffic light control programme and dissemination of forecasted congestion, etc. To overcome the possible low PR of VIL, extensive modelling, simulation and validation were performed to incorporate the concept of VIL with the benefits of Deep Learning. A detailed simulation of a real intersection in the City of Turin was conducted with real traffic flows in SUMO for traffic forecasting based on VIL for traffic sensing and Gradient Boosted Machine (GBM) for traffic modelling and prediction. Extensive tests with diverse scenarios and different types of data sets were conducted to replicate a real day's traffic situation and prediction. The system can achieve very high classification accuracy, up to 95% with a very low PR of 10%. Furthermore, a single trained machine can forecast the intensity of traffic at a high-resolution with roughly 80% accuracy with a varying PR from 1% to 10%. Moreover, tests showed that during the training phase of the real system, VIL data can be collected only once at a fixed PR, afterwards, lower PRs could be derived from it to make the system feasible.

Acknowledgements

I would like to express my deep gratitude to Prof. Guido Albertengo, my research supervisor, for his patient guidance, enthusiastic encouragement and useful critiques of this research work. I would also like to extend my appreciation to Prof. Marco Mellia, Prof. Paolo Giaccone and Prof. Fabio Dovis from Politecnico di Torino, for their advice and assistance. Moreover, Mr Andrea Bragagnini and Mr Roberto Quasso from Telecom Italia encouraged me to continue with this research.
Furthermore, I wish to thank my wife for her continued support and encouragement throughout my study, none of this would have been possible without her support.
Lastly, I would like to appreciate Ms Noha Selim's time and effort for proofreading.

to my loving wife...

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This research lies in the Intelligent Transportation System (ITS) domain with two major points of focus. The aim is to answer two questions; 1) Can smartphones be considered as reliable sources of data to solve traffic problems in modern smart cities? 2) Can recent advances in Machine Learning be beneficial for utilising and enriching crowd-sourced data for traffic forecasting? To achieve this, Chapter 2 proposes a mechanism to relate the dynamics of a smartphone with that of a vehicle. Chapter 3 introduces a generic framework for sensor data collection and processing. Based on the framework, a working example for analysis of driving style is described in Chapter 4. Chapter 5 presents a novel and simple yet accurate software-based traffic sensing solution. The feasibility of using Machine Learning for traffic forecasting is explored in Chapter 6. Chapter 7 joins the previously discussed concepts and validates them using simulations on real traffic and real road networks.

The motivations behind this work can be grouped into 3 major points:

1. Demand for mobility and consequential congestion
2. Adoption rates and ubiquity of smartphones
3. Versatility of high-resolution mobile data and the phenomenon of crowdsourcing

1.1 Mobility demand and traffic congestion

People are on the move more than ever in the world, but particularly in the European Union (EU). In the EU, more than 70% of all journeys are made by a car (be it a private car, a taxi or a car-sharing service). It is evident that mobility is becoming more and more essential and vital due to ever-increasing distances. Due to the growing urban population and consequently growing urban cities, distances between home, work, schools/colleges/universities, and other facilities are

increasing. It is nearly impossible to participate in social and economic life without adequate means of personal mobility for many people. All of these factors are having a huge impact on rates of ownership of vehicles across the globe. For example, in the EU the average number of vehicles per 1000 inhabitants (also known as *motorisation rate*) is 505 [14]. This figure goes as high as 662 in the case of Luxembourg. In Italy, the number of vehicles per 1000 inhabitants is 625, which is the 2nd highest across Europe. In modern urban cities, people may rely on taxis and car-sharing/on-demand services, but the problem remains more or less the same. According to the International Energy Agency, the global number of cars on the road will nearly double by 2040 [1].

On the other hand, the quality and quantity of roads and other infrastructure remain more or less the same if not worse. Quality of roads network based on a survey by the World Economic Forum, using a scale from 1 (extremely underdeveloped) to 7 (extensive and efficient) reports that the score for EU is 4.76, while that of Italy is 4.4. Although the index of connectivity is high at 84%, the quality of this infrastructure is decreasing, currently at 56.4% [66].

Increasing urban population, increasing motorisation rates, decreasing quality and relative quantity of roads is creating more and more traffic congestion. Literature presents two main approaches to the measurement of the total costs of traffic congestion [54]. The first approach uses a modelling framework in which actual traffic conditions are compared with theoretical "free-flow" conditions, where there is no congestion what-so-ever. The second approach utilises data on actual traffic delays often based on police reports. Since the latter approach is based on major documented delays and traffic jams it is likely to yield rather lower estimates of the total costs of congestion than is the first approach.

Regardless of how it is estimated, traffic congestion is a very costly problem. According to the EU, it costs about 100 billion € every year [12]. This amount is roughly equal to 1% of the EU's Gross Domestic Product (GDP). The bigger problem is that traffic congestion is not expected to decrease, rather it is expected to increase. By 2050, the cost due to traffic congestion is expected to increase by about 50% [12]. The problem of traffic congestion is spread all over the world. Figure 1.1 shows a brief summary of peak time lost (in the year 2017) in traffic congestion around major European cities. Across Europe, drivers may spend up to 30% of driving time during peak hours in traffic congestions. Even in comparatively smaller cities, like Turin, this number is as high as 13%.

1.2 Smartphone adoption rates

As of 2018, there are about 2.53 billion smartphone users worldwide. This clearly makes smartphones one of the most ubiquitous and omnipresent modern technologies. Moreover, the number of smartphones is rapidly increasing and the

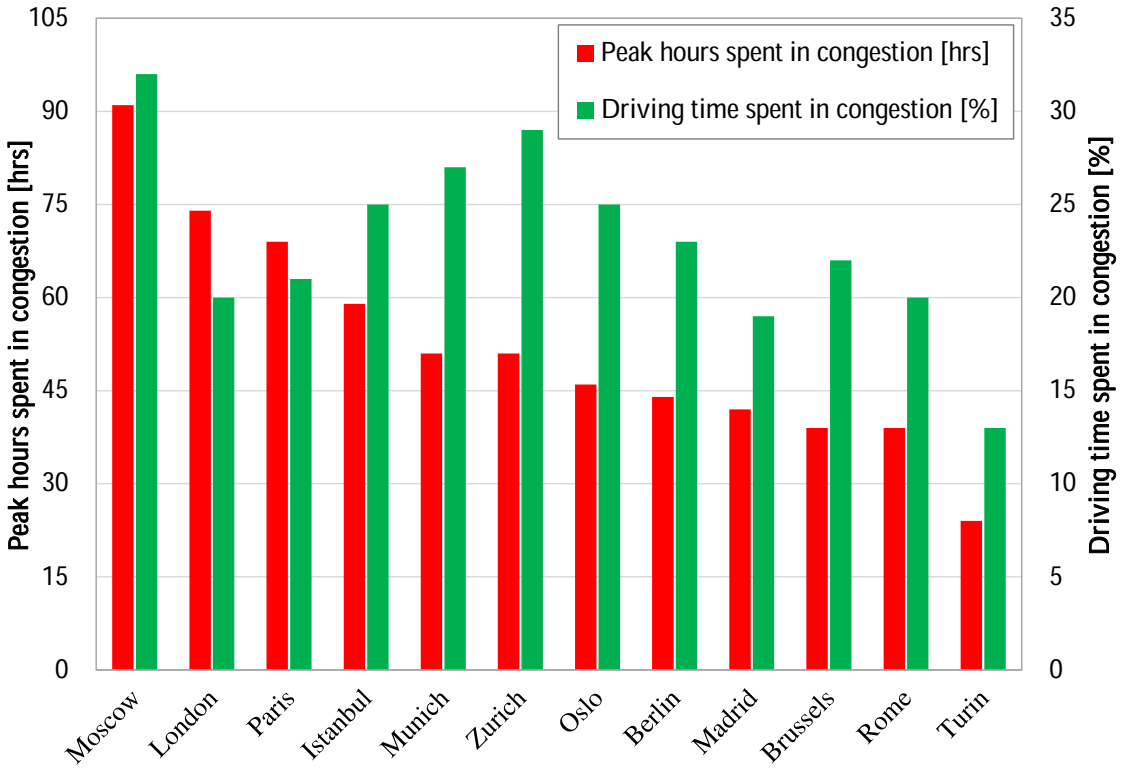


Figure 1.1: Peak time lost (in the year 2017) in traffic congestion around major European metropolitan cities [50].

same for non-smartphones (also called *dumbphones*) is decreasing. We are living in an era where all of us have a device in our pockets which, compared to super-computers and media centres from the previous decade, is more powerful, has a faster mobile broadband connection, and is more contextually aware of its physical surroundings. Almost 8 out of 10 Internet users in the EU surfed via a mobile or smartphone in 2016 [13]. At year-end 2017, there were 465 million unique mobile subscribers in Europe alone, equivalent to 85% of its population [36]. By 2020, smartphones will account for 76% of all mobile connections, up from 65% in 2016 [36]. Smartphones are truly universal all around the world, especially in Europe. Figure 1.2 shows number of smartphone users and its penetration rates for European countries. The average penetration rate of smartphones in the EU is about 67.3% of its population. European countries like Germany, the Netherlands, Sweden and the United Kingdom have a smartphone penetration rate of roughly 80%.

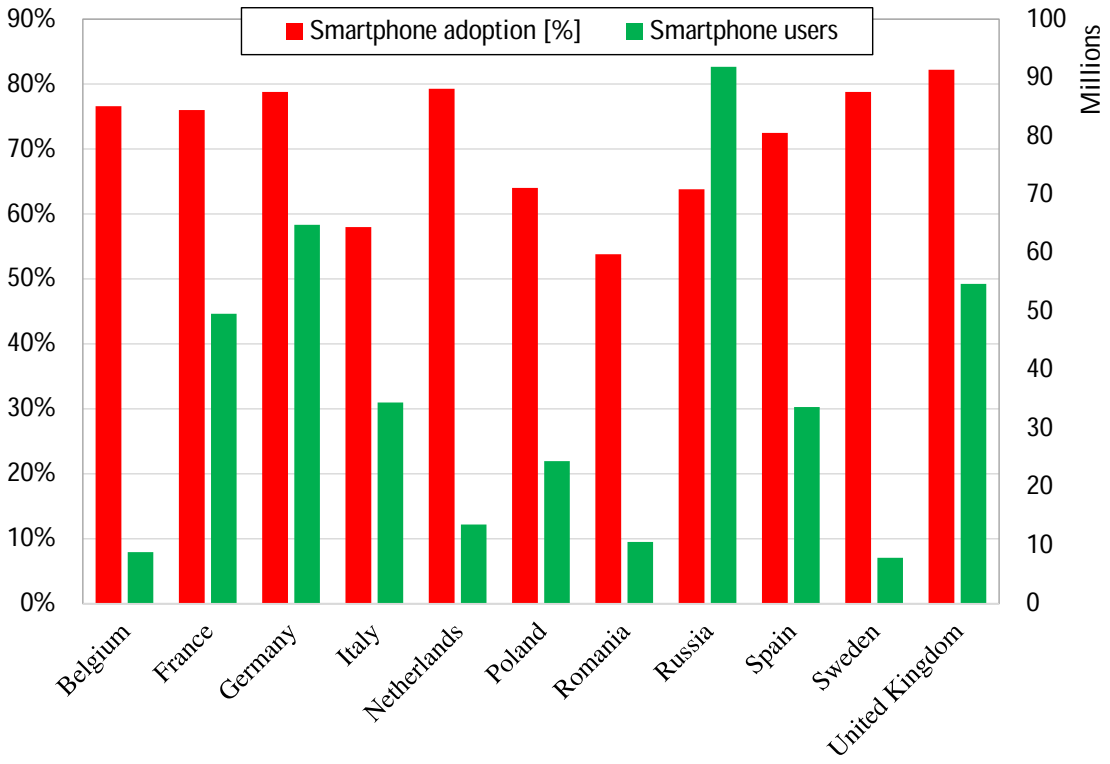


Figure 1.2: Top European countries by Smartphone Users and Penetration as per the entire population [60].

1.3 High quality mobile data

Modern smartphones pack more computation power than yesteryear's computers. They have a lot of advanced high-speed hardware to process computationally intensive tasks. As of 2019, most of the smartphone released this year have octa-core or quad-core Central Processing Units (CPUs) with 12 GB to 4 GB of RAM. Multiple low-power sensors allow them to be physically and contextually aware of their surroundings and environment. The most common sensors included in Android smartphones are listed in Table 1.1. Low-power Micro-Electro-Mechanical System (MEMS) based Inertial Measurement Unit (IMU) enable them to be orientation aware with minimal power consumption. The most commonly used sensors for the context of this research are Global Positioning System (GPS), accelerometer, gyroscope, and magnetometer. These sensors allow smartphones to have 3 to 9 degrees of freedom. Most Android have built-in sensors that are capable of measuring motion, orientation, and various environmental conditions. These sensors are capable of providing raw data with high precision and accuracy, monitoring three-dimensional device movement or positioning, or monitoring changes in the

Sensor category	Name	Output	Common usage
Motion sensors	Accelerometer	Acceleration force in m/s^2	Motion detection (shake, tilt, etc.)
	Gyroscope	Rate of rotation in rad/s	Rotation detection (spin, turn, etc.)
Environmental sensors	Barometer	Ambient air pressure in hPa or mbar	Monitoring air pressure changes
	Photometer	Ambient light level (illumination) in lx	Controlling screen brightness
	Thermometer	Ambient room temperature in °C	Monitoring temperatures
	Humidity sensor	Relative ambient humidity in %	Monitoring dew point, absolute/relative humidity
	Microphone	Ambient sound level in dB	Noise level, voice recording
Position sensors	GPS	Location in latitude and longitude	Navigation, location-based services
	Magnetometer	Ambient geomagnetic field in µT	Compass and orientation

Table 1.1: Sensor types, output data and usage [32].

ambient environment near a device. All of these factors enable data recorded from smartphones to be of more than decent quality.

1.3.1 Data crowdsourcing

Crowdsourcing is *the practice of obtaining information or input into a task or project by enlisting the services of a large number of people, either paid or unpaid, typically via the Internet* according to the Oxford dictionary. Cloud-based crowdsourcing for data has opened new doors for providing rich services to users without any dedicated hardware for data collection. This approach consists of building large data sets (real-time or not) with the help of a large group of people using a medium such as a smartphone application. This phenomenon is of paramount importance in

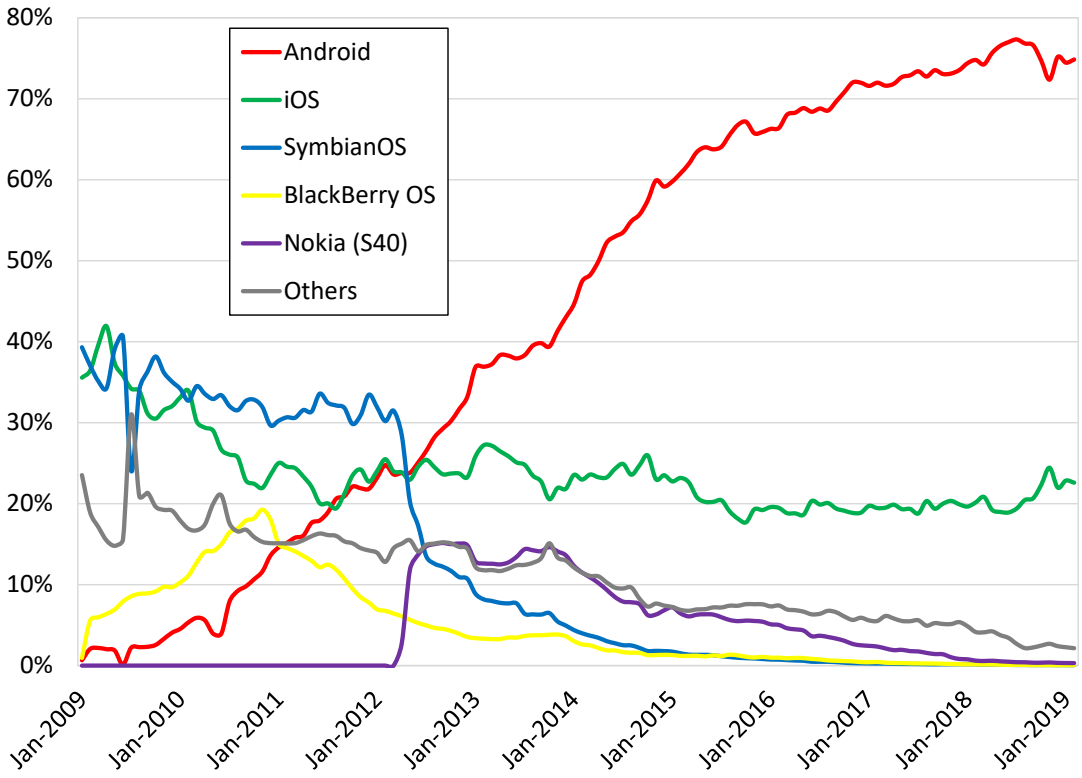


Figure 1.3: Mobile Operating System (OS) worldwide market share in percentage [20].

modern Smart Cities. Apart from significantly reducing data collection hardware installation and maintenance costs, it also dramatically improves the smart city dynamics because of collaboration and citizen engagement.

1.3.2 Mobile OS market share

As of 2019, there are two major competitors in the mobile OS category in the market, namely Google's Android and Apple's iOS. For all the work that follows we opted to work with Android OS. The reason for this technical choice is twofold. Firstly, the market share of mobile OS shows that Android is the dominant mobile OS with a market share of roughly 75% while iOS has a share of only roughly 22% [20]. This implies that an application that supports Android OS can potentially be used by 75% of the market. Secondly, Apple's iOS has some developer restrictions that do not allow the use of background activities. This is a major hindrance for an application that needs to collect sensor data in the background seamlessly. Figure 1.3 shows the worldwide market share of major mobile OS from 2009 till 2019. Mobile OS market share forecast is not clear at the moment, but in

any case, Android is expected to maintain its market share (if not increase) [49].

Chapter 2

Vehicle Dynamics Data Acquisition (VDDA)

The ubiquity of smartphones, with sensors, high processing power, and high bandwidth connectivity, makes them ideal candidates for hosting ITS applications such as those described in [84], [80] and [44]. Smartphone sensors use two reference systems: 1) *smartphone's* (referred by accelerometer and gyroscope); 2) *Earth's* (referred by GPS). Therefore, the smartphone reference system and the Earth reference system must be related to each other to jointly use the measurements of both groups [11]. In ITS applications, the relationship between smartphone and vehicle coordinate systems is needed in case the smartphone must detect some vehicle dynamics. The determination of this relationship can be, in many cases, a serious impairment to the usability of smartphones, since whenever applications require data related to the vehicle dynamics, only two approaches are possible: 1) to fix the position and orientation of the smartphone in the vehicle; 2) to recalculate the orientation of the smartphone with respect to vehicle any time data are collected. The first solution is the simplest one but, in my opinion, few customers will be willing to use applications requiring to place their smartphones in a cradle screwed on the dashboard of their car. The second solution is still under investigation and some solutions are already available with some limitations in their usage. My proposal is a simple real-time procedure to convert measurements from smartphone sensors into the vehicle coordinate system. This allows complete freedom of movement of the smartphone while overcoming most of the limitations of current proposals, as it will be discussed in the following sections.

Available solutions

A detailed survey and review of the major smartphone to vehicle alignment techniques have been presented by [76]. Some prominent solutions are summarised here as well. In Mobile sensor platform for Intelligent Recognition of Aggressive

Driving (MIROAD) [51] the authors propose putting the smartphone in a cradle to fix its orientation with respect to the vehicle. As already pointed out that this solution is impractical since it reduces the convenience of the driver and requires the installation of a cradle in the vehicle. Other authors such as [4] and [56] suggest collecting the steering wheel angle from the Controller Area Network (CAN) bus of a vehicle using an On-Board Diagnostics (OBD) module. This method is invasive since it requires connecting additional hardware to the vehicle which is not always applicable. Moreover, this particular sensor might not be accessible from the OBD-II in all vehicles and, when accessible, usually requires the full knowledge of the proprietary communication protocol used by the car manufacturer to retrieve these data from the OBD-II interface. This makes the implementation of this solution even more difficult.

The *full auto-calibration* proposal described in [2] uses the smartphone accelerometer, gyroscope, and GPS, and translates these measures into the vehicle coordinate system using the yaw angle between the vehicle and the smartphone. Its value is calculated by forcing the vehicle to move forward, with no lateral acceleration, until its longitudinal axis is identified. Nericell project by Microsoft Research [59] also leverages on the processing of acceleration data to find out the device orientation: it also requires the vehicle to brake and travel in a straight line to generate a recognisable acceleration in a known direction. In both cases, after this initial setting phase, the orientation of the smartphone with respect to the vehicle must be kept unchanged, which implies the use of a cradle, although in this case, it could be a removable one.

Another solution based on an accelerometer, magnetometer and GPS [75] has a typical settling time of 60 seconds. Although the authors claim that the smartphone does not necessarily have to be fixed with respect to the vehicle, according to the authors' experience, convergence is difficult to achieve when the smartphone is free to move with respect to the vehicle (which commonly happens when the smartphone is in the driver's pocket or is handheld by its user) and the settling time is high.

My approach

In my opinion, for non-safety related ITS applications, the flexibility and comfort of the driver should take precedence over the accuracy of the measurements: forcing the driver to place his/her smartphone in a cradle to fix its orientation with respect to the vehicle would have a negative impact on the usability of the applications and consequently on their Penetration Rate (PR). This is becoming even more evident in newer cars, in which the smartphone can be connected to the car's entertainment system via Universal Serial Bus (USB) to get access to many smartphone applications through voice, steering wheel commands and car's dashboard display. Any application that relies on the user putting their device in a cradle, will either not function at all if the user does not put their device in the cradle or

will be highly inaccurate. Also, the accuracy of sensors (especially magnetometers) in smartphones can be greatly degraded by magnetic cradles and modern cradles with wireless charging facility. Therefore, applications that can work seamlessly and without any unnecessary user interaction would likely be more acceptable. As a result, a trade-off between accuracy and usability must be found.

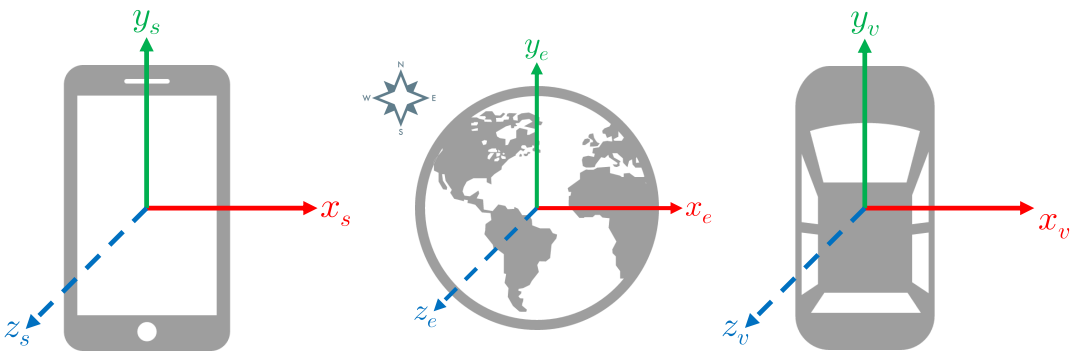
In this Chapter, I propose a completely flexible and adaptive solution called Vehicle Dynamics Data Acquisition (VDDA): using data from the accelerometer, magnetometer, and GPS our solution computes the orientation of the smartphone with respect to the vehicle in real-time and in the smartphone itself. The driver is not forced to place the smartphone in a cradle to fix its orientation with respect to the vehicle, thanks to its very low settling time (tens of milliseconds) which allows the smartphone to change its orientation with respect to the vehicle while data are collected. Such movements always occur when the smartphone is in the pocket of the driver which is in turn lightly, but constantly, moving while driving or when the smartphone is placed somewhere close to the driver but is not in a cradle screwed to the vehicle. My solution is flexible in terms of the placement of the smartphone inside the vehicle and hardware compatibility (with roughly all Android device). The smartphone can be present in the pocket the driver, placed on the seat/dashboard or even placed in a cradle. It is adaptive thanks to its responsiveness allowing it to adapt very quickly to the new orientation changes.

2.1 Methodology

The goal to be reached is the identification of the rotation matrix values to convert between the coordinate systems of the smartphone and the vehicle in a way fast enough to be repeated whenever it is needed. I propose a two-step procedure able to convert any measure taken by the smartphone into the vehicle reference system in a few milliseconds using data sampled almost simultaneously to perform all calculations. To describe it, firstly the coordinate systems involved are defined, and secondly the remapping of any measure taken in its own reference system onto that of the vehicle is shown.

2.1.1 Coordinate system definitions

The three reference systems we are dealing with are the one of the smartphone, the one related to Earth and, finally, the one of the vehicle. They are described in the Table 2.1 and Figure 2.1. Please note that all three coordinate systems use the same axes for rotational movements as they do for linear or directional movements. Rotation is positive in the counter-clockwise direction; that is, an observer looking from some positive location on the x, y or z-axis at a device positioned on the



(a) relative to a smartphone

(b) relative to Earth

(c) relative to a vehicle

Figure 2.1: Coordinate system definitions relative to smartphone (a), Earth (b), and vehicle (c).

Coordinate system	**x-axis**	**y-axis**	**z-axis**
Smartphone	(x_s) pointing to the right wrt the surface of the display	(y_s) pointing upwards wrt the surface of the display	(z_s) pointing vertically outwards wrt the surface of the display
Earth	(x_e) tangential to the ground at the current location and pointing towards East	(y_e) tangential to the ground at the current location and pointing towards North	(z_e) perpendicular to the ground and pointing towards the zenith
Vehicle	(x_v) pointing laterally towards the side of the vehicle in right direction when viewing from the top	(y_v) pointing towards the front of the vehicle when viewing from the top	(z_v) pointing outwards vertically from the vehicle's roof

Table 2.1: Coordinate system definitions relative to smartphone, Earth, and vehicle.

origin would report positive rotation if the device appeared to be rotating counter-clockwise. In case the coordinate system refers to a tablet device, it is based on landscape orientation rather than portrait since the coordinate system is always based on the natural orientation of the device.

2.1.2 Axis Remapping

The coordinate system remapping is done by periodically performing two operations in sequence, namely a 3D rotation followed by a 2D rotation.

1. In the first step, the sensor data are converted from smartphone to Earth coordinate system using Equation (2.1). The 3 x 3 rotation matrix R_{SE} is computed using data from accelerometer and magnetometer as per Equation (2.2g). Notice that modern smartphone operating systems (including Android and iOS) provide two types of sensor data *i.e.* calibrated and uncalibrated. Corrections (such as temperature compensation, bias compensation, scale calibration and drift) have been eliminated from calibrated sensor data. To minimise the effect of sensor biases, calibrated sensor data is used.

$$\underbrace{\begin{bmatrix} x_e \\ y_e \\ z_e \end{bmatrix}}_{\text{relative to Earth}} = R_{SE} \underbrace{\begin{bmatrix} x_s \\ y_s \\ z_s \end{bmatrix}}_{\text{relative to smartphone}} \tag{2.1}$$

$$\vec{g} = \begin{pmatrix} g_x, & g_y, & g_z \end{pmatrix} \tag{2.2a}$$

$$\vec{n} = \begin{pmatrix} n_x, & n_y, & n_z \end{pmatrix} \tag{2.2b}$$

$$\vec{u} = \vec{n} \times \vec{g} \tag{2.2c}$$

$$\hat{g} = \frac{\vec{g}}{\|\vec{g}\|} \tag{2.2d}$$

$$\hat{u} = \frac{\vec{u}}{\|\vec{u}\|} \tag{2.2e}$$

$$\hat{v} = \hat{g} \times \hat{u} \tag{2.2f}$$

$$R_{SE} = \begin{bmatrix} \hat{u}_x & \hat{u}_y & \hat{u}_z \\ \hat{v}_x & \hat{v}_y & \hat{v}_z \\ \hat{g}_x & \hat{g}_y & \hat{g}_z \end{bmatrix} \tag{2.2g}$$

The accelerometer provides the gravity vector ($\vec{g}$) and the magnetometer provides the magnetic North vector ($\vec{n}$). A perpendicular vector ($\vec{u}$) is obtained by the cross product of gravity and North vector. Another perpendicular unit vector ($\hat{v}$) is obtained by the cross product of gravity and $\vec{u}$ unit vector. The two cross products ensure that vectors $\hat{u}$, $\hat{v}$ and $\hat{g}$ are mutually perpendicular. The 3D rotation matrix (R_{SE}) is composed of these unit vectors (*i.e.* $\hat{u}$, $\hat{v}$ and $\hat{g}$). Their orientation is shown in Figure 2.2. Notice that $\hat{u}$ and $\hat{v}$ roughly point towards West and North, respectively. Their offset depends on the relative position of the North Magnetic Pole with respect to the Geographical North Pole as seen from the smartphone location.

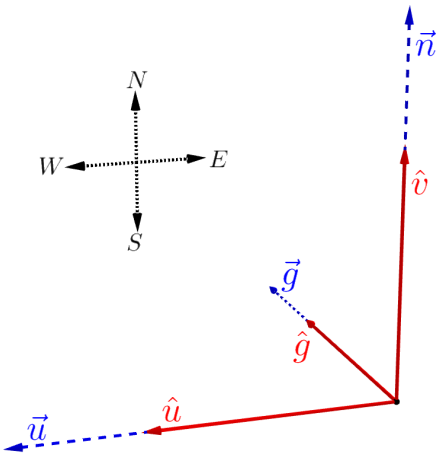


Figure 2.2: Orientation of vectors used for computation of 3D rotation matrix assuming that North Magnetic Pole and Geographic North Pole are coincident.

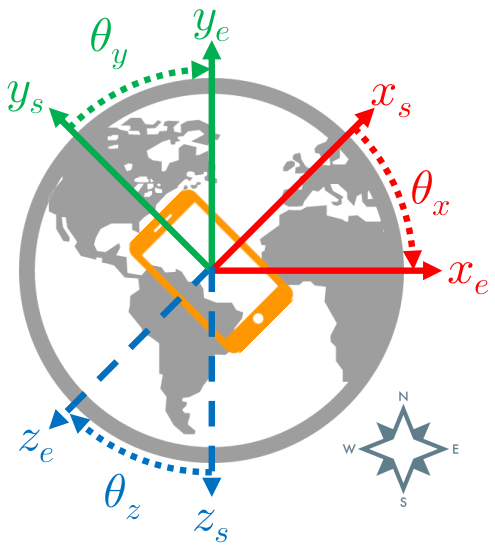


Figure 2.3: Relationship between smartphone coordinate system and Earth coordinate system.

Figure 2.3 shows the relationship between smartphone and Earth coordinate systems in the simplified case where the North Magnetic Pole and Geographic North Pole are coincident. In any real case, they are not coincident, and the North Magnetic Pole instead of the Geographic North Pole is simply used throughout all formulas. Since the smartphone (in 3D) can be placed in any orientation with respect to Earth, there are three angles of reference θ_x, θ_y and θ_z. Google's Sensor Manager Application Programming Interface (API)

is used to compute the rotation matrix [30]. Noteworthy is that the rotation matrix can also be easily computed manually as explained earlier, but here readily available API is used, which works the same way. Moreover, note that the API has some limitations as stated by the official documentation, "*The matrices returned by this function are meaningful only when the device is not free-falling and it is not close to the magnetic north*". But clearly, it is safe to say that these limitations do not affect our computations in most of the cases. The 3D rotation matrix is recomputed every time there is an update from accelerometer or magnetometer.

2. In the second step the data, already referred to the Earth coordinate system, are remapped to the vehicle reference system, as shown in Equation (2.3a) and Equation (2.3b). To perform this 2D rotation, the 2 x 2 rotation matrix R_{EV} is computed using the magnetic bearing angle (θ_b) which again refers to the North Magnetic Pole, which was used in the previous 3D rotation in place of the Geographic North Pole. The bearing is the horizontal direction of travel of the mobile device and is not related to the device's orientation.

$$\underbrace{\begin{bmatrix} x_v \\ y_v \end{bmatrix}}_{\text{relative to vehicle}} = R_{EV} \underbrace{\begin{bmatrix} x_e \\ y_e \end{bmatrix}}_{\text{relative to Earth}} \tag{2.3a}$$

$$\underbrace{z_v}_{\text{relative to vehicle}} \approx \underbrace{z_e}_{\text{relative to Earth}} \tag{2.3b}$$

To compute θ_b from the true bearing θ'_b provided by GPS, it is necessary to know the magnetic declination (θ_{md}), which is defined as the angle on the horizontal plane between the North Magnetic Pole and the Geographic North Pole. The World Magnetic Model produced by the United States National Geospatial-Intelligence Agency is used to estimate the magnetic declination anywhere on Earth based on location and time [58]. The magnetic bearing angle is finally computed using Equation (2.4a). After this, the 2 x 2 rotation matrix R_{EV} is computed using Equation (2.4b).

$$\theta_b = \theta'_b - \theta_{md} \tag{2.4a}$$

$$R_{EV} = \begin{bmatrix} \cos(\theta_b) & \sin(\theta_b) \\ -\sin(\theta_b) & \cos(\theta_b) \end{bmatrix} \tag{2.4b}$$

Figure 2.4 shows the relationship between Earth coordinate system and vehicle coordinate system. Since under normal circumstances (travelling on flat/semi-flat ground), the z-axis of Earth (z_e) and the z-axis of the vehicle (z_v)

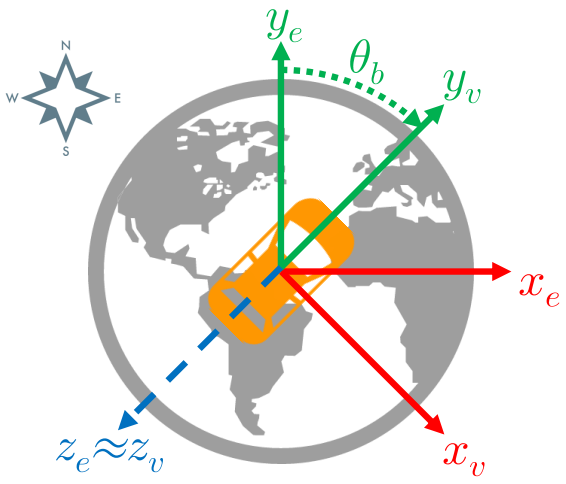


Figure 2.4: Relationship between Earth coordinate system and vehicle coordinate system.

are approximately parallel (*i.e.* assuming $z_e \approx z_v$), it will only be necessary to perform the rotation in 2D (xy plane), using the previously defined magnetic bearing angle θ_b to compute the four values of the 2D rotation matrix. This matrix is recomputed every time there is an update of the bearing from GPS.

Using the above mentioned two rotation matrices, sensor data is rotated from smartphone coordinate system to Earth coordinate system and eventually to vehicle coordinate system. This enables the mapping of the coordinate system of a smartphone on a vehicle. Note that the novelty here is not the usage of Google's API, the computation of rotation matrices or the rotation of data itself, rather the complete solution provided by the combination of simple techniques allowing complete freedom of placement of the smartphone. Modern lower power accelerometer and magnetometers can be sampled very frequently (the minimum sampling period can be as low as 20 ms [32]). GPS receivers in smartphones provide an update every single second. This enables my system to be adaptive and responsive to changes in the heading of the vehicle in the great majority of cases.

2.1.3 Bearing retention

There are some conditions under which this approach fails, such as when a vehicle moves at a very slow speed or completely stops (for example at a traffic signal) and the GPS does not provide a bearing angle until the vehicle is again in motion. In these cases, *bearing retention* is used which means to retain the last

known bearing until a new bearing is available. This works simply because for example, in case of stops, the heading direction of a vehicle before stopping and after starting again is always the same, implying that the bearing angle remains unchanged during the stop.

2.1.4 Handheld device

Another situation to be considered is when the user is physically interacting with the smartphone. In this case, the rotation matrices may not be accurate at all, since the smartphone orientation could change very quickly, faster than the update frequency of the two matrices. One solution is to identify when a user is interacting with the phone and to suspend data collection. User interactions can be detected by a keypress, a touch on the screen, proximity sensor or a phone call [59]. Another possible solution could be to compute and use time averages to provide estimates of the smartphone's average orientation. This is a potential candidate for further studies and was not explored during this work.

Notice however that newer vehicle usually provide a wired connection between the driver's smartphone and the entertainment systems (through Android Auto and/or Apple CarPlay) which makes it less likely that the smartphone is handled while driving. Furthermore, a majority of authorities from different countries have enacted laws to ban the handheld use of mobile phones while driving due to safety concerns. This should deter drivers from handling the device while driving in any case.

2.1.5 Sources of error and countermeasures

The total error associated with sensor data rotation performed using axis remapping can be expressed as:

$$error \approx n_s \pm n_{\theta_b} \pm n_{\theta_{md}}$$

where n_s is the noise in sensor readings, n_{θ_b} is the error in bearing reported by GPS and $n_{\theta_{md}}$ is the error in bearing due to magnetic declination. Notice that any measure to reduce the effect of the first two error components on my data was not applied, although there are many possible techniques to act on them. The noise in sensor readings can be removed using a low-pass filter as suggested by [51], median values over a temporal window utilised by [59] or a Kalman filter described by [5]. Smartphone sensor noise is well represented by white noise and therefore a Kalman filter is usually used to filter out such noise from sensor measurements [2]. As far as the error in the bearing reported by GPS is concerned, there are two possible solutions. First is to again use a Kalman filter which is applied very frequently in Inertial Navigation System (INS) such as [35]. The second solution is to use a snap to road technique in real-time to correct the bearing reported by GPS in

real-time. However, this may incur some performance degradation, since snap to road is usually achieved by using a remote API. The offset error in bearing due to the presence of magnetic declination can be evaluated using the World Magnetic Model and subtracted from the bearing as reported by Equation (2.4a). Notice that the noise component due to the presence of magnetic declination was eliminated since it is likely the most significant part of noise affecting data rotation.

2.2 Testbed and measurement dataset

My data rotation methodology was extensively tested in the field with multiple users and mobile devices. More than 2 million location points were collected and processed over 16 months using 14 different Android smartphones. The closed beta test included roughly 10 users in which 5 were actively participating. The testbed consisted of an Android application and a server (offering a web service and data post-processors).

- The **Android application** collected sensor data (including accelerometer and magnetometer) and location data (GPS), calculated rotation matrices and rotated data in real-time. Raw and processed data were stored locally until a Wireless Fidelity (Wi-Fi) or a mobile data connection became available. As soon as the preferred data connection was ready, locally stored data were uploaded onto the server only for the sake of estimating the accuracy of the data rotation.

- The **web service** allowed the application to upload collected data onto the server and to store them into a MySQL database.

- The **post processor** fetched the data from the database, computed snapped latitude (lat_x), longitude (lng_x) and bearing (θ_s). It used *Google Places Snap to Roads* API to snap raw GPS trails to real roads on a map (this API takes up to 100 GPS points collected along a route and returns a similar set of data with the points snapped to the most likely roads the vehicle was travelling along). The entire trip was snapped to a road in sequence by feeding the API with a maximum of 100 GPS points at a time. Equation (2.5e) was used to calculate snapped or true bearing along the road considering all consecutive sets of snapped latitudes and longitudes. In the equation lat_x and lng_x are snapped latitude and longitude respectively, $(x^\circ)^c$ refers to the conversion of degrees to radians, x° is an angle represented in degrees while x^c is an angle

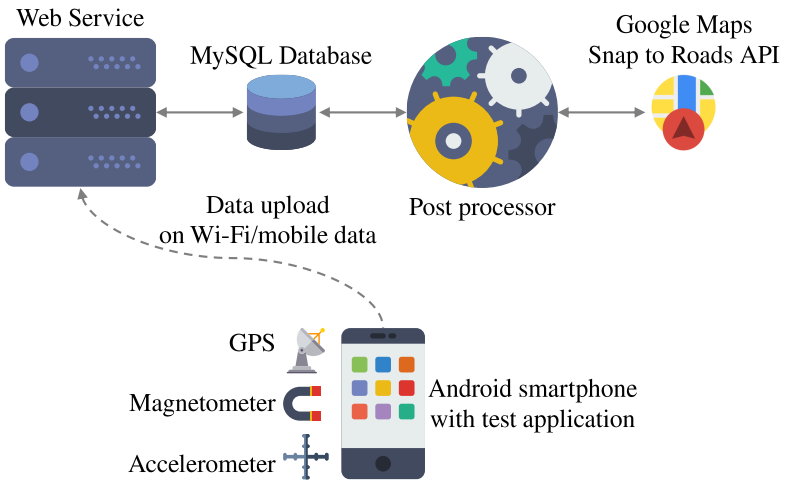


Figure 2.5: Block diagram of the testbed for the collection and processing of data.

represented in radians.

$$\Delta_{lng} = (lng_2^\circ - lng_1^\circ)^c \tag{2.5a}$$
$$b = \sin(\Delta_{lng}) \times \cos(lat_2^c) \tag{2.5b}$$
$$a = \cos(lat_1^c) \times \sin(lat_2^c) \tag{2.5c}$$
$$- \sin(lat_1^c) \times \cos(lat_2^c) \times \cos(\Delta_{lng}) \tag{2.5d}$$
$$\theta_s = ((\arctan_2(b, a))^\circ + 360^\circ) \mod 360^\circ \tag{2.5e}$$

Figure 2.5 shows a block diagram of the testbed with all components. For the sake of conformity, a filter was applied to all samples to remove all location points that have a bearing or snapped bearing equal to zero. The reason is that in Android, if the GPS location does not have a bearing associated with it then a 0.0° value is returned. Moreover, if the actual value of bearing was 0° then there would be no 2D rotation since a rotation by 0° generates an identity matrix, making the error evaluation inapplicable.

The resulting dataset contained nearly 1.5 million location points. Figure 2.6 shows a sample of dataset points plotted on a map.

Figure 2.7 shows the distribution of horizontal accuracy and speed in the input dataset as a heat map. The colour scale on the right of the figure represents the probability. μ_{speed} and σ_{speed} represent the average and standard deviation of speed while $\mu_{accuracy}$ and $\sigma_{accuracy}$ represent the average and standard deviation of accuracy. The distribution of horizontal accuracy is bimodal with peaks at 5 m and 12 m. This is most likely due to the fact that a diverse range of smartphone makes and models were used to collect the data. Overall, the majority of the location points fall in the range of 5 m to 15 m horizontal accuracy and 10 km/h to 60 km/h speed. Moreover, the hot region centred at 135 km/h speed and 4 m

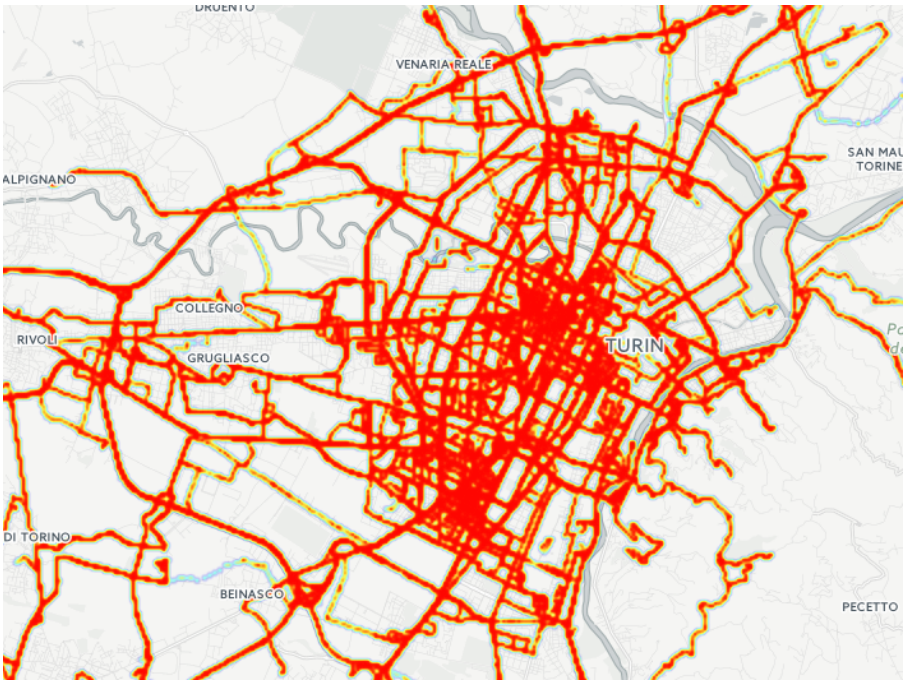


Figure 2.6: A sample of location dataset collected from the testbed based on a smartphone application and a central server. Each red dot corresponds to one collected geographical location.

accuracy represent data collected during sub-urban and intercity travels. While the two hot regions centred at 30 km/h speed and 5 m accuracy along with 30 km/h speed and 12 m accuracy represent urban trips. It is noteworthy that Android defines horizontal location accuracy as the radius of 68% confidence. Also, the accuracy estimation does not indicate the accuracy of bearing, velocity or altitude.

2.2.1 Power consumption

The proposed methodology relies on data collection from an accelerometer, magnetometer and GPS sensors. In modern smartphones, accelerometer, gyroscope and magnetometer are usually coupled together in a MEMS based lower-power or ultra-low-power IMU chip. Following are the rated current consumptions of relevant chips from a common modern smartphone (Samsung® Galaxy® S8):

- **Accelerometer and Gyroscope**: ST LSM6DSL has a nominal current consumption of 0.29 mA to 0.65 mA.
- **Magnetometer**: AKM AK09916 has an active current consumption of 1.1 mA.

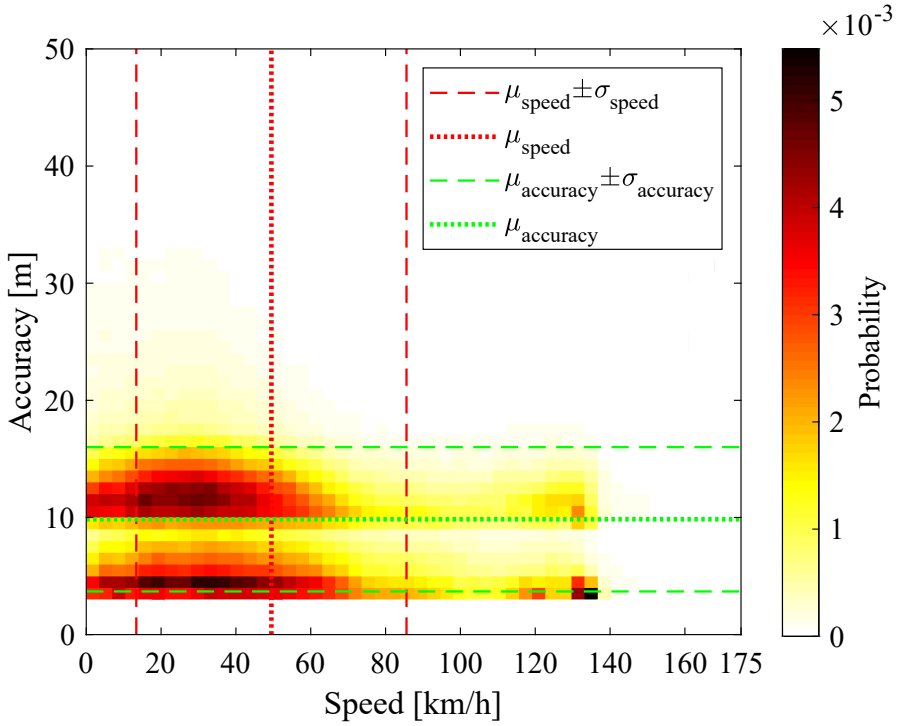


Figure 2.7: Heat map (speed vs horizontal accuracy) of location dataset collected from the testbed based on a smartphone application and a central server.

- **GPS**: Broadcom BCM4774 has an average current consumption of 10 mA to 100 mA.

It is evident from the above that GPS receivers are the biggest source of power consumption among the relevant sensors. To reduce the energy consumed by GPS receiver, I used a variable sampling period of GPS where speed and sampling period were inversely proportional. In other words, at a high speed, the application samples location with a low frequency since the bearing is expected to remain the same as a previous location point. Whereas at a low speed, the application samples location with a higher frequency since the bearing may change (for example during a turn) between subsequent location points.

2.3 Results

The possible noise in the 3D rotation matrix used during the 3D rotation is not discussed here since the matrix is computed using *Google Sensor Manager* API and investigating the accuracy of the API is beyond the scope of this Chapter. However, the error from the 2D rotation depends entirely on the accuracy of the bearing angle (θ_b), which is analysed in the following paragraphs. A bounded error

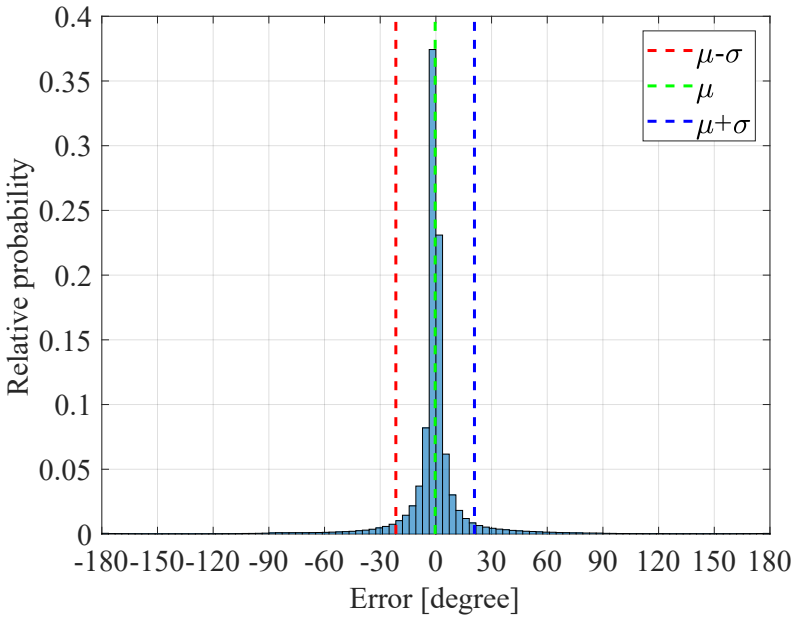


Figure 2.8: Histogram of error E in bearing θ_b highlighting average μ and standard deviation σ.

	Min	**Max**	**Average** (μ)	**Standard deviation** (σ)	**Root Mean Square Error (RMSE)**
E	-179.999	179.978	-0.417	21.073	21.077
ϵ	0	179.999	8.625	19.231	21.077

Table 2.2: Statistical parameters of error E and absolute error ϵ in degrees.

in bearing (E) was calculated for the dataset. E is bounded between the closed interval $[-180°, +180°]$. Equation (2.6b) is used to calculate the value of error E and Equation (2.6c) is for calculating absolute error ϵ in bearing θ_b.

$$\Delta = \theta_b - \theta_s \tag{2.6a}$$

$$E = \begin{cases} 360° - \Delta, & \text{if } \Delta > 180° \\ -360° - \Delta, & \text{if } \Delta < -180° \\ \Delta, & \text{otherwise} \end{cases} \tag{2.6b}$$

$$\epsilon = |E| \tag{2.6c}$$

Figure 2.8 and Table 2.2 show the distribution and statistical parameters of error E. The error E is evenly distributed around 0° with an average of $-0.417°$. The standard deviation σ is 21.073° which is also very low compared to the bounds of error E $[-180°, +180°]$. The average absolute error μ of ϵ is 8.625°.

The computation of the rotation matrix involves the calculation of trigonometric ratios (sine and cosine) of the bearing angle θ_b. This implies that an error in bearing angle will result in an error in the value of the trigonometric ratio. To estimate the worst-case scenario based on average absolute error $\omega = \max(|\sin(\theta_b) - \sin(\theta_b \pm \mu_\epsilon)|, |\cos(\theta_b) - \cos(\theta_b \pm \mu_\epsilon)|)$ is used. The worst-case results in a difference of 0.1516 in the trigonometric ratio (when θ_b is between the intervals [3°,6°], [84°,87°], [93°,96°] and [174°,177°]). Considering all possible applications, this error is either negligible or manageable.

As a comparison, my average absolute error is higher than the one for full calibration achieved by [2] (8.7° compared to 3.9°), but my solution does not require any action from the user for convergence such as braking or driving in a straight line. The IMU alignment achieved by [75] (using an accelerometer, magnetometer, and GPS) shows a typical error of 2° for each Euler angle (roll, pitch, and yaw), while reaching a steady state in 60 seconds. On average, the error in yaw angle estimation is in between 3.21° to 2.08°. My solution outperforms this latter one in terms of negligible settling time (a few milliseconds) and reaches convergence even if the smartphone is moving with respect to the vehicle. Again, this guarantees that there are no restrictions imposed on the driver, making my solution more adaptable to most ITS applications.

2.4 Conclusions

In this Chapter, I proposed a procedure to convert any measure taken by a smartphone sensor into the vehicle coordinate system in real-time. It uses information from low power IMU (accelerometer and magnetometer) and GPS to perform data conversion, applying first a 3D rotation (from smartphone to Earth coordinates) and then a 2D rotation (from Earth to vehicle coordinates). With this procedure, the driver of a vehicle is no more constrained to place their smartphone in a cradle all along a trip but can leave it in a bag, pocket or even handle it for short periods. To obtain this result, the accuracy is traded with usability; reducing the first one to increase the second one. The result is a very low penalty in accuracy, negligible in most ITS applications, and a very high increase in usability which is a factor of paramount importance for any customer-oriented application. This is the first solution, to the best of my knowledge, that can achieve real-time axis remapping with reasonable accuracy without placing any restrictions on the state of the device or driver. My approach enables the implementation of numerous ITS applications without installing dedicated hardware and using only already available mobile devices.

Chapter 3

Vehicle Data Acquisition Platform (VDAP)

This Chapter describes the research activities related to a multi-purpose software platform for the collection and processing of data from vehicles using a smartphone application. The latest advances in information technology offer new possibilities for urban mobility. Smartphone applications do not only generate new services for the users, but also generates massive data originating from the users. The collection and analysis of this data may open new doors to provide useful services to users and city managers.

ITS applications have a lot in common; All of them collect some data from the user, process it locally to some extent, transfer it to a server for extended services, and use the back-end server to provide value-added services to multiple users. During this research, I did not come across a generic and customisable platform to serve this purpose. The idea is to identify the basic building blocks needed to implement common ITS applications, design the individual blocks such that they provide maximum functionality and flexibility and implement the platform using a modular approach.

3.1 System architecture

The system is called Vehicle Data Acquisition Platform (VDAP) and essentially consists of a smartphone application platform and a server back-end. The smartphone application and server work together to provide the ITS services to users.

One of the biggest advantages of the system is its modularity, which allows it to be highly customisable and optimisable for the specific application scenario. The platform may use a simple eXtensible Markup Language (XML) based configuration for all units and modules. This section describes the platform architecture in detail.

3.1.1 Application platform

The smartphone application platform can be used to create a variety of meaningful applications for ITS scenario. The smartphone application serves three basic purposes:

- data acquisition
- initial processing
- data upload

After acquisition and initial processing, some service may be provided to the user directly. For other more complicated services, the data collected by the application is uploaded to the server, which in turn processes it and provides other services. For the context of this Chapter, the application will refer to an Android application.

Overview

The application platform consists of 3 basic modular units.

1. Control unit
2. Trigger unit
3. Data collection and processing unit

Figure 3.1 shows the generic relationship between the basic building blocks of the application platform. The control unit provides basic control functionality. The trigger unit generates stimulus to start and/or stop a vehicle trip. The data collection and processing unit is responsible for acquiring data from a smartphone to process them and transfer them to a server (if needed). In the following sections, individual units of the application platform are described.

Control unit

The control unit acts as the main control hub of the platform. It offers services such as background always-on service, communication, data storage and database management. Figure 3.2 shows some basic modules of the control unit. The *data storage module* handles all storage requests made by other modules (such as data collection and processing modules). It enables unified access to the storage medium and provides services such as creating, deleting and appending data file(s).

The *communication module* caters for all communication-related needs of the platform. The options for possible communication techniques are available in Section 3.1.2. The choice of communication technique depends on the type and amount

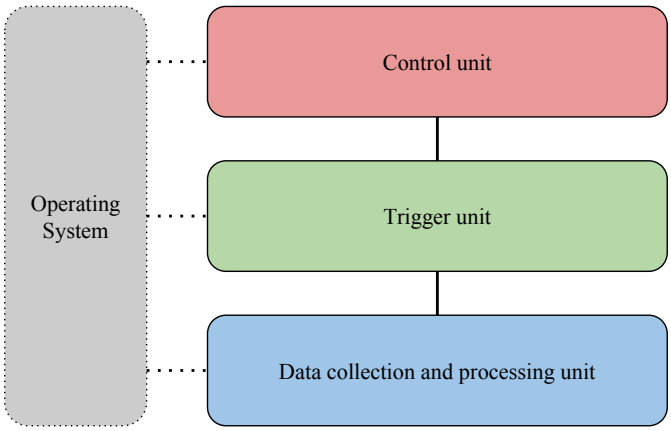


Figure 3.1: A high-level block diagram showing an overview of the application platform.

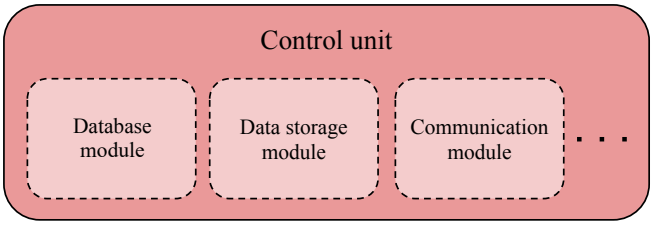


Figure 3.2: A block diagram of the control unit emphasising important modules.

of data to be sent. As an example, a short real-time location update can use a simple light-weight JavaScript Object Notation (JSON) encapsulated RESTful web service. Whereas HyperText Markup Language (HTML) multipart form-data can be used to upload large trip files. Furthermore, the module offers the possibility to use Wi-Fi only or Wi-Fi and mobile data connection options for data transfer.

The *database module* provides database management services for lower level modules. The database contains a common section and an application specific portion. A common shared table between all applications can be the trip list table or a device to vehicle association table. The unit allows applications to create their own tables and perform queries. The database can be hosted using native SQLite database or Firebase Realtime Database built using NoSQL database.

Trigger unit

The trigger unit provides multiple automatic triggers for automatic recognition of a vehicle trip. It uses multiple APIs to achieve this functionality. Figure 3.3

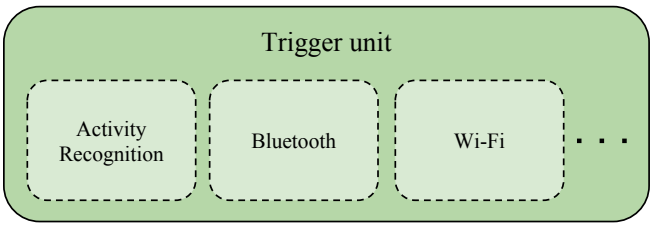


Figure 3.3: A block diagram of the trigger unit underlining important modules.

shows some possible modules for the trigger unit. Due to the modularity of the entire system, new trigger modules can be added or removed from the unit.

- **Activity recognition:** This module uses Google Play Services activity recognition API to estimate the possible activity of the device with an associated confidence. For the context of ITS, some interesting activities include: in a vehicle, on foot (walking/running), being still or device being tilted (possibly held in hand) etc. A sample case can be that the module generates a trip-start trigger when the device is most likely present inside a moving vehicle and a trip-stop trigger when the device is on the body of a walking person.

- **Bluetooth device:** This module produces triggers based on connection and disconnection with a known Bluetooth device. The module uses Bluetooth connection events to recognise the connection or disconnection with a saved Bluetooth device(s). A Bluetooth device may be associated with a particular vehicle. A sample case can be, the module generates a trip-start trigger when the user enters his/her vehicle and the device establishes a connection with the Bluetooth hands-free of the vehicle. When the device disconnects from the Bluetooth hands-free, a trip-stop trigger is generated.

- **Bluetooth beacon:** This module produces triggers based on reception of a Bluetooth Low Energy (BLE) message from a beacon. The module can use Google's Nearby Messages API to provide the desired functionality. A sample case can be that a Bluetooth beacon is associated with a vehicle and placed in it. When the device receives a message from this beacon, it identifies the beacon (and vehicle) and generates a trip-start trigger. A trip-stop trigger is generated when the device stops receiving messages from this beacon.

- **Wi-Fi device:** This module produces triggers based on connection and disconnection with a known Wi-Fi device. The module uses a similar mechanism as the Bluetooth device module. A sample case can be that when the device disconnects from a known Wi-Fi device (such as the home network) a trip-start trigger is generated. A trip-stop trigger is generated when the device connects to another known Wi-Fi device (such as the office network).

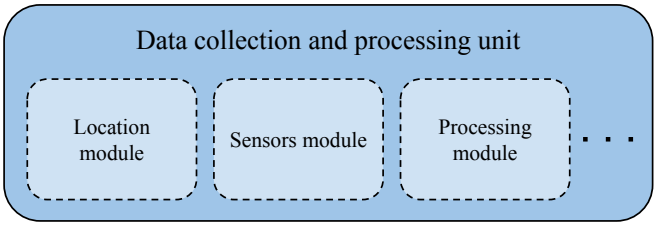


Figure 3.4: A block diagram of the data collection and processing unit highlighting important modules.

Data collection and processing unit

The data collection and processing unit serves as the main unit for acquisition and handling of all data. Figure 3.4 shows the most common modules for data collection and processing unit.

The *sensors module* is capable of using multiple hardware and software sensors available on the smartphone device. The platform supports three general classes of sensors:

- **Motion sensors:** These sensors measure acceleration and rotational forces along three axes. This class includes accelerometers, gravity sensors, gyroscopes, linear acceleration and rotational vector sensors.
- **Environmental sensors:** These sensors measure various environmental parameters, such as ambient air temperature, pressure, illumination, and humidity. This category includes barometers, photometers, and thermometers.
- **Position sensors:** These sensors measure the physical position and orientation of a device within a frame of reference. This class includes orientation sensors and magnetometers.

The *processing module* is where all the management and organisation of the collected data is performed. It offers multiple common and specialised algorithms and analysis techniques depending on the requirements of the application.

The *location module* is another vital component for ITS applications. One of the unique features of mobile applications is location awareness. The platform uses Google Play services location APIs which are preferred over the Android framework location APIs. These APIs use fused location provider to estimate the best possible location as per the configuration or application requirements. The fused location provider fuses location from multiple providers including GPS, mobile network location, Wi-Fi location and BLE location. As well as the geographical location (latitude and longitude), the API also provides further information such as the bearing (horizontal direction of travel), altitude, and velocity of the device.

3.1.2 Server platform

The server part of the platform offers back-end connectivity, data storage, and processing capabilities. The server platform is used to provide mandatory back-end services to ITS applications. The server offers the following basic services:

- connectivity
- data storage and processing
- client services

Overview

The server platform consists of 3 basic modular units.

1. Messaging unit
2. Database unit
3. Post processors unit

Figure 3.5 shows the generic relationship between the basic building blocks of the server platform. The messaging unit acts as the end-point for all communication from smartphone applications. The database stores data for analysis and model building. The post processors extract important information from collected data from multiple sources. In the following sections, individual units of the server platform are described.

Messaging unit

The messaging unit acts as a two-way communication end-point for one or multiple messaging techniques. Depending on the application scenario, different messaging techniques may be deployed to fulfil the requirements. It offers communication using the following techniques:

- REpresentational State Transfer (REST) based web services
- Simple Object Access Protocol (SOAP) based web services
- Firebase Cloud Messaging (FCM)

FCM (previously called Google Cloud Messaging (GCM)) is a service provided by Google which is a cross-platform messaging solution that offers reliable delivery of messages without any cost. As per server configuration, this unit may also provide a push or pull based services to third-party client(s).

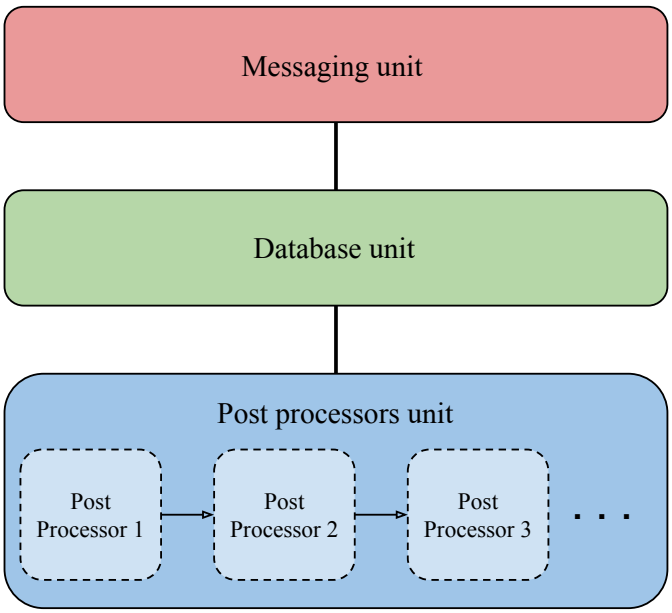


Figure 3.5: A high-level block diagram illustrating the overview of the server platform.

Database

The database unit offers services in order to store and organise data collected from applications. This unit may use any popular DataBase Management System (DBMS) solution such as MySQL or Microsoft SQL. Database unit works closely with the post processors in order to enhance available data. It enables the possibility to use data mining techniques to generate new information which was otherwise not obvious.

Post processors

The post processors unit provides a platform for multiple modules for extracting extra information from the collected data. The post-processor is a multi-threaded pipeline based application where each step of the pipeline performs a specific operation. The unit can be configured in order to add/remove post-processing stages or modules depending on the application requirements. Individual threads may work periodically or upon triggers (such as newly available data). Some examples of common ITS post-processing operations include:

- **Snap to roads:** Snap to roads relates raw GPS trails to physical roads on the map. Google's Snap to Roads web service API is used for this purpose. The API takes a maximum of 100 location points and may return 100 or fewer points. It relates latitude and longitude information into snapped latitude,

snapped longitude and place ID. Place IDs uniquely identify a place in the Google Places database and on Google Maps.

- **Reverse geocoding:** Reverse geocoding converts location data into a human-readable format. Google Place Details web service API is used to perform reverse geocoding. After trips are snapped to roads, all snapped points are assigned a place ID. These place IDs are reverse geocoded into human readable information using the API. The said IDs can be translated into information such as the street address, name of business, road, neighbourhood, locality, etc.

- **Trip segmentation:** A reverse geocoded trip can be segmented into different sections based on road characteristics. The segmentation technique is based on the reverse geocoded data. Trip segmentation helps in analysis and comparison with other trip data.

3.2 Applications

The platform can be used to implement a variety of ITS related applications. Some examples of such application are Driving Style Analysis (DSA), Virtual Induction Loop (VIL), Public Transportation System Advanced Vehicle Management system and so on. Using VDAP, some applications including DSA (refer to Chapter 4) and VIL (refer to Chapter 5) were designed, implemented and tested.

3.2.1 DSA

DSA is a system that is capable of collecting information about the driving behaviour of a user and providing some feedback about it. The system consists of an Android application, a back-end cloud server, and a web interface. The application works in the background to automatically collect information from multiple sensors of the smartphone when the user is on the move. The collected data is processed in real-time and processed information is sent to the server. The server performs some post-processing operations on the collected data and makes it available to the web interface. Users can access the web interface to analyse their vehicle trips and receive valuable feedback.

Figure 3.6 shows a detailed block diagram of the DSA Android application implemented using the Vehicle Data Acquisition Platform. In the application, the *control unit* uses the database, data storage, and communication modules. The database module provides access to a trip list table which stores information and status of all recorded trips. The data storage module handles all collected raw and processed data from sensors using multiple Comma-Separated Values (CSV) files. The communication module communicates with the back-end cloud server

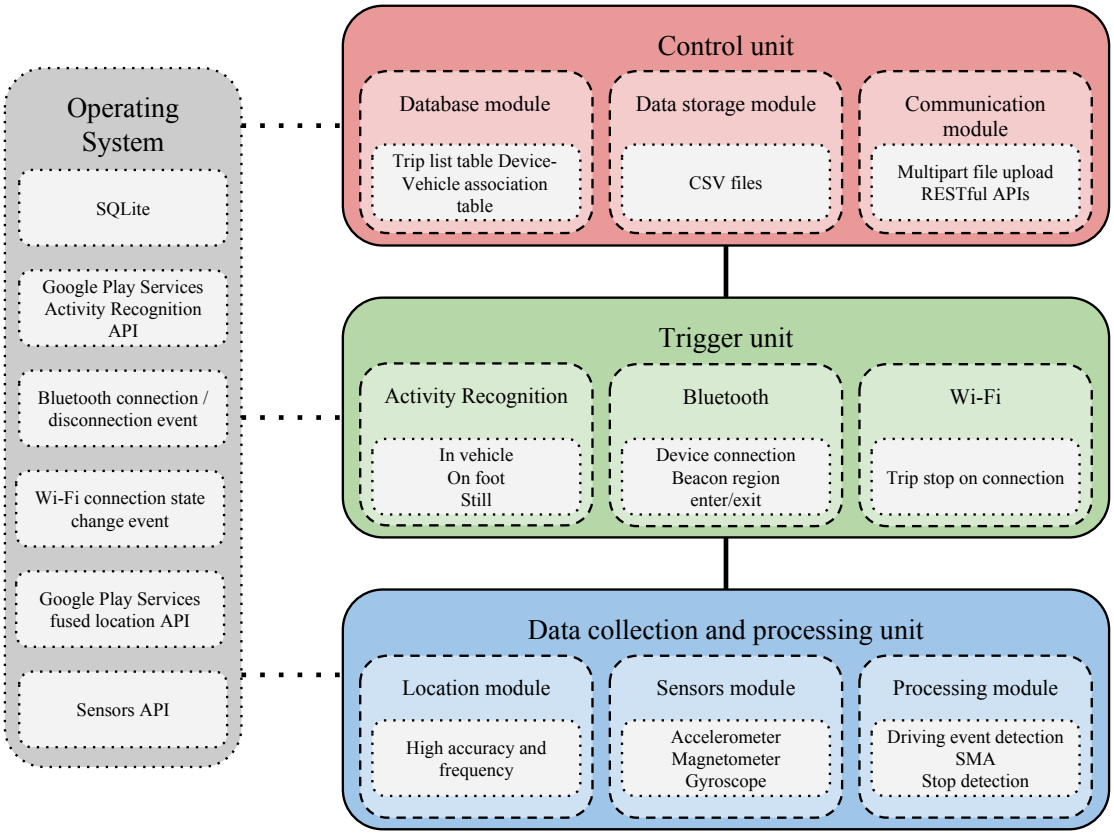


Figure 3.6: A block diagram showing the design of DSA application implemented using the Vehicle Data Acquisition Platform.

using RESTful APIs and HyperText Transfer Protocol (HTTP)'s multipart file upload. The *trigger unit* uses activity recognition, Bluetooth and Wi-Fi modules to detect if the user is on the go or not. The triggers provided by the modules are essential for the application to function autonomously. The *data collection and processing unit* is responsible for collecting and processing data in real-time using location, sensors, and processing modules. The location module provides highly accurate location very frequently. The sensor module subscribes to events of the accelerometer, magnetometer, gyroscope and linear acceleration sensors. The processing module detects the event in real-time using an algorithm composed of Simple Moving Average (SMA) and multiple thresholds. It also detects physical stops of the vehicle using data from location module.

3.2.2 VIL

VIL is a system that has the potential of complementing/replacing physical induction loops which are physically installed in road asphalt to monitor traffic

intensities. The system consists of an Android application and a back-end cloud server. The application works in the background and is fully autonomous. Virtual loops are defined in the server at intersections of interest and are synchronised with the application. The application automatically detects when the user is on the move and intelligently samples location to identify transits over virtual loops and sends relevant information to the server. The server, then, aggregates and processes data from multiple users and forwards information such as vehicle count, passage time and passage speed to an Urban Traffic Control (UTC) such as Urban Traffic OPtimisation by Integrated Automation (UTOPIA).

Figure 3.7 shows a detailed block diagram of the VIL Android application implemented using the Vehicle Data Acquisition Platform. In the application, the *control unit* uses database and communication modules. The database module provides access to a trip-list table and a cloud-synced table with information about nearby virtual loops. The communication module communicates with the back-end cloud server using RESTful APIs and Firebase Realtime Database to keep the database table(s) in sync. The *trigger unit* uses activity recognition, Bluetooth and Wi-Fi modules to detect if the user is on the go or not. The triggers provided by the modules are essential for the application to function autonomously. The *data collection and processing unit* is responsible for collecting and processing data in real-time using location and processing modules. The location module intelligently switches between low accuracy, less frequent and low energy location updates and high accuracy, more frequent and high energy location updates based on proximity to a virtual loop. Then the processing module estimates proximity to a nearby virtual loop and the time of transit over a virtual loop. Lastly, the transit information is forwarded to the server in real-time where it is processed and sent to a client.

3.3 Conclusions

This Chapter describes in detail the idea and the proposed structure of a modular platform for the collection and processing of data for ITS related applications. The platform called VDAP consists of a smartphone and a back-end server section. VDAP is highly customisable and can be adapted to provide a number of ITS related services. More details about the design and implementation of two applications using VDAP are available in Chapter 4 and Chapter 5.

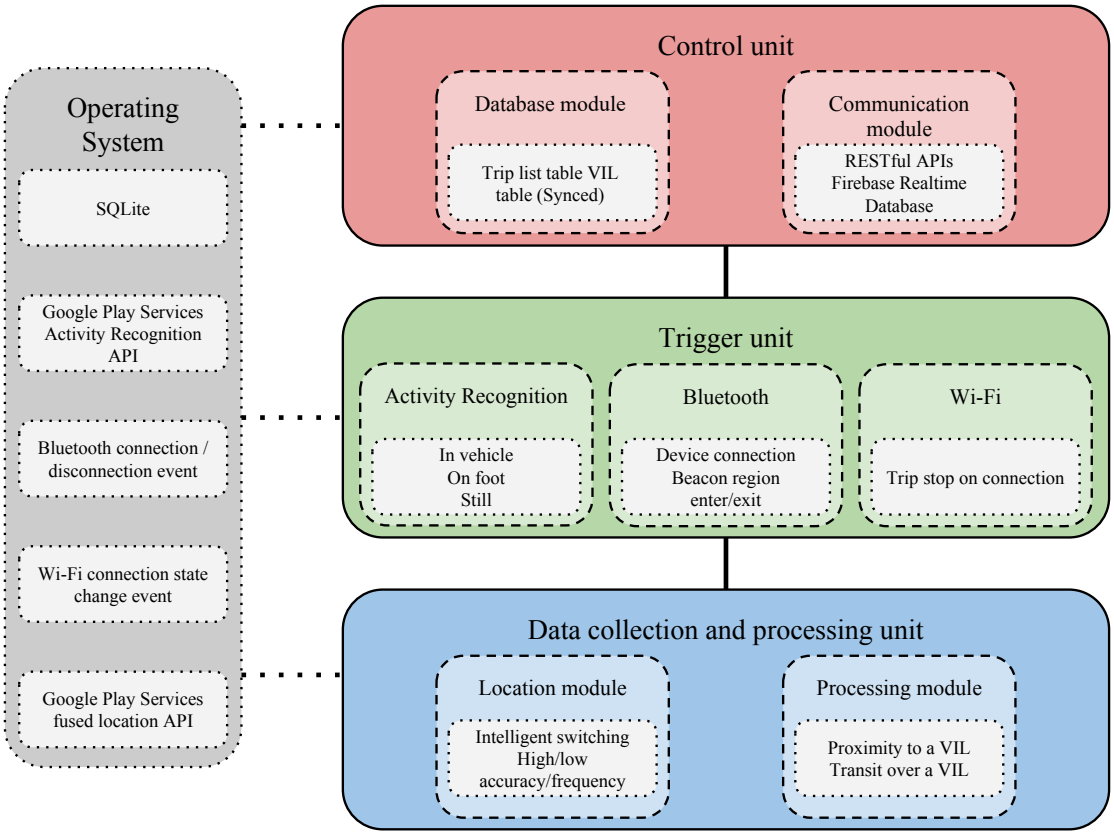


Figure 3.7: A block diagram detailing the design of VIL application implemented using the Vehicle Data Acquisition Platform.

Chapter 4

Driving Style Analysis (DSA)

In this Chapter, I describe the research activities related to the automatic recognition of the driving style of drivers. In every sector of life, be it personal, business or leisure, commutation is vital for us. This *need to commute* makes travel a necessity rather than a luxury. People strive for shorter travel times, fuel-efficient journeys while being safe from any hazards. Safety is an indispensable concern for both drivers as well as passengers. Driving style can characteristically be divided into two categories: typical (non-aggressive) and atypical (aggressive). In order to promote driver safety, studies have found that a driver's behaviour is relatively safer when being monitored, when feedback of specific driving events is provided, and when reports of potentially aggressive events are recorded [51].

On an industrial scale, big companies use a large number of vehicles, usually called a fleet, to transport its technicians and personnel on sites where they repair and set-up elements of the infrastructure. The management of this fleet is crucial for a company in terms of workforce productivity, cost control and safety. Apart from these technical fleets, some companies also own or maintain pick and drop service for employees or carpooling arrangements. Usually, *How's my driving?* bumper stickers are utilised to not only gather feedback for their drivers but also to give them a sense of being monitored. This approach enables the driver to be self-conscious and drive more responsibly [71]. If this manual approach of monitoring can be replaced with an automatic and unbiassed system, it would not only improve safety standards but also reduce human error in monitoring and may provide automatic assisted safety mechanisms.

Use of smartphone

The idea is to use a freely placed smartphone to recognise the driving style and behaviour of a driver and use this data to assist the driver not only to improve

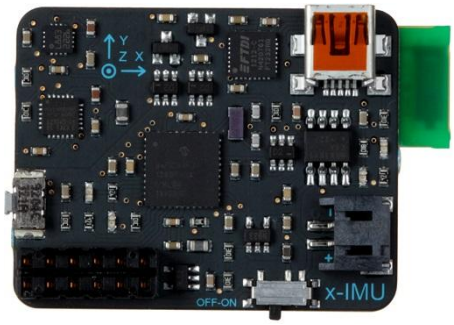


Figure 4.1: x-IMU from x-IO; an IMU with Bluetooth interface.

his/her driving style, but also to provide assisted-safety mechanisms in case of a predicted incident. A 3-axis accelerometer, 3-axis gyroscope, and 3-axis magnetometer can be used to recognise normal turns, normal accelerations and decelerations, abrupt or aggressive turns, aggressive accelerations and decelerations, and others. By monitoring data from sensors of a smartphone freely placed in a vehicle, it may be possible to understand the behaviour of the driver and the (so-called) *driving events*. This information can also be used to not only alarm, notify or penalise drivers but also to help them improve their driving skills. Using new generation networking capabilities, such information can be stored and further analysed at a remote server for research purposes. Also, an analysis of a driver's driving style history can be used to recognise his/her mood or physical state (tired, sleepy etc.).

Use of dedicated hardware

Another source of motion data may be a dedicated hardware device present in the vehicle. Such a device can be a simple data logger with a 9-axis IMU. This device may have capabilities to store the inertial and rotational data locally and, at the end of a trip, upload the information to a server using a built-in mobile data connection. Such a device may also function alongside a smartphone device. The IMU can sample and record data locally, later on, it can send the data via Bluetooth interface to the smartphone where it is uploaded to a server. At the server, analysis can be performed to identify driving events and trips. An example of a simple IMU device is shown in Figure 4.1. The x-IMU from x-IO hosts onboard sensors, algorithms, configurable auxiliary port and real-time communication via USB, Bluetooth or Universal Asynchronous Receiver-Transmitter (UART).

4.1 DSA application

DSA is an Android application designed to collect and process data from motion and position sensors of an Android smartphone device. Motion sensors are useful for

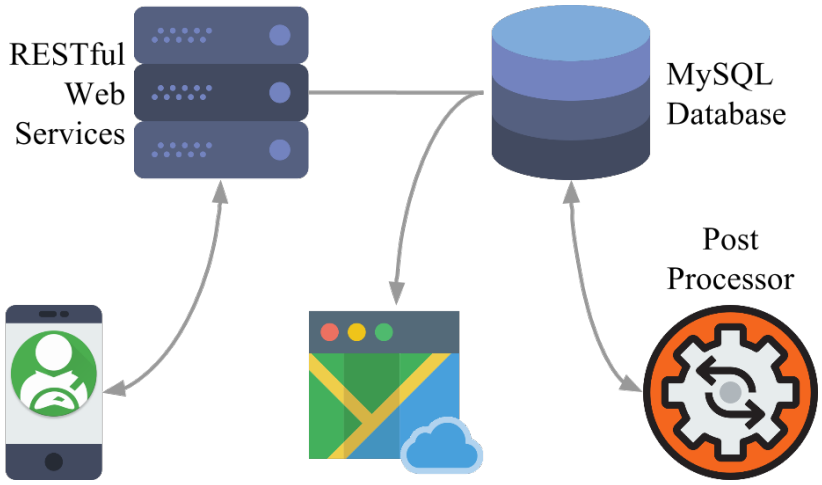


Figure 4.2: DSA system high-level block diagram.

monitoring device movement such as tilt, shake, rotation, or swing. The movement is usually a reflection of direct user input (for example, a user steering a car in a game or a user controlling a ball in a game), but it can also be a reflection of the physical environment in which the device is sitting (for example, moving with you while you drive your car) [25]. Position sensors are useful for determining a device's physical position in the world's frame of reference. For example, you can use the geomagnetic field sensor in combination with the accelerometer to determine a device's position relative to the magnetic North Pole [28]. Analysis of driving style can be broken up into recognising *driving events* or *manoeuvres* and creating a *driving profile* of each individual driver. The current driving style of a driver can be then compared with their known driving profile (or also possibly with other users' driving profile).

To support the DSA application, some other entities are implemented including several RESTful web services, MySQL based database, a post processor, and a web interface for analysis and visualisation. Figure 4.2 shows a high-level block diagram of the entire system.

The DSA application uses a number of motion and position sensors to collect and process data. A number of APIs are used to achieve the functionality. Figure 4.3 shows a high-level block diagram of the application.

4.1.1 Android sensor API

All smartphones produced within the last decade have a multitude of motion, orientation, and various environmental sensors. These sensors are capable of providing raw data with high precision and accuracy and are useful if it is required to monitor three-dimensional device movement, positioning, or changes in the ambient

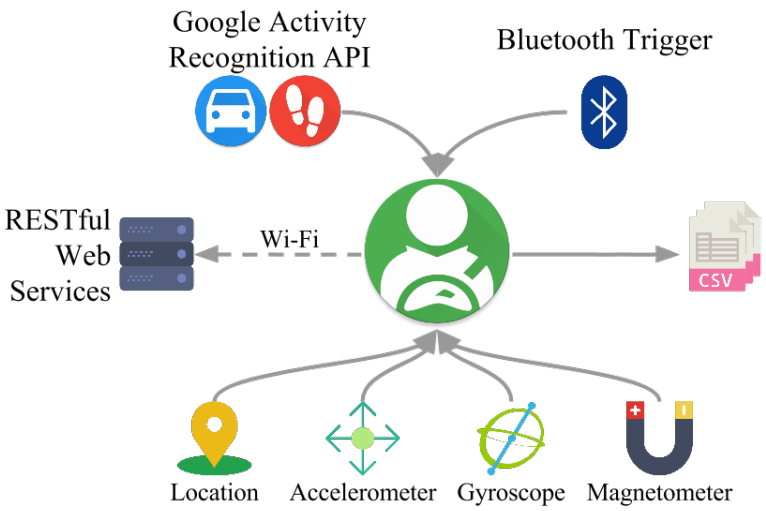


Figure 4.3: DSA application system block diagram.

environment near a device. Figure 2.1a shows the coordinate system that is used by all sensor APIs. Some of the relevant sensors provided by Android Sensor API are as follows:

Accelerometer

An acceleration sensor measures the acceleration applied to the device, including the force of gravity. All values are in SI units (m/s^2). The Android Sensor API for accelerometer (Sensor.TYPE_ACCELEROMETER [29]) provides the following data:

- Acceleration force along the x-axis (including gravity)
- Acceleration force along the y-axis (including gravity)
- Acceleration force along the z-axis (including gravity)

Linear acceleration sensor

The linear acceleration sensor provides a three-dimensional vector representing acceleration along each device axis, excluding gravity. All values are in SI units (m/s^2). The Android Sensor API for linear acceleration sensor (Sensor.TYPE-_LINEAR_ACCELERATION [29]) provides the following data:

- Acceleration force along the x-axis (excluding gravity)
- Acceleration force along the y-axis (excluding gravity)
- Acceleration force along the z-axis (excluding gravity)

Gyroscope

The gyroscope measures the rate of rotation around a device's x, y, and z-axis. Rotation is positive in the counterclockwise direction. All values are in SI units (rad/s). The Android Sensor API for gyroscope (Sensor.TYPE_GYROSCOPE [29]) provides the following data:

- Angular speed around the x-axis
- Angular speed around the y-axis
- Angular speed around the z-axis

Geomagnetic field sensor

The magnetometer estimates magnetic field at a given point on Earth, and in particular, computes the magnetic declination from true north. It provides a measure of the ambient magnetic field in the x, y and z-axis. All values are in SI units (μ T). The Android Sensor API for geomagnetic field sensor (Sensor.TYPE_MAGNETIC_FIELD [29]) provides the following data:

- Ambient magnetic field on the x-axis
- Ambient magnetic field on the y-axis
- Ambient magnetic field on the z-axis

4.1.2 Google play services location API

Google Play Services [26] provide highly optimised interface for receiving location updates. The Fused Location provider fuses location information from multiple sources (such as GPS, Wi-Fi or Network) to provide the best possible location information while balancing accuracy and power consumption. The API may provide the following information:

- **Latitude**: the latitude, in degrees.
- **Longitude**: the longitude, in degrees.
- **Speed**: the speed (if available), in metre/second over ground.
- **Altitude**: the altitude (if available), in metres above the World Geodetic System (WGS) 84 reference ellipsoid.
- **Bearing**: the bearing (if available), in degrees. The bearing is the horizontal direction of travel of the device and is not related to the device orientation. It is guaranteed to be in the range [0.0, 360.0].

- **Accuracy**: the estimated accuracy of this location, in metres. Accuracy is defined as the radius of 68% confidence.

4.1.3 Activity recognition

Activity Recognition API [21] from Google Play Services provides the ability to recognise a number of physical activities being performed by the user (or the physical state of the device). The API is able to identify the following activities:

- **In vehicle**: The device is in a vehicle, such as a car.
- **On bicycle**: The device is on a bicycle.
- **On foot**: The device is on a user who is walking or running.
- **Running**: The device is on a user who is running.
- **Still**: The device is still (not moving).
- **Tilting**: The device angle relative to gravity changed significantly.
- **Walking**: The device is on a user who is walking.

The API also provides a level of confidence for each activity in percentage. At the same time, multiple activities may have high confidence values.

4.1.4 Sensor data collection

The application uses a background service to collect and process data. The service has the ability to auto-start on start-up of the device. The service waits in idle mode until it is triggered by one of the available triggers. As of now, the following triggers are implemented in the app:

- Manual user trigger
  - Manually start and stop the trip using buttons in the app.
- Automatic activity recognition trigger
  - If the activity recognition API detects that the user is in a vehicle and the confidence level of this activity is beyond a certain threshold (in my experiments a value of 75% is reasonable) a detection session is initiated. The session may be terminated manually by the user or by the activity recognition API in case it is detected that user is on foot (implying that he has left the vehicle).
- Automatic Bluetooth device connection trigger

  - If the device connects/disconnects to/from a saved Bluetooth device, a trip is triggered. If there is no saved device, the application notifies the user in case a Bluetooth connection is established. ACTION_ACL-_CONNECTED [22] and ACTION_ACL_DISCONNECTED [22] broadcasts are used to start and stop a trip respectively.

During a session, raw data from multiple motion and position sensors are acquired and saved in CSV format on the internal memory of the device. In case the device has an external memory, the data is stored in external memory. The data files are available in the following location:

- "Internal storage\Download\SensorMonitorLog" if the device has no external memory card installed.
- "External storage\Download\SensorMonitorLog" if the device has an external memory card installed.

Data from following sensors are recorded by the service:

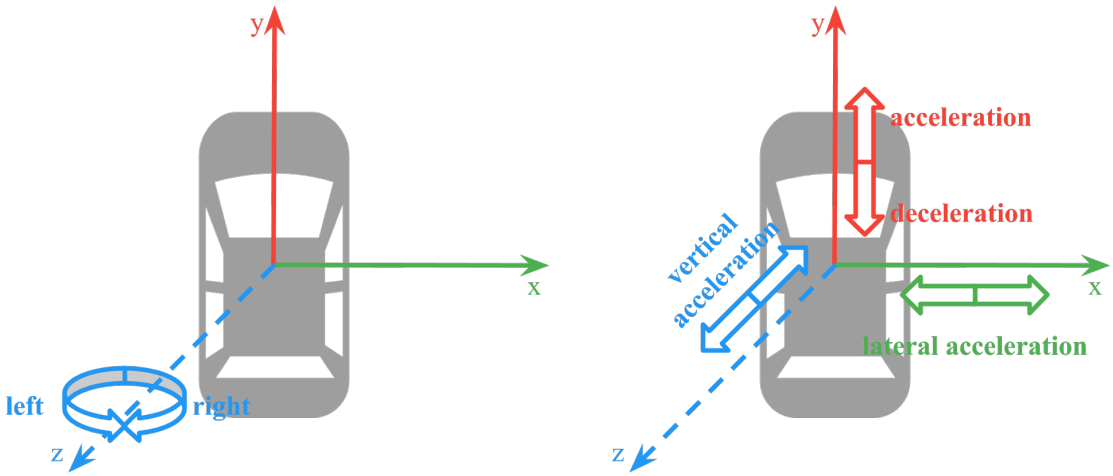
- **Accelerometer**: For calculating Rotation Matrix (refer to Equation 2.2g).
- **Linear Acceleration**: For identifying driving manoeuvres based on acceleration.
- **Gyroscope**: For identifying driving manoeuvres based on rotation.
- **Magnetic field**: For calculating Rotation Matrix (refer to Equation 2.2g).

Along with sensor data, location data is also recorded and stored as a CSV file. The sensor data are sampled at a very high sampling frequency in order to achieve a high level of definition of raw data. The sampling period is set to SENSOR-_DELAY_UI [31] which is defined as rate suitable for user interface and is roughly between 40 ms to 60 ms. For location updates, the inexact update interval is 2 sec while the exact maximum interval between location updates is 1 sec.

4.1.5 Definition of driving manoeuvres

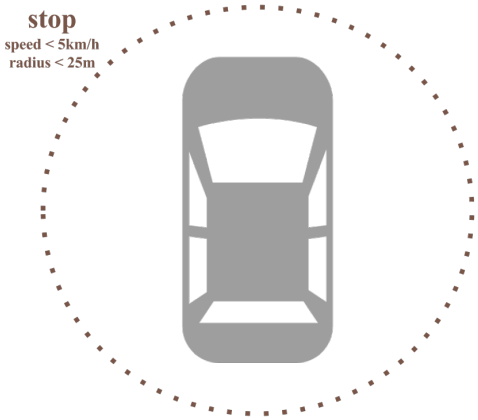
The driving manoeuvres can be classified as follows:

- Rotation based (refer to Figure 4.4a)
  - **left**: a counter-clockwise rotation along the z-axis of vehicle
  - **right**: a clockwise rotation along the z-axis of vehicle
- Acceleration based (refer to Figure 4.4b)



(a) rotation based

(b) acceleration based



(c) location based

Figure 4.4: Driving manoeuvres classification w.r.t. vehicle.

  - **accelerate**: a positive acceleration along y-axis of vehicle
  - **decelerate**: a negative acceleration along y-axis of vehicle
  - **lateral acceleration**: an acceleration along x-axis of vehicle
  - **vertical acceleration/bump**: an acceleration along z-axis of vehicle
- Location based (refer to Figure 4.4c)
  - **stop**: a halt in an area with low or zero speed

4.1.6 Data processing

In order to recognise driving manoeuvres, the raw data collected from sensors has to be related with the coordinate system of the vehicle. All raw data from sensors are originally referred to the coordinate system of the device. Since the device can be placed in any orientation or position in the vehicle, it is not possible to establish a fixed relationship between the reference system of device and vehicle. Furthermore, the device can be moved inside the vehicle during a driving session. Due to this, the relationship between the coordinate system of a device and a vehicle is established in a 2 step procedure.

Using the procedure explained in Chapter 2, raw data from the linear acceleration sensor and gyroscope are rotated from the device's coordinate system to the vehicle's coordinate system. After the data has been correctly rotated and is referenced to the vehicle's coordinate system, a combination of simple moving average and thresholds are used to identify driving manoeuvres. *A SMA is the unweighted mean of the previous n data.* SMA is calculated for the last n readings of sensor data values v_x. If those readings are represented as v_M, v_{M-1}, $\cdots$, $v_{M-(n-1)}$ then the formula for calculating SMA is mentioned in Equation 4.1. In my experiments, using 10 as the value of n is a decent choice.

$$SMA = \frac{v_M + v_{M-1} + \cdots + v_{M-(n-1)}}{n} \tag{4.1}$$

The timeline of an event has been illustrated in Figure 4.5. The Figure shows gyroscope data with reference to z-axis. The green region is an event recognised as right turn and the pink region as a left turn. TH_{upper} and TH_{lower} are defined as the upper and lower thresholds.

- **A**: At point A, $|SMA| < TH_{upper}$, so no event is identified.
- **B**: At point B, $|SMA| = TH_{upper}$, so the start point of turn event is identified.
- **C**: Between point B and D, $|SMA| > TH_{lower}$ and $SMA < 0$, so the event is recognised as a right turn event.
- **D**: At point D, $|SMA| = TH_{lower}$, so the end point of turn event is identified.
- **E**: Between point D and F, $|SMA| < TH_{upper}$, so no event is identified.
- **F**: At point F, $|SMA| = TH_{upper}$, so the start point of turn event is identified.
- **G**: Between point F and H, $|SMA| > TH_{lower}$ and $SMA > 0$, so the event is recognised as a left turn event.
- **H**: At point H, $|SMA| = TH_{lower}$, so the end point of turn event is identified.
- **I**: At point I, $|SMA| < TH_{upper}$, so no event is identified.

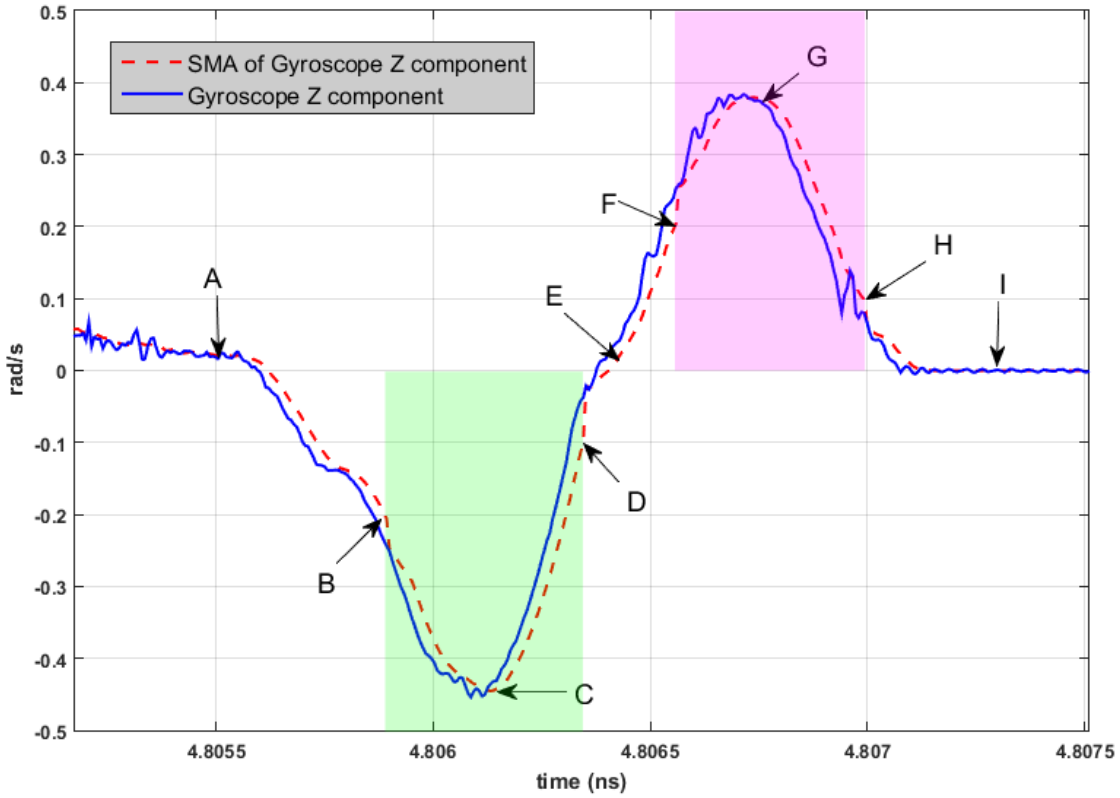


Figure 4.5: Example of gyroscope z-axis component and simple moving average.

According to my experiments, following values of threshold are suitable.

- SMA window size = 10 samples
- For left and right turns:
  - $\mathrm{TH}_{\mathrm{upper}} = 0.2$ rad/s
  - $\mathrm{TH}_{\mathrm{lower}} = 0.1$ rad/s
- For acceleration and deceleration:
  - $\mathrm{TH}_{\mathrm{upper}} = 0.8$ m/s^2
  - $\mathrm{TH}_{\mathrm{lower}} = 0.4$ m/s^2
- For lateral acceleration:
  - $\mathrm{TH}_{\mathrm{upper}} = 1.0$ m/s^2
  - $\mathrm{TH}_{\mathrm{lower}} = 0.5$ m/s^2
- For vertical acceleration:

  - $\text{TH}_{\text{upper}} = 0.7\ \text{m/s}^2$
  - $\text{TH}_{\text{lower}} = 0.35\ \text{m/s}^2$
- For stop:
  - Speed <= 5km/h
  - Radius <= 25m

4.1.7 Driving manoeuvres ranking

After the driving manoeuvres are recognised, they are ranked by a whole number. The rank of an event gives an idea about the *intensity* or *aggressiveness* of an event. A higher ranked event is similar to an impulsive event, such as an emergency brake. The event rank is calculated at the end of every event. Equation 4.2b is used to calculate the rank of an event. In the equation, $i = 0$ signifies the start of an event and $i = x$ signifies the end of the same event.

$$\text{event rank} = \left\lfloor \frac{\text{event SMA aggregate}}{\text{event time interval}} \right\rceil \tag{4.2a}$$

$$= \left\lfloor \frac{\sum_{i=0}^{x} |SMA_i|}{t_x - t_0} \right\rceil \tag{4.2b}$$

4.1.8 Automatic data upload

The data collected by the application needs to be uploaded to a server. In order to create a driver's profile based on his/her historic driving styles, the server needs data from that driver in a database. The background service collects a lot of data from multiple sources including accelerometer, rotated accelerometer, linear acceleration, rotated linear acceleration, gyroscope, rotated gyroscope, magnetometer, location, activity recognition, acceleration events, and rotation events. Due to the high sampling frequency of all sensor data, it would be traffic intensive to upload all data to a server on mobile data connection. For this reason, as soon as there is a new trip ready to be uploaded to the server, a lightweight background service is started to handle the upload.

The service checks if there is an active Wi-Fi connection available, if there is, the service uses standard HTTP POST's multipart form data uploading feature to upload big files to the RESTful web services. In case if a Wi-Fi connection is not available, the service waits in the background for a working Wi-Fi connection. Also, roughly every hour, the service activates and checks if there is a trip in the queue to be uploaded to the server and handles it accordingly.

4.2 Post processing

In order to process extra information from the collected data, a post-processing application is implemented in Java language. The post-processor is a multi-threaded pipeline based application where each step of the pipeline performs a specific operation. Every individual thread waits for a trip that is ready to be processed, periodically.

4.2.1 Driving events parsing

All driving events are identified by the DSA application in real-time. After the data is uploaded using DSA RESTful APIs, it needs to be parsed to compute certain parameters for the detected events. This significantly improved the performance of the JavaScript-based DSA web application. All types of events are parsed and inserted in tables. The information related to each event includes the ID of trip, the type of event, the rank of event (in case of STOP event is duration in seconds), the start timestamp of event, the stop timestamp of event, timestamp of the nearest (in time) location point, and timestamp in local timezone.

4.2.2 Reverse geocoding

Reverse geocoding converts location data into a human-readable format. Google Place Details web service API [24] is used to perform reverse geocoding. After trips are snapped to road, all snapped points are assigned a place ID. These place IDs are reverse geocoded into human readable information using the API. The said IDs are translated into the following information:

- Name of the road (route)
- Name of Country
- Names of administrative area level 1 to 5 (if available)

In case route name is not available, it is replaced with *NA*.

4.2.3 Segment statistics computation

After a trip has been segmented and all driving events are parsed, certain statistics related to each trip segment are computed and stored in a database table. The following statistics are computed in relation to each segment:

- Average speed
- Maximum speed

- Duration in seconds
- Snapped length in km
- The number of rotation, acceleration and stop events
- The average range of rotation, acceleration and stop events

The distance between two location points is calculated using Equation 4.3e.

$$\Delta_{lng} = (lng_2^\circ - lng_1^\circ)^c \tag{4.3a}$$
$$x_1 = sin(lat_1^c) \times sin(lat_2^c) \tag{4.3b}$$
$$x_2 = cos(lat_1^c) \times cos(lat_2^c) \times cos(\Delta_{lng}) \tag{4.3c}$$
$$x = x_1 + x_2 \tag{4.3d}$$
$$distance_{km} = \arccos(x)^\circ \times 60 \times 1.1515 \times 1.609344 \tag{4.3e}$$

4.2.4 Snap to roads

After a trip is successfully uploaded to the DSA server, it is snapped to roads. This relates raw GPS trails to physical roads on the map. Google's Snap to Roads web service API [33] is used for this purpose. The API takes a maximum of 100 location points and may return 100 or less points. It relates latitude and longitude information into snapped latitude, snapped longitude and place ID [27]. Place IDs uniquely identify a place in the Google Places database and on Google Maps [27]. This information is saved into the database.

4.2.5 Snapped bearing computation

After a trip has been snapped to roads, its new snapped bearing is calculated using snapped latitudes and longitudes. Bearing between two points is calculated using Equation 4.4d.

$$\Delta_{lng} = (lng_2^\circ - lng_1^\circ)^c \tag{4.4a}$$
$$a = \cos(lat_1^c) \times \sin(lat_2^c) - \sin(lat_1^c) \times \cos(lat_2^c) \times \cos(\Delta_{lng}) \tag{4.4b}$$
$$b = \sin(\Delta_{lng}) \times \cos(lat_2^c) \tag{4.4c}$$
$$\theta_s = ((\arctan_2(b, a))^\circ + 360^\circ) \mod 360^\circ \tag{4.4d}$$

4.2.6 Activity recognition confidence averages computation

The confidence averages for activity recognition data are computed using Equation 4.5a. All of the data is saved in the database. In the equation $i = 1$ signifies

the 2nd sample of data and $i = N$ signifies the last sample of data.

$$\text{average confidence activity}_x = \frac{\text{aggregate activity}_x}{t_{total}} \tag{4.5a}$$

$$\text{aggregate activity}_x = \sum_{i=1}^{N} \text{confidence activity}_x(i) \times (t(i) - t(i-1)) \tag{4.5b}$$

$$t_{total} = t(N-1) - t(0) \tag{4.5c}$$

4.2.7 Trip length computation

The distance between two location points is calculated using Equation 4.3e. The distance between all of the snapped location points are summed up to obtain the length of trip.

4.2.8 Trip segmentation

After the trip has been reverse geocoded, it is segmented into different sections based on road characteristics. The segmentation is based on the route name reverse geocoded data. After segmentation, information to identify a segment is saved in database, which includes trip ID, route name, country name, and names of administrative levels from 1 to 5.

4.3 Analysis interface

4.3.1 MatLab analysis scripts

In order to visually inspect the large amounts of data collected by the application, an interface was required to visualise the data. At first a couple of MatLab scripts were designed in order to parse the group of CSV files produced by the application. In order to visually inspect collected data, it was necessary to relate the detected events with location information on a map. Unfortunately due to the absence of a native mapping API for MatLab from major map vendors, the inspection was very limited in functionality and had a poor performance. Figure 4.6 shows a screenshot of the basic interface. The interface shows location points plotted on a static map image alongwith the speed of location points on a color scale and recognised events with a symbol (*i.e.* $<$ for left, $>$ for right , $\bigcirc$ for acceleration, $\triangle$ for acceleration (bearing), $\bigtriangledown$ for deceleration (bearing) and $\star$ for lateral acceleration).

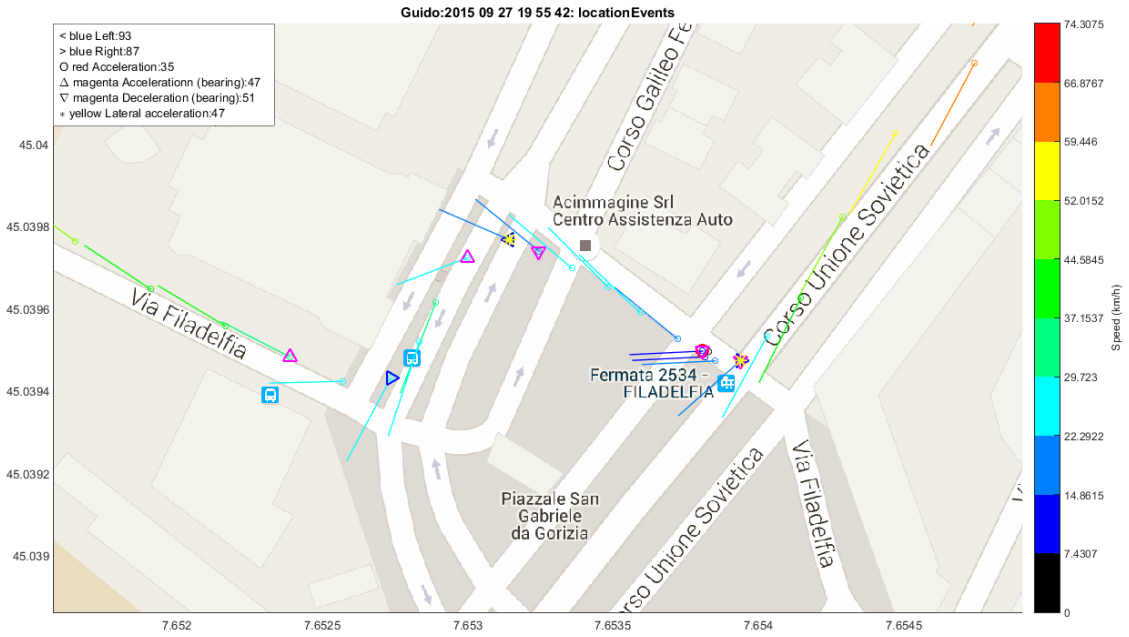


Figure 4.6: A sample screenshot of visual output from MatLab.

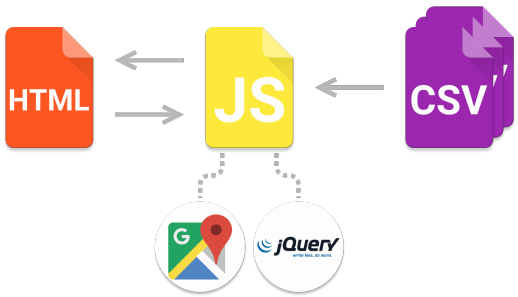


Figure 4.7: Block diagram of web interface for driving style analysis using CSV files as an input.

4.3.2 Driving style analysis web interface

Google Maps provides a fully functional high-performance API for JavaScript. To increase the functionality and improve the performance of MatLab analysis scripts a web interface was designed to use Google Maps JavaScript API. The development was done in two stages. In the first stage, the web application used JavaScript alongside Google Maps JavaScript API [23] and jQuery APIs [15] to parse the group of CSV files. At this stage, the CSV files were uploaded manually by the user to the web application. The JavaScript parsed all CSV files in order to provide an output for visual inspection of trips and driving events. Figure 4.7 shows a block diagram of the first stage for the driving style analysis web application.

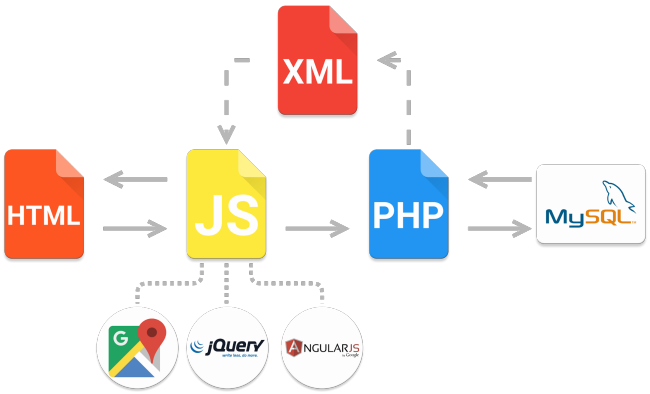


Figure 4.8: Block diagram of web interface for driving style analysis using MySQL database as data input.

(a) raw GPS mode (b) snapped to roads mode (c) segmented mode

Figure 4.9: Example of visualisation modes in DSA web interface.

The second stage of development benifited greatly from the automatic data upload feature which enabled the storage of all trip data from multiple users in a central MySQL based database. The web application was extended in order to allow users to visualise the trip data directly from the MySQL database server without the need to manually upload CSV files. Figure 4.8 shows a block diagram of the new web interface which uses the MySQL database as a data source. The DSA web interface provides 3 visualisation modes to analyse data shown in Figure 4.9 including raw GPS location mode, snapped to roads location mode and segmented view mode. The interface also provides various information and statistics such as:

- List of all trips by a user
- Confidence averages of activity recognition
- Location route with speed, bearing and accuracy in original mode

- Snapped location route with speed and snapped bearing in snapped mode
- Trip segments in segmented mode
- Driving events (refer to Section 4.1.5) with event rank and duration
- Trip and segment statistics
  - Average speed
  - Maximum speed
  - Length
  - Duration
  - Driving events' count and average ranks
- Observations in the segmented mode in comparison with comparison base related to
  - Average speed
  - Maximum speed
  - Driving events' average ranks
- Trip overall driving score based on scoring scheme (refer to Section 4.3.2)

Scoring scheme

In order to give a quantitative number to each recognised driving event and to estimate the entire trip's driving score quantitatively, a simple scoring scheme was created. This is a comparative scoring scheme which penalises higher than average values more than lower than average values. For each *comparison parameter* under observation, a *comparison base* is created. The comparison base can be created in one of two ways from all data available for the segment under consideration; 1) from all users, 2) only current user. In order words, the system can compare a users' data in a particular stretch of road to their own past behaviours or with the behaviours of all other users who have driven on the same stretch of road. After the choice of the comparison base, an average is calculated for each parameter under observation. Then the relative difference is computed between the comparison parameter and the average of the comparison base. Based on this difference, range and score are allocated to each segment of the trip. The complete scoring scheme is shown in Figure 4.10. For example, if a user's average speed in a segment is 35% higher than the average of the comparison base, a score of -3 will be given. On the other hand, if the average speed in a segment was 35% lower than the average of the comparison base, then a score of 0 will be given for this segment. After

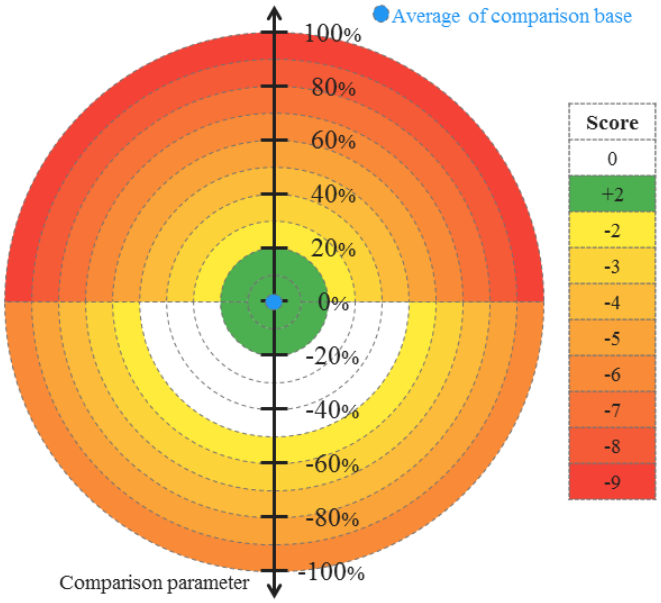


Figure 4.10: Scoring scheme for comparison of comparison parameters with comparison base.

scores are calculated for all segments of a trip, they are presented individually and as a cumulative sum of all scores. A positive overall score represents good driving behaviour, while a negative score implies bad driving behaviour. The scores are implemented as a lookup table in the database for easy configuration.

4.4 Conclusions

This Chapter demonstrates a sample use case of VDAP for the recognition and analysis of the driving style of a driver with a freely placed smartphone. The algorithm utilises VDDA for full-calibration and data rotation of IMU. The Chapter discusses in detail the APIs used from Android's Software Development Kit (SDK). DSA is essentially a two-part approach consisting of: 1) recognition of driving events or manoeuvres; 2) assignment of driving scores. DSA is able to recognise left/right turns, acceleration, deceleration, lateral acceleration and stop events with quite a high accuracy. The driving scores are constructed using the ranks (or intensities) of individual events and a comparison with other drivers' trips on that particular stretch of road. A scoring scheme is used to penalise higher or lower compared values (such as average speed, event ranks etc.).

Chapter 5

Virtual Induction Loop (VIL)

UTC system is an integral part of any smart city scenario. Efficient use of road resources and infrastructure became essential due to the ever-growing number of road vehicles. To achieve traffic flow control and coordination, UTC systems monitor, distribute and control the traffic flows [82]. Mainly, a UTC system consists of a model of the physical infrastructure, sensors, a single or multiple controllers, and actuators. The most common type of a road-traffic sensor is the Induction Loop (IL) detector which is usually buried under the road surface and identifies the passage of a vehicle through the changes in its inductance. The actuator (such as a traffic light) controls the flow of traffic according to the instructions of the controller. The controller constantly monitors and forecasts the traffic status and optimises the control strategy according to flow efficiency and/or environmental criteria. Adaptive Traffic Control System (ATCS) is a traffic light control programme which provides fully responsive traffic signal control based on real-time traffic conditions [68].

Urban intersections, being the hotspots of urban traffic networks, provide a very interesting case-study to demonstrate the potential of connected mobility. However, UTC systems heavily rely on data gathered from induction loops [3] to estimate the traffic situation and optimise the traffic flow and are not designed to include data from mobile devices. [53] mentions that the installation, maintenance, and operation of these infrastructure-based detectors lead to substantial costs for municipalities and road operators. Moreover, in the case of faulty or inoperative induction loops, the performance of the UTC system suffers significantly. In many cities, the percentage of induction loops that are out of service forces the operators to change the traffic signal control to fixed-time control, which leads to reduced performance.

In this Chapter, the aim is to supersede the induction loop detectors with VIL, a feasible and practical software-based solution which consists of an Android application and a central server in the cloud. While [34] discusses a VIL solution based on cooperative vehicular communication, it requires placement of Road Side

Units (RSUs) as well as vehicles with GPS that are capable of one-hop wireless communication. The goal of this Chapter is to study how a mobile application can be used as VIL and how such an application can be functionally integrated with a UTC system and particularly in UTOPIA described by [57].

5.1 System description

VIL is a system consisting of an Android application and a central server in the cloud. The application works in the background and is fully autonomous. Virtual loops are defined in the server at intersections of interest and are synchronised with the application. The application automatically detects when the user is on the move, intelligently samples location to identify transits over virtual loops and sends relevant information to the VIL server. The server aggregates and processes data from multiple users then forwards information such as vehicle count, passage time and passage speed to a UTC system.

5.1.1 Automatic trip start/stop detection

The Android application is able to detect that the user is on the move using one of the following techniques:

- Activity Recognition: using the smartphone's sensors to detect the possible activity being performed by the user; refer to [63]
- Bluetooth: based on connection/disconnection to/from a known Bluetooth device (such as the vehicle's hands-free);
- Bluetooth beacon: based on the presence of a Bluetooth beacon emitted by a known Bluetooth beacon associated with the vehicle.

The application automatically samples location with a variable sampling period (to save energy) based on information from the above-mentioned techniques. For example, the application samples location with high sampling period (around 10 to 20 seconds) to save energy. On the other hand, when the application detects the device is in the vicinity of a virtual loop, it samples location with a higher frequency (every 2 to 4 seconds) to increase accuracy.

5.1.2 Definition of a VIL

In order to detect the passage of a vehicle through a goal or a VIL in real-time, which may coincide with a real IL, goals need to be defined. A database of all goals is defined using the following parameters.

- Goal ID: a unique identifier of the goal
- Latitude: in signed decimal degrees
- Longitude: in signed decimal degrees
- Bearing: in degrees

5.1.3 Goal passage timestamp evaluation

To identify the passage of a vehicle over a goal, the distance and bearing difference between the vehicle and the goal were assessed. When a car is moving towards a goal, the distance between them will decrease. On the other hand, if the car is moving away, the distance will increase. At some point there will be two successive minimum distances, one is computed just before the goal and the other one right after it. However, detecting these minimum distances is not sufficient, because the distance does not take into account the direction.

Each measurement should have a speed value and a timestamp value; v_0, t_0 before the goal and v_1, t_1 after the goal. Assuming a constant linear acceleration, the equation holds $a = \frac{\Delta v}{\Delta t} = \frac{v_1 - v_0}{t_1 - t_0}$. The acceleration can be used to utilise kinematic equations, which can be used to find the time t (the time of passage over point G). Based on the position of the car relative to the goal, there may be two different application modes *i.e.* forward mode and backward mode. In the forward mode, the vehicle is approaching the goal and moving forward towards it. In the backward mode, the vehicle is moving away from the goal. By carefully choosing the signs of the parameters, all possible cases can be addressed to find the unknown time of passage t.

5.1.4 GPS accuracy problem

Naturally, the accuracy of the time of passage evaluation heavily depends on the accuracy of the location information from GPS. Since the GPS is not always an exact system, it may return biassed coordinates. A generic scenario of such a problem can be seen in Figure 5.1. A and B are two location points with different accuracies and an offset error, whereas point G is the VIL under consideration. Coordinates of point G are known and are snapped to the road so that there is no error in its coordinates. Due to the error in location A and B, the distance between the goal and the projected location of the vehicle on the stretch of road need to be computed. To solve and evaluate this problem, a number of different approaches are applied.

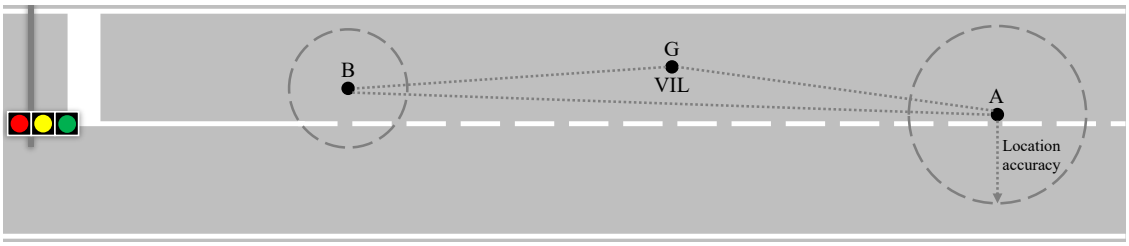


Figure 5.1: A generic scenario showing GPS accuracy problem with two location points A and B and a goal point G.

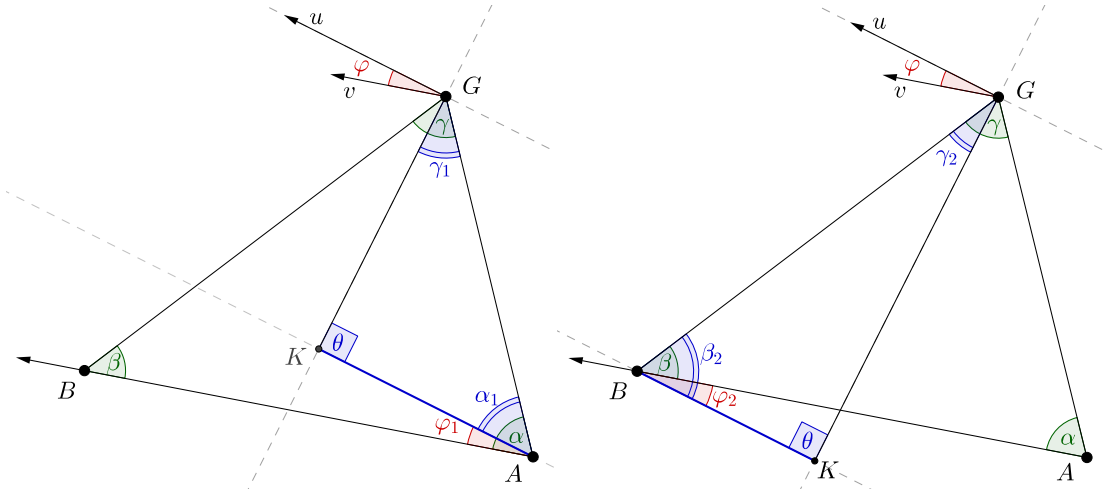


Figure 5.2: Trigonometry application, portion (a) before and (b) after the goal.

First approach: Trigonometry

The first approach exploits trigonometric calculations. Let us analyse the triangle in Figure 5.2a, the triangle formed by three points: the goal G, the last location before goal A and the first location after goal B. It is assumed that the bearing of the point A and B is the same. The distance needed to apply the forward mode is the result of a projection of the segment $\overline{AG}$ over the straight line passing through the bearing vector. In the same way, it is possible to compute the projection of the segment $\overline{GB}$ over the same straight line to apply the backward mode.

Using the area of $\triangle ABG$ and law of sines, the angles α, β and γ can be found. The angle φ is computed as the difference between the bearing of G and B. The angle α_1 is computed by subtracting φ from α. The angle γ_1 is computed by subtracting α_1 and 90° from 180°. In the triangle $\triangle AKG$, the adjusted distance, namely the value of the segment $\overline{AK}$, is found applying the proportion given by the law of sines. Similarly, in backward mode the adjusted distance $\overline{BK}$ shown

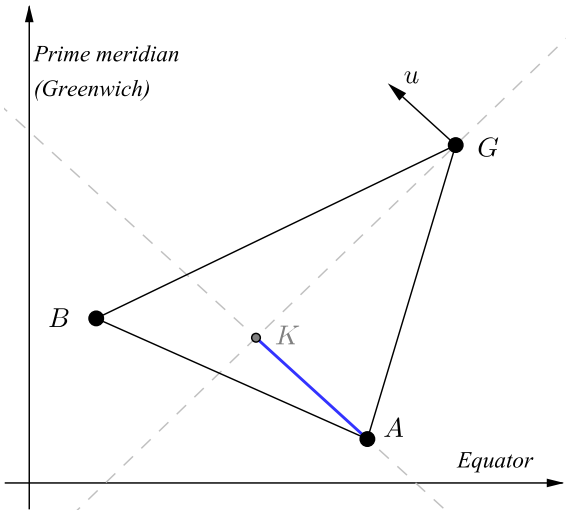


Figure 5.3: Plane geometry application.

in Figure 5.2b can be found.

Second approach: Plane geometry

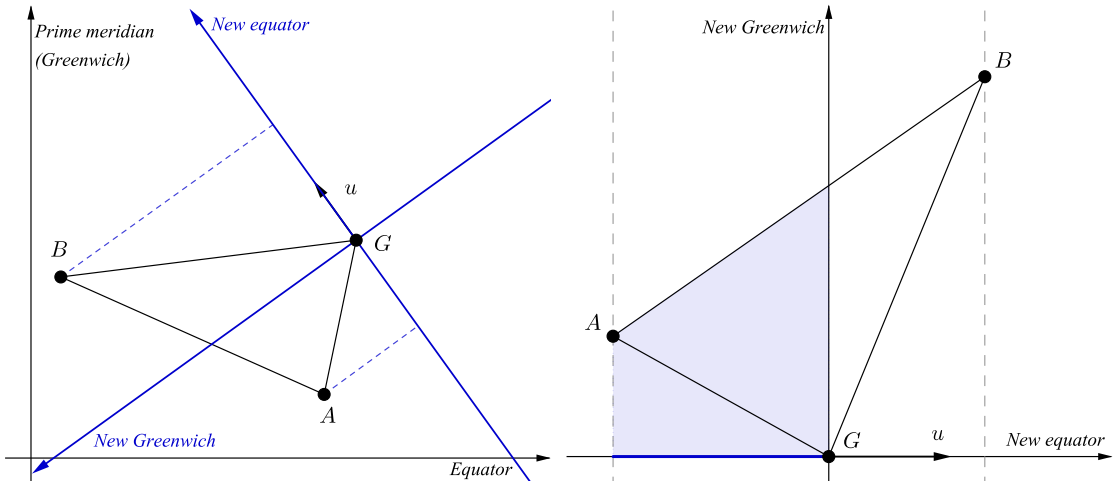
The second approach exploits plane geometry. The x-axis is the equator line, the y-axis is the prime meridian (Greenwich) and a point on Earth is determined by its latitude and longitude. Again, the scenario consists of three points forming a triangular shape: the goal G, the point before the goal A and the one after B as shown in Figure 5.3. The distance needed to apply the forward mode is the result of a projection of the segment $\overline{AG}$ over the straight line parallel to the bearing vector. The distance between K and A is the projection of the segment $\overline{AG}$.

First, the slope of straight lines parallel to bearing vector u, $m = \tan(\theta)$ is found, where θ is the angle measured anticlockwise from the x-axis. The coordinates of K *i.e.* (y_K, x_K) can be found using the slope-intercept form of a line parallel to bearing vector and passing through A *i.e.* $y - y_A = m(x - x_A)$, and a line perpendicular to bearing vector and passing through G *i.e.* $y - y_G = -\frac{1}{m}(x - x_G)$. Similarly, it is possible to compute the projection of the segment $\overline{GB}$ to apply the backward mode.

Third approach: Roto-translation

The third approach performs a rotation and translation centred around the goal as shown in Figure 5.4a, where the black axes are the old reference system and the blue axes are the new reference system. All geographical coordinates are converted to Cartesian coordinates expressed in metres for getting more precise results.

First, the origin of the reference system is shifted to the goal. The coordinates of



(a) (b)

Figure 5.4: (a) Pre and (b) Post roto-translation application.

the goal are subtracted from the coordinates (x, y) of A and B using the translation equations $x' = x - x_G$ and $y' = y - y_G$. In this way the goal is at the origin and other points are located according to the new reference. Secondly, the reference system is rotated around G (the new origin) such that the rotation angle is $\theta = 90° - \beta$. The new rotated coordinates (X, Y) of the points A and B are computed as $X = x' \cdot \cos\theta + y' \cdot \sin\theta$ and $Y = -x' \cdot \sin\theta + y' \cdot \cos\theta$.

After the roto-translation, the scenario appears like Figure 5.4b. In the roto-translated reference system, the modulus of the x-component of A and B represents the distance between the point and the goal. The sign of the x-component indicates whether the point is before (negative) or after (positive) the goal.

5.2 Results

In order to prove the accuracy of the different approaches, a number of tests were performed to access the error in passage timestamp, CPU time and success rate. Location data collected from multiple users using an Android application was used as input for the tests. For the sake of conformity, a filter was applied to the input data to remove some of the collected data points which do not satisfy the following criteria:

- Sample bearing must be within ±10° of goal bearing. This ensures that only location points on a straight stretch of road are used and turns are avoided (since the bearing experiences significant change in a turn).
- Sample accuracy must be less than a threshold. Analysis of the test-set

Method	Average (μ)	Min	Max	Standard deviation (σ)
No-adjustment	1.62e-2	-4.62e-4	4.99	0.4837
Trigonometry	1.63e-2	-2.42e-6	5.00	0.4990
Plane Geometry	5.22e-2	1.72e-4	5.03	0.4717
Roto-translation	3.38e-2	1.00e-3	4.99	0.4786

Table 5.1: Average, minimum, maximum and standard deviations of timestamp error in seconds for each method.

suggests a threshold of around 20-25 m.

- Samples must not be placed too far apart *i.e.* $d(A, G) > r_A + r_G$.
- Samples must not be placed too close to each other (such as nearly overlapped location points at a traffic signal).

Out of the test-set data, five random trips were selected for the tests. In every trip, after performing the above-mentioned filtering, tests were run. For every trip, every location sample is iterated over, considering three consecutive location samples at one time and taking the middle one as the goal. The computed timestamp is the average of the result from forward and backward mode, if both exist, otherwise the timestamp is equal to the only result successfully computed. Furthermore, a linear constant acceleration is assumed throughout the analysis.

5.2.1 Timestamp error

Since three consecutive location samples (triplets) are used at once for analysis, the difference between computed timestamp (timestamp of passage) and actual timestamp can be easily calculated. A comparison is presented between the different approaches.

Figure 5.5 shows the distributions of the timestamp error using the four different methods over a random sample of more than 700 triplets. The negative values in the distribution imply that the computed timestamp is smaller than the actual timestamp and vice versa for the positive ones. It is clear that the majority of the values are concentrated in the vicinity of zero. The values tend to form a bell-shaped curve which is symmetric around zero. Comparing the four distributions, there are no major differences, but only with plane geometry there is a small trend of returning timestamps greater than the actual timestamp. It is clear after this analysis that overall no adjustment is required for error correction. The method without any adjustment/error correction is quite accurate on its own.

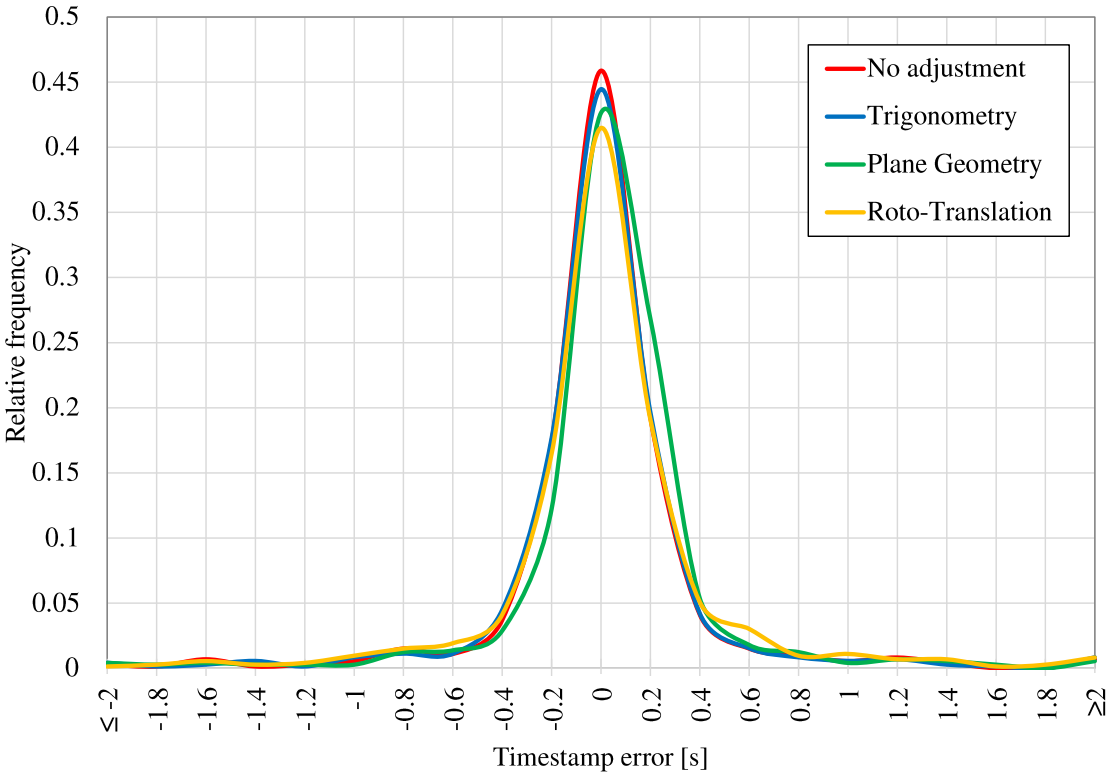
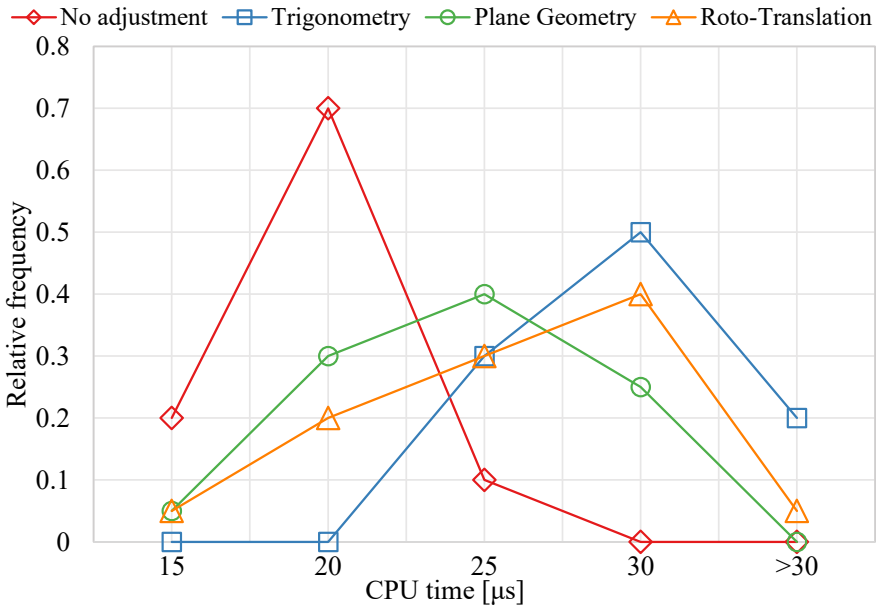


Figure 5.5: Error distribution in computed time of passage for different approaches.

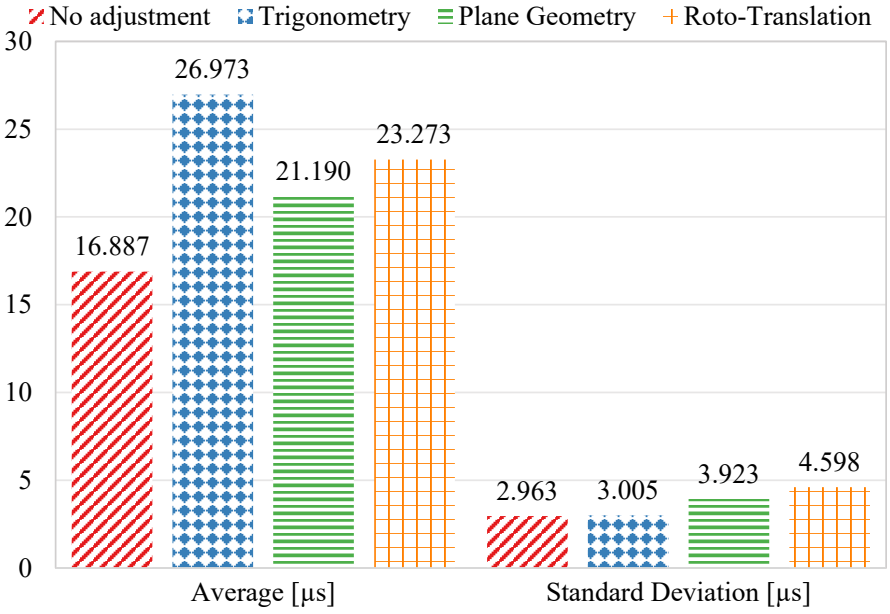
Table 5.1 lists the average (μ), minimum, maximum and standard deviation (σ) of timestamp errors achieved by all methods. The table shows that the average error is very small regardless of the method used, which is predictable since the distribution is symmetric around zero. The minimum error is quite low in each case, whereas apparently, the maximum error depends mainly on the nature of the trip since it is very similar for all methods. The standard deviations of the samples are not so high in all cases with respect to the average. This means that the values in all cases are quite close to the mean rather than spread out over a wide range.

5.2.2 Execution time

An important parameter to be analysed is the execution time for each method. The complexity of the algorithms influences the CPU execution time. To evaluate the execution time, five random trips were selected as before and the algorithm was executed on all triplets in the data sample. All four variants of the algorithm were run one by one. To measure the CPU execution time of each method a Java Profiler was used, profiling each method separately in order to avoid inaccuracies due to the scheduling.



(a) distribution comparison



(b) average (μ) and standard deviation (σ) comparison

Figure 5.6: CPU execution time comparison for all algorithms.

Figure 5.6a shows the distribution of the CPU execution time of the four methods. The chart shows that the values are well distributed with a sharp peak concentrated around 20 μs for the no-adjustment method. For other methods, the values

have relatively flatter peaks concentrated between 25 to 30 μs. Figure 5.6b shows a comparison bar chart of average and standard deviation of the CPU execution time for all four methods. The no-adjustment method is the fastest with the best average and low standard deviation due to the fact that it does not involve any extra adjustment calculations. Methods with adjustments do not show diversity in averages and standard deviations. However, the analysis for the other three methods is indecisive.

5.2.3 Success rate

The success rate is the fraction or percentage of success among a number of attempts. Due to the nature of mathematical calculations involved in the algorithms, it is possible that a method may result in a failure. As an example, when dealing with distances based on high accuracy latitude/longitudes, a fraction may have a numerator and/or denominator so small that it is approximated to a 0. This would result in a $\frac{0}{0}$, which is an undefined operation, returning NaN. Note that the returned timestamp is the average of the two timestamps from forward and backward modes if they both exist. Otherwise, it is equal to either of the valid timestamps. This makes success rate an important parameter to be analysed.

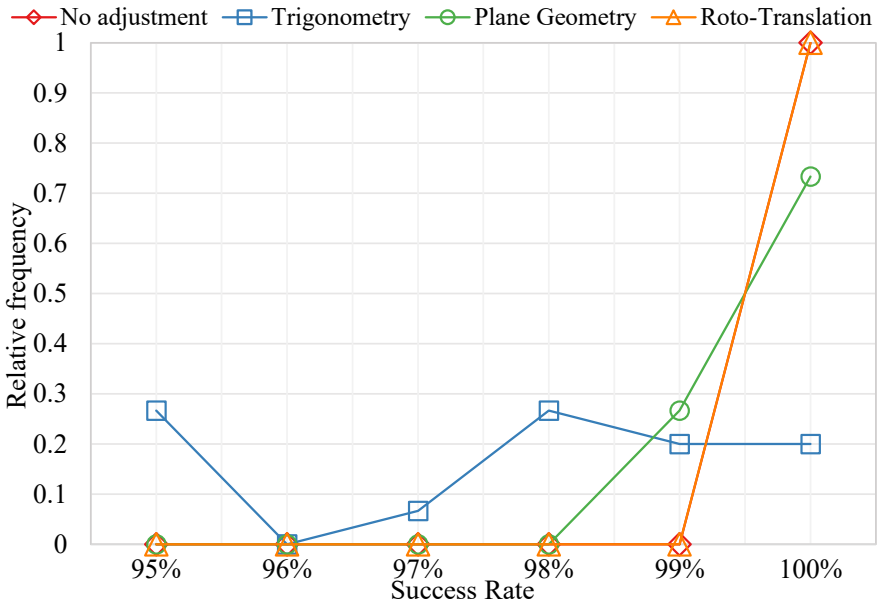
Figure 5.7a shows the distribution of the success rate for all methods. The no-adjustment and roto-translation methods exhibit similar behaviour with a concentration between 99% and 100%. Plane geometry method also shows concentrated distribution towards a high success rate, whereas the trigonometry method shows the worst performance due to the nature of calculations. Figure 5.7b shows a comparison bar chart of averages and standard deviations for all methods. The best success rate is achieved by the method without adjustment and roto-translation. Since no-adjustment and roto-translation methods show the same parameters, this implies that in this case, the few failures do not depend on the adjustment calculations, but rather on the kinematics mathematical model.

5.3 Integration

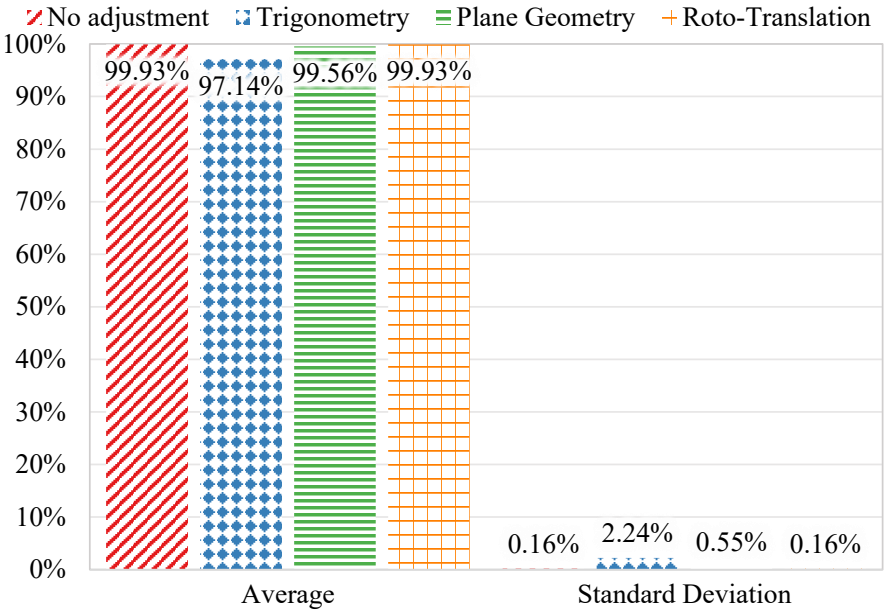
The information generated by the VIL system can be highly valuable to a UTC system. UTC systems consume data from conventional sources (such as IL detectors). In the following sections, a UTC system is introduced called UTOPIA and comment on the possible integration mechanism with UTOPIA.

5.3.1 UTOPIA/SPOT

UTOPIA is an adaptive UTC system that has been successfully implemented in many cities around the world and optimises traffic flow by taking into account



(a) distribution comparison



(b) average (μ) and standard deviation (σ) comparison

Figure 5.7: Success rate comparison for all algorithms.

both private traffic and public transport vehicles. The goal of the optimisation is to minimise the total travel time in the network. UTOPIA is a closed loop control system, where the control actions (*i.e.* signal timings) are decided based on

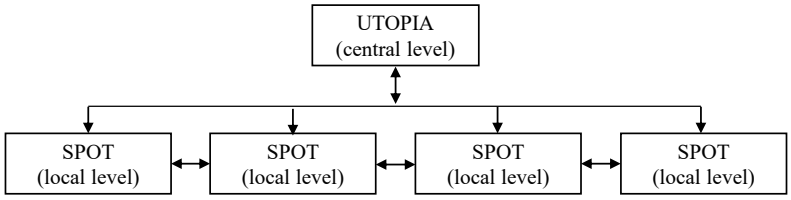


Figure 5.8: Hierarchical architecture of UTOPIA/SPOT.

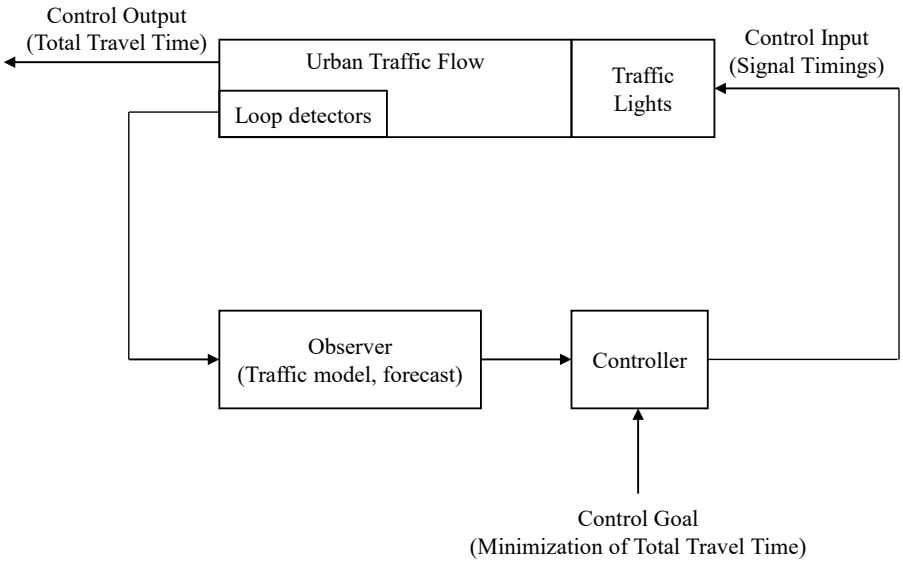


Figure 5.9: Control loop of UTOPIA/SPOT.

real-time traffic estimation. Induction loops are used to measure traffic flow and update the traffic estimation every 3 seconds, whereas traffic actuation takes place every second. To deal with the complexity of the signal optimisation problem in a traffic network consisting of numerous intersections, UTOPIA applies the closed loop approach in two levels (hierarchical system). The upper level (central level) monitors and controls the whole network consisting of areas with several intersections. The lower level (local level) controls single intersections while taking into account also adjacent intersections and the central level. The core intelligence of the system is located on the local level where the SPOT unit plays the role of the local observer and controller. Figure 5.8 shows the hierarchical approach of UTOPIA/SPOT and Figure 5.9 depicts the control loop of UTOPIA/SPOT based on [57] and [62].

5.3.2 Functional Integration of VIL

For the integration of VILs in the UTOPIA/SPOT system, I propose the communication of the VIL server with the local units (SPOT) and not with the central

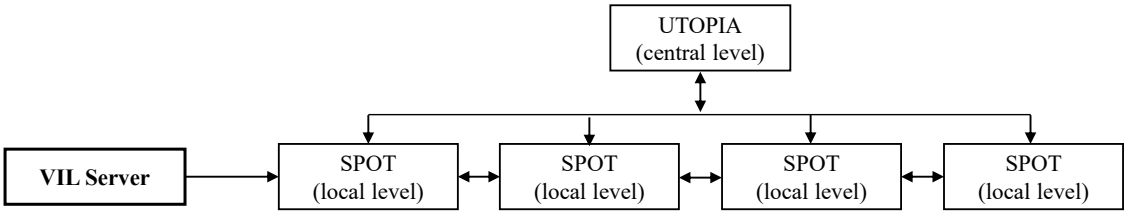


Figure 5.10: Integration of VIL server at the local level.

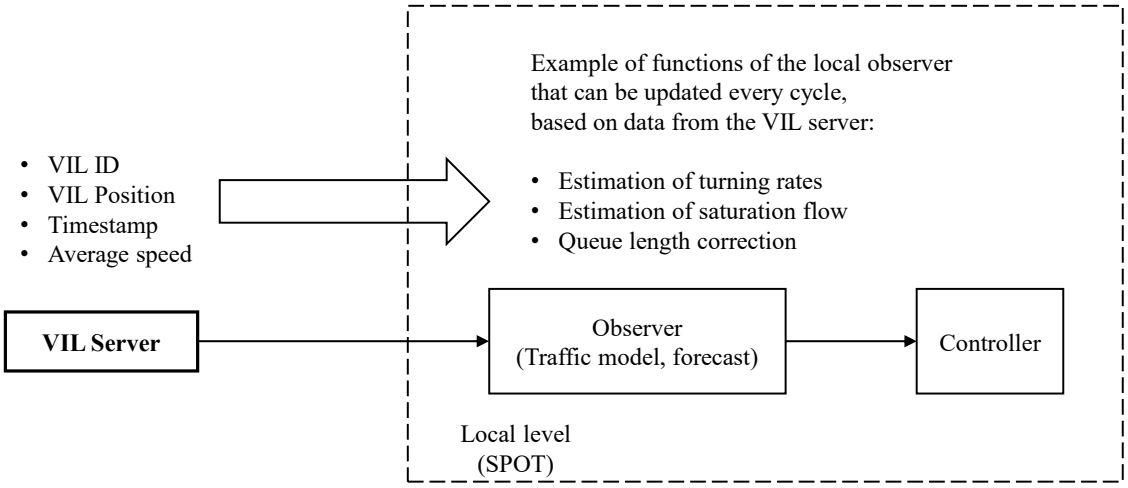


Figure 5.11: Integration details of VIL at the local observer.

level (UTOPIA) (refer to Figure 5.10 and Figure 5.11). The reason is that the local observer is already designed to update the estimation of certain crucial parameters every traffic signal cycle (typically around 70-90 seconds). If the VIL server contains information on an intersection that is proven to be more reliable than the information coming from stationary loop detectors, it can be allowed to update the estimated values for certain cycles. This information will then be communicated also to the central level and to the neighbouring intersections due to the inherent communication of the UTOPIA/SPOT system. The VIL should not be modelled as a typical IL in the SPOT unit, because of the completely different detection rate *i.e.* it cannot be expected that every passing vehicle is equipped with such an application. There are two possible approaches: Either the local observer has to be extended in order to *translate* the information from VIL into parameter estimation or the VIL server has to deliver the estimated parameter and its precision.

5.4 Conclusions

Intelligent traffic control systems are crucial for modern cities. The data source for these control systems has mainly been conventional induction loops which are

expensive to install and maintain. While [34] discusses the possibility of switching traditional loops with virtual induction loops, the suggested VILs are also another piece of hardware. I propose a completely software-based solution with already available and ubiquitous hardware, *i.e.* smartphones. The Chapter proves the feasibility of using smartphones to collect data and provide highly accurate information about a passage of a user over pre-defined goals. Simulation results based on real vehicular traces show negligible timestamp calculation errors with nearly 100% success rate. Moreover, due to the flexible and scalable nature of VIL, the goals can be defined in real-time to focus on areas of special interest. I also proposed a practical integration scheme to allow UTC systems, such as UTOPIA, to benefit from VIL.

Chapter 6

Traffic forecasting

Without traffic forecasting, UTC systems can only rely on the current situation of traffic, estimated by road sensors, which is not sufficient for planning and optimisation [81]. The aim of traffic flow prediction is to estimate the number of vehicles per unit time at a given location point or road segment [85]. It allows the implementation of several ITS solutions, such as ATCS, traffic management systems, advanced public transportation systems and logistics management. Forecasting is achieved using multi-sourced historical and real-time data processed by multiple types of forecasting and prediction models [83].

Machine learning is seeing more advancements and application than ever. Ever since the revival of Deep Learning [52], there has been more and more applications in self-driving cars, robotics, image/object recognition, voice recognition, healthcare, cancer diagnosis, earthquake prediction and weather forecasting. *Deep Learning is a type of machine learning which uses multiple-layer architectures to extract inherent features and structure in data from the lowest to highest level.* According to the Gartner hype cycle for emerging technologies of 2017, machine learning is at its peak of inflated expectations and is expected to reach a plateau within 2 to 5 years [77].

Available solutions

Traffic prediction approaches have been extensively researched in literature and generally can be grouped into three basic categories, *i.e.* *parametric*, *non-parametric* and *simulation-based* models. Parametric models include time-series models [19], Kalman filtering models [61], etc. Non-parametric models include Support Vector Regression (SVR) methods [8], Artificial Neural Network (ANN) [69], etc. Simulation-based approaches use traffic simulation tools to predict traffic flow [64].

Reviews of short-term traffic forecasting indicate that models are becoming more and more data-driven rather than analytical, due to an increase in computational

intelligence [73]. Researchers have been proposing traffic flow prediction models as early as in the seventies. For example, the Auto-Regressive Integrated Moving Average (ARIMA) model and its derivations have been used to predict short-term flows of traffic for over 3 decades [7]. Although, now the focus has moved from classical statistical models to neural-network based models [74]. Non-parametric models have also been extensively researched due to the non-linear nature of traffic flow. Apart from the three basic categories of prediction models, hybrid models, which combine one or more techniques have also been proposed [47]. A Stacked Auto-Encoder (SAE) based model was proposed for 15 to 60 min traffic flow prediction [83]. This model generates predictions with reasonable accuracy, but is computationally complex due to the need for a higher number of hidden layers and hidden units in the learner.

In summary, many traffic prediction models have been proposed in the literature to provide real-time traffic flow information in ITS, UTC and smart cities. Extensive comparison and review studies suggest that there is no technique that clearly outperforms other methods in all situations [45, 73, 6].

My approach

In this Chapter, I propose a short-term urban traffic forecasting solution applying supervised window-based regression analysis using Deep Learning. Experiments were performed with multiple configurations and prediction schemes to optimise the learning and prediction process. The resulting model is much simpler than other Deep Learning based traffic models proposed in the literature, yet it still outperforms them.

6.1 The Dataset

The dataset used to train the Artificial Intelligence (AI) machine is publicly available at the Uniform Resource Locator (URL) `http://opendata.5t.torino.it/get_fdt` in XML format [70]. It provides information about the number and average speed of the vehicles travelling in the urban area of Turin, detected by the means of ILs or aerial sensors. The dataset contains a single table with the following columns:

- **start_time**: Timestamp for the start of the period to which the data are referred
- **end_time**: Timestamp for the end of the period to which the data are referred
- **period**: Aggregation period [min]

- **lat**: Latitude in the WGS 84 system of the measuring station
- **lng**: Longitude in the WGS 84 system of the measuring station
- **Road_name**: Name of the road where the measuring station is located
- **Road_LCD**: Traffic Message Channel (TMC) code of the road on which the measuring station is present
- **lcd1**: Code of the initial node or TMC code of the initial location of the source arc
- **direction**: TMC direction (positive or negative)
- **accuracy**: Accuracy of the measurement [percentage]
- **offset**: Distance from the start of the arc along the direction of travel [m]
- **flow**: Vehicular flow [vehicle/hour]
- **speed**: Average speed [km/h]

The table contains roughly 130 data sources that are updated every 5 minutes. In the analysis that follows, the data sources that are inside the boundaries of Metropolitan City of Turin (some 126 observation points) were considered. This data is collected by UTOPIA project [57]. UTOPIA is a hierarchical decentralised traffic light control system that has been implemented and tested in a large area within the city of Turin.

6.1.1 Dataset analysis

Figure 6.1 shows the map of all data sources in the urban region of Turin. The data sources are well spread out over major city roads and intersections. I developed a simple data logging tool to fetch real-time traffic data from 5T and locally archive it in a database. The source code for this logger are publically available [40]. For the sake of this analysis and tests, data was collected from 01-Oct-2017 till 28-Feb-2018. During this period there were 103 weekdays, 20 Saturdays, 21 Sundays and 7 holidays.

The analysis of the dataset shows that the data can be categorised into four types of days [16]:

1. Weekdays (from Monday till Friday)
2. Saturdays
3. Sundays

Figure 6.1: Geographical location of all data sources (red dots) in the urban area of Turin.

4. Holidays (national and local holidays)

The average vehicular flow in the city changes according to the type of day, as seen in the Figure 6.2. As expected, weekdays show the highest flow while holidays experience the lowest flow. During weekdays, the two highest peaks are from 07:40 until 8:50 and from 17:10 until 18:35. These peaks are due to commuters travelling between home and workplace. Saturdays and Sundays experience much less traffic than weekdays and the highest peaks are between 12:05 and 12:50. The average speed in the city is shown in Figure 6.3. Saturdays, Sundays and holidays follow a very similar trend and the average stays at roughly 29 km/h. On weekdays, traffic slows down during peak hours (*i.e.* from 07:20 till 09:55 and from 16:55 till 19:25). This is due to the high vehicular flow during those periods, which generates congestion.

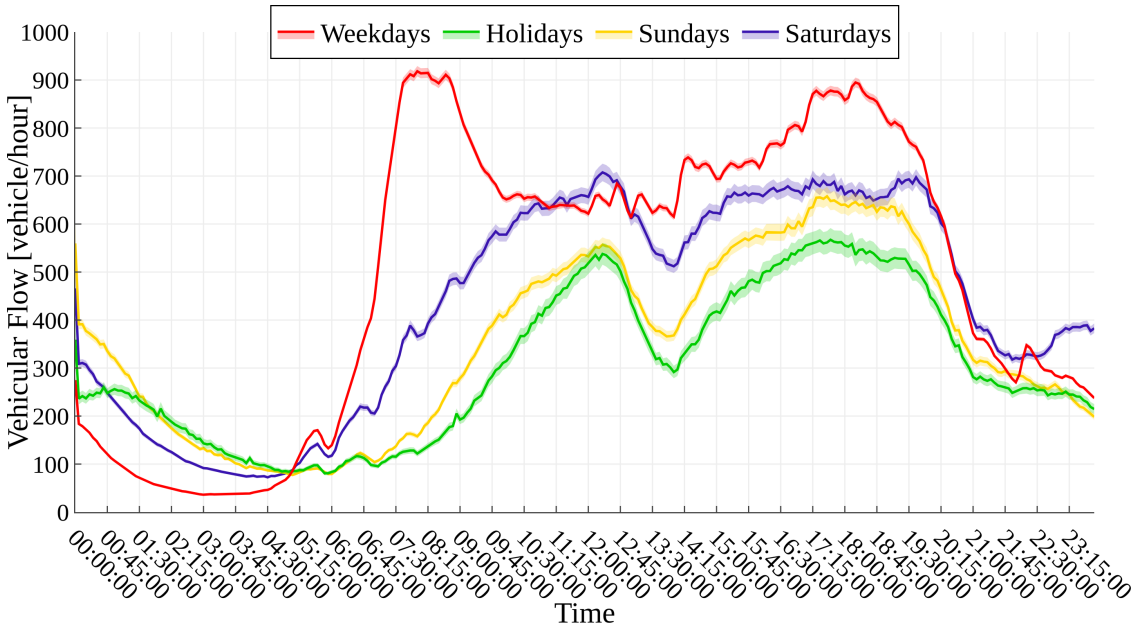


Figure 6.2: Average vehicular flow in the city of Turin categorised by type of day (bands represent 95% confidence interval).

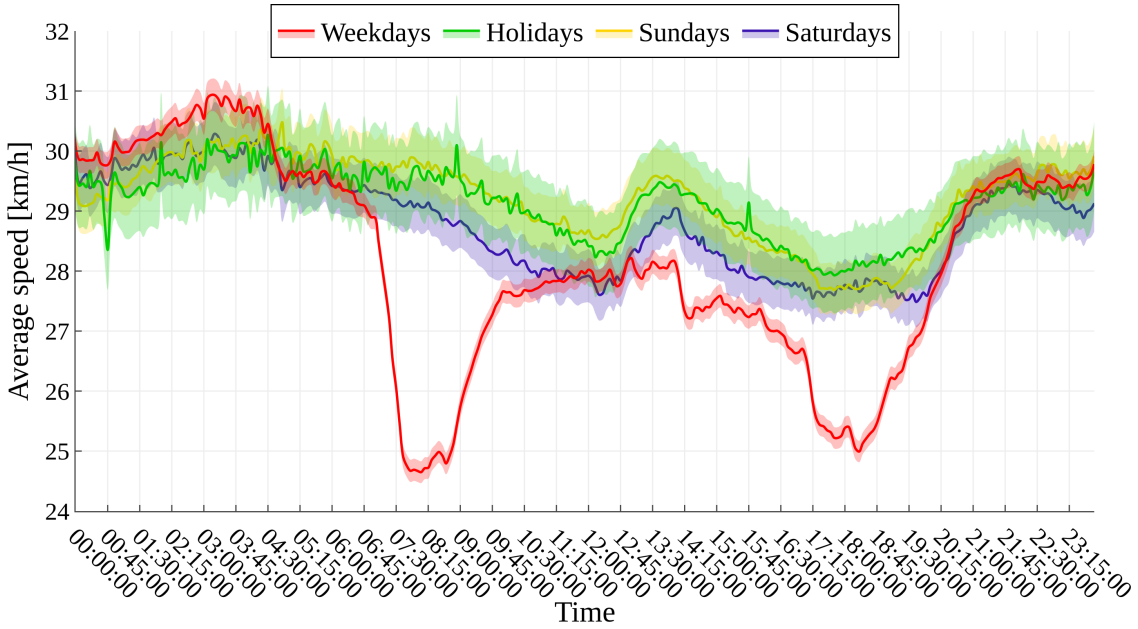


Figure 6.3: Average vehicular speed of vehicular flows in the city of Turin categorised by type of day (bands represent 95% confidence interval).

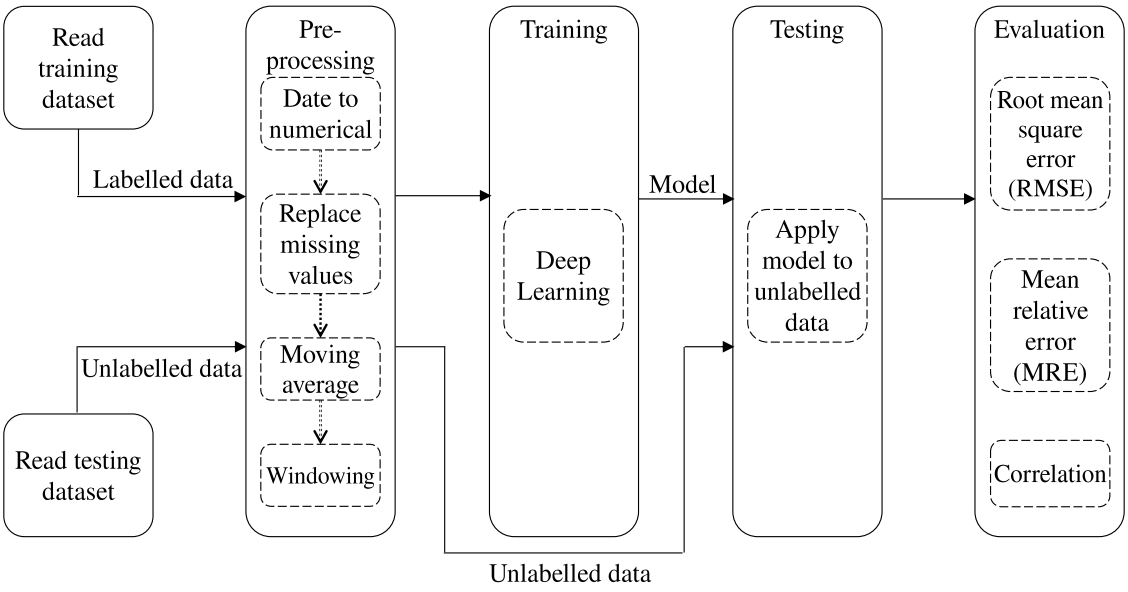


Figure 6.4: Prediction work-flow for supervised Deep Learning based regression for windowed time series forecasting.

6.2 Methodology

To predict the intensity of traffic, supervised window-based regression analysis with Deep Learning was used. *Regression analysis* is a set of statistical processes for estimating the relationships among variables. *Supervised learning* is the machine learning task of inferring a function from labelled training data. *Deep Learning* is based on a multi-layer feed-forward ANN that is trained with stochastic gradient descent using back-propagation. A *feed-forward Neural Network (NN)* is an ANN wherein connections between the units do not form a cycle.

The process of machine learning usually involves gathering data, preparing data, selecting a model, training the model, evaluating its performance, tuning model parameters, and finally generating predictions. Figure 6.4 shows prediction work-flow for forecasting this time series. Firstly, the labelled training dataset is read from the database. Then it passes through a few pre-processing steps depending on the type of test being performed. An explanation related to the different pre-processing steps can be found in Section 6.2.1. The labelled training dataset is passed to the Deep Learning machine which creates a model of the data. The unlabelled testing dataset is read and passed through the same pre-processing steps to evaluate the performance of the generated model. Once the testing dataset is labelled, it is evaluated using measures described in Section 6.2.5.

It is very important to highlight here that labelled and unlabelled datasets never overlap each other *i.e.* they never belong to the same period of time (day, week,

month). For example, if the labelled dataset contains 2 weeks of training data, the unlabelled dataset belongs to a different week of testing data (which is never before observed by the Deep Learning machine). In fact, this is also different from a k-fold cross-validation scheme in which training and testing datasets are chosen from within the same dataset.

Deep Learning and other medium and sometimes shallow NNs are mostly inferred as black boxes. Although they have superior accuracy, their validation solely relies on empirical evidence and not theoretical proof. It is very difficult in case of complex NNs to trace a prediction back to an important feature or to understand the reason of an outcome over another. My method, which relies on the same Deep Learning technique, is not free from this well-known and documented intrinsic behaviour [55, 78].

For the sake of conducting tests, multiple intersections across the city were randomly selected as data sources. All of them were tested individually and results were very consistent across all intersections. In the following, as an example, the intersection between Corso Giovanni Agnelli (CGA) and Via Filadelfia (VF) is chosen as the data source. At this intersection, two data sources are available in the open data, specifically for northbound traffic (towards city centre) and southbound traffic (away from the city centre). The data source with northbound traffic was selected for the discussion that follows.

6.2.1 Pre-processing

To further enhance and enrich the collected data, some pre-processing operations are performed. Their description is provided below.

Date to numerical

Date to numerical converts a text-based timestamp into an integer number indicating the minutes from the origin of the dataset. For example, in the case of weekdays and weeks dataset, 00:00 Monday is set to number 0 and 00:05 Monday to 5 and so on. For weekends dataset, 0 is set to Saturday midnight. This allows the machine to treat the string-based timestamp attribute as an index.

Replace missing values

Replace missing values fills in gaps in a series due to missing data. A simple linear interpolation is applied to the data if one value is missing in the series. If more than one value is missing, the missing portion is replaced with the minimum value of the series.

Moving average

Moving average is commonly used with time series data to smooth out short-term fluctuations and highlight long-term trends or cycles. In other words, moving average works like a low-pass filter. A triangular weighted window function is applied to the data with a window width of five ($5 \times 5\ mins = 25\ mins$) and the result of the weighted average is inserted at the end of the window.

Time series windowing

Time series windowing is a very popular pre-processing step used in time series analysis and prediction. It takes any time series data and transforms it into a cross-sectional format. It essentially converts time values into cross-sectional attributes on which any predictive modelling algorithm can be used to predict future values. In other words, it transforms the given example set containing series data into a new example set containing single valued examples. For this purpose, windows with a specified *window size* (the width of the used window) and *step size* (the distance between the first values) are moved across the series and the attribute value lying *horizon* (the distance between the last window value and the value to predict) values after the window end is used as the label which should be predicted.

As an example, in case of the vehicular flow time series of one data source, a window size of six, a step size of one, and a horizon of one would mean:

- The window considers the vehicular flow of last 30 mins;
- The window is moved 5 mins per step;
- The window is used to predict the vehicular flow of the next 5 mins.

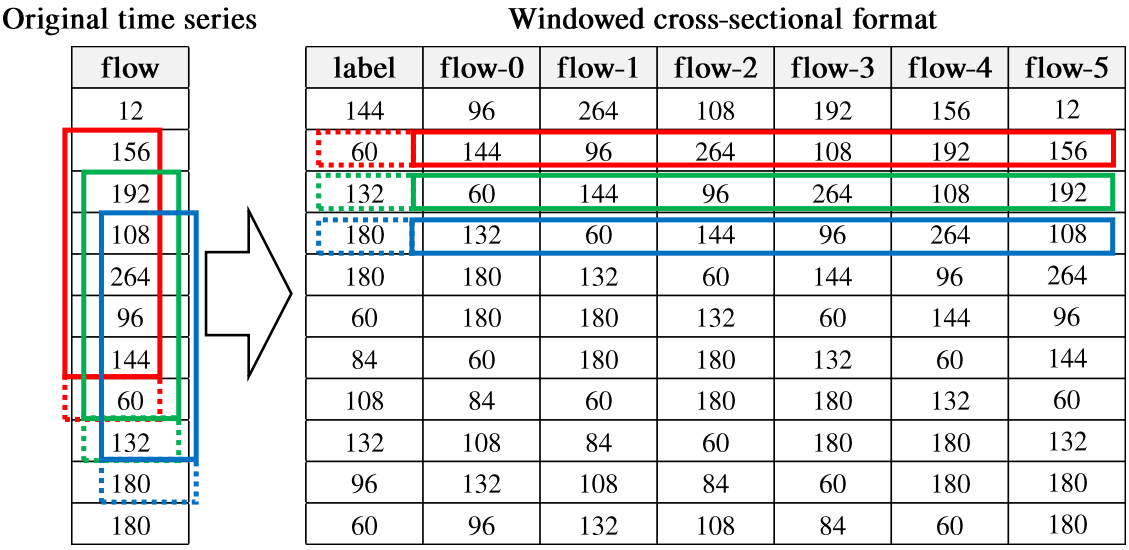
Figure 6.5 shows a visual representation of the same example. The solid line boxes represent the input values while the dotted box represents the future value to be predicted.

Shuffled order

Shuffled order pre-processing step shuffles the order or sequence of a windowed data table in a random fashion. The purpose of adding a shuffled order pre-processor is to further verify that the learner can correctly predict the traffic flow without timestamp even when the order of samples is shuffled (after windowing). Figure 6.6 shows a sample windowed data table randomly shuffled.

6.2.2 Pre-processing schemes

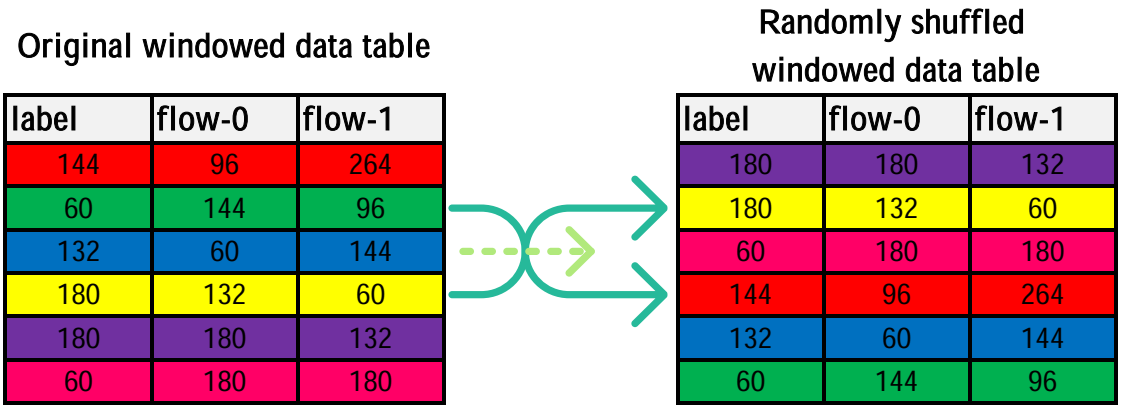
Experiments with six different combinations of pre-processing schemes were performed to investigate the best approach for traffic prediction, including:



flow
12
156
192
108
264
96
144
60
132
180
180

label	flow-0	flow-1	flow-2	flow-3	flow-4	flow-5
144	96	264	108	192	156	12
60	144	96	264	108	192	156
132	60	144	96	264	108	192
180	132	60	144	96	264	108
180	180	132	60	144	96	264
60	180	180	132	60	144	96
84	60	180	180	132	60	144
108	84	60	180	180	132	60
132	108	84	60	180	180	132
96	132	108	84	60	180	180
60	96	132	108	84	60	180

Figure 6.5: An example of original time series to windowed cross-sectional format conversion.



label	flow-0	flow-1
144	96	264
60	144	96
132	60	144
180	132	60
180	180	132
60	180	180

label	flow-0	flow-1
180	180	132
180	132	60
60	180	180
144	96	264
132	60	144
60	144	96

Figure 6.6: An example of random shuffling of windowed data table.

A. *Standard*: This scheme involves no special pre-processing other than converting the date to a number, replacing missing values and windowing.

B. *Moving average*: This scheme uses data passed through the date to a number, replacing missing values, moving average and windowing blocks.

C. *Moving average without timestamp input*: The dataset for this test does not contain any attribute related to time. The pre-processing steps include replacing missing values, moving average and windowing.

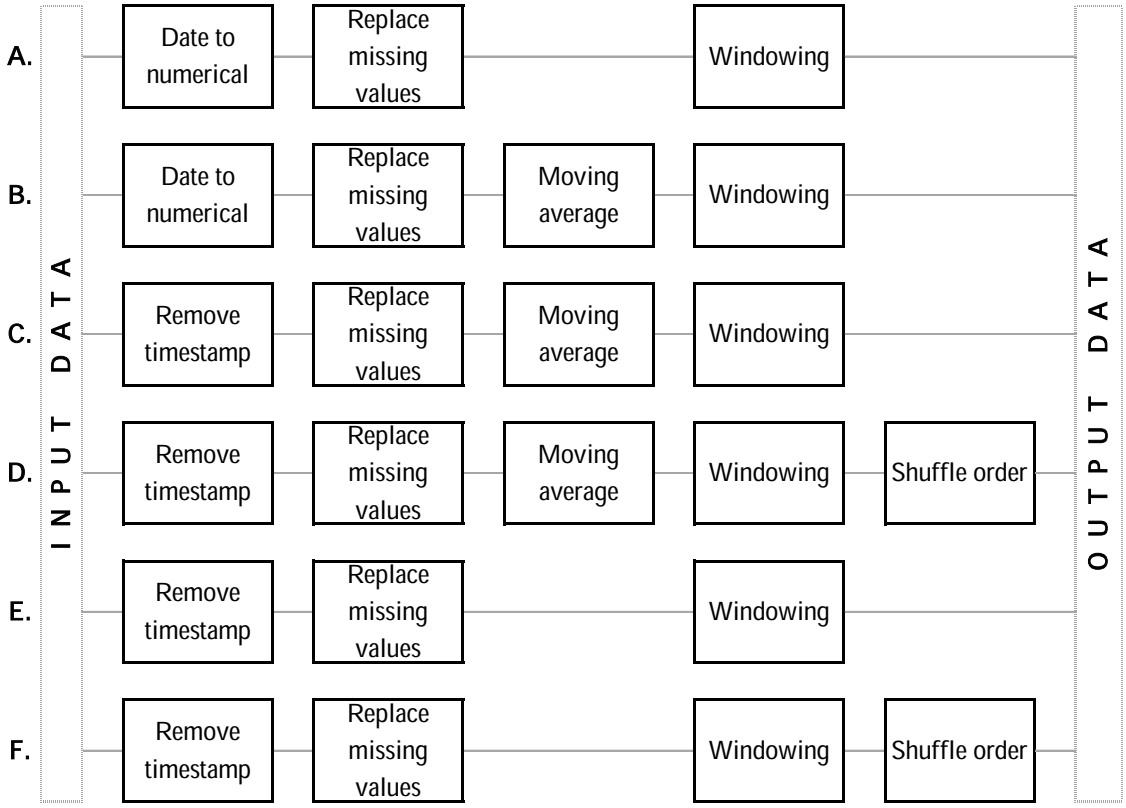


Figure 6.7: A comparison of all pre-processing scheme pipelines.

D. *Moving average without timestamp input and shuffled order*: The dataset for this test does not contain any attribute related to time. At the testing stage, the test dataset is shuffled to remove any chronology associated with it. The pre-processing steps include replacing missing values, moving average and windowing.

E. *No timestamp input*: The dataset for this test does not contain any attribute related to time. The pre-processing steps include replacing missing values and windowing.

F. *No timestamp input with shuffled order*: The dataset for this test does not contain any attribute related to time. At the testing stage, the test dataset is shuffled to remove any chronology associated with it. The pre-processing steps include replacing missing values and windowing.

Figure 6.7 shows a comparison between all pre-processing scheme pipelines. Note that the alphabets on the left of the Figure correspond with items in the list.

Hyper-parameter	Grid range			
	Min	Max	Steps	Scale
Window size	2	12	10	Linear
Epochs	10	200	10	Linear
Learning rate	0.0	1.0	5	Linear

Table 6.1: Grid search based hyper-parameter optimisation configuration.

6.2.3 Hyper-parameter optimisation

A very important step involved in machine learning is tuning the model and the pre-processing parameters. *Hyper-parameter optimisation* is the process of choosing a set of optimal hyper-parameters for a learning algorithm. A *hyper-parameter* is a parameter whose value is set before the learning process begins. By contrast, the values of other parameters are derived via training. An exhaustive grid search based hyper-parameter optimisation was used to optimise the window size of windowing pre-processor and the number of epochs and learning rate of the Deep Learning machine. The loss function is based on Mean Relative Error (MRE). The grid search configuration for all hyper-parameters is shown in Table 6.1. This results in 726 total combinations of hyper-parameters per pre-processing scheme. The best combination of hyper-parameters is chosen based on the minimum value of MRE. Please note that the number of hidden layers and the number of neurons per hidden layer are fixed to two and 50, respectively.

6.2.4 Software configuration

For the experimentation, RapidMiner version 7.6 [65] was used. RapidMiner is a data science software platform that provides an integrated environment for data preparation, machine learning, Deep Learning, text mining, and predictive analytics. The specific implementation of Deep Learning algorithm used in these tests is included in H_2O 3.8.2.6 [39]. H_2O is an open source in-memory platform for distributed and scalable machine learning.

6.2.5 Error measures

The error measures used to evaluate the accuracy of the predictions are listed below.

- **RMSE** is the averaged root-mean-squared error such that:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{f}_i - f_i)^2} \tag{6.1}$$

where $\hat{f}$ is the predicted value, f is the true value and N is the total number of readings.

- **MRE** is the averaged absolute deviation of the prediction from the actual value divided by the actual value such that:

$$MRE = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{\hat{f}_i - f_i}{f_i}\right| \tag{6.2}$$

where $\hat{f}$ is the predicted value, f is the true value and N is the total number of readings.

- **Correlation (COR)** is the averaged correlation coefficient between the label and prediction attributes.

6.3 Results

The results for six different pre-processing schemes applied to three different sets of data is presented. Table 6.2 shows the summary of all test results for all schemes. The results presented in the table are the best results after hyper-parameter optimisation for each pre-processing scheme. In general, the machines can follow the general trend of vehicular flow and the results are quite promising.

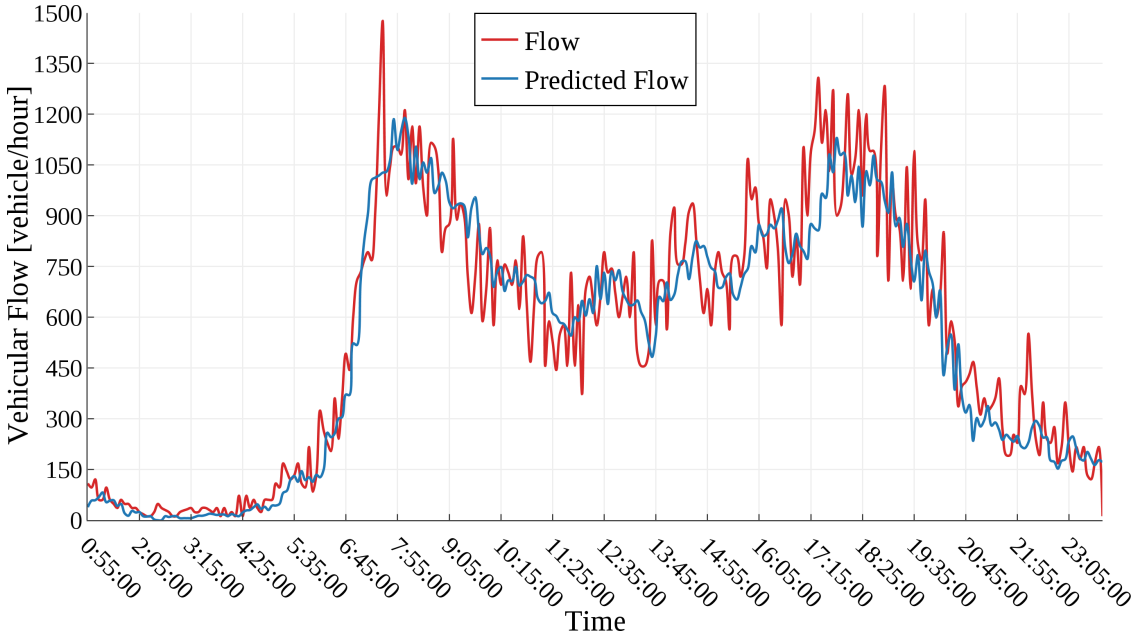
The moving average pre-processor improves the accuracy of the predictions significantly. On average for all datasets, RMSE, MRE and COR for all schemes without moving average are 120.82, 27.69% and 93.51% respectively. The same values for schemes with moving average improve as much as 36.29, 8.15% and 99.38%. This constitutes up to 70% improvement in RMSE and MRE and 6% improvement in COR due to moving average. Figure 6.8b and Figure 6.8a shows the comparison between predicted and true value of vehicular flow for the same dataset with and without moving average respectively.

The machine can generate an accurate model of the data with and without timestamp as an input. On average, for all datasets, it shows a degradation of only 0.9% in MRE without timestamp as an input. This implies that the machine can model traffic without being aware of the time domain at all. It is possible to generate a very high accuracy prediction just by observing a few samples of vehicular flow in the past. Also, by removing time as an input attribute to the learner, it simplifies the complexity of the learner, resulting in faster training and predictions.

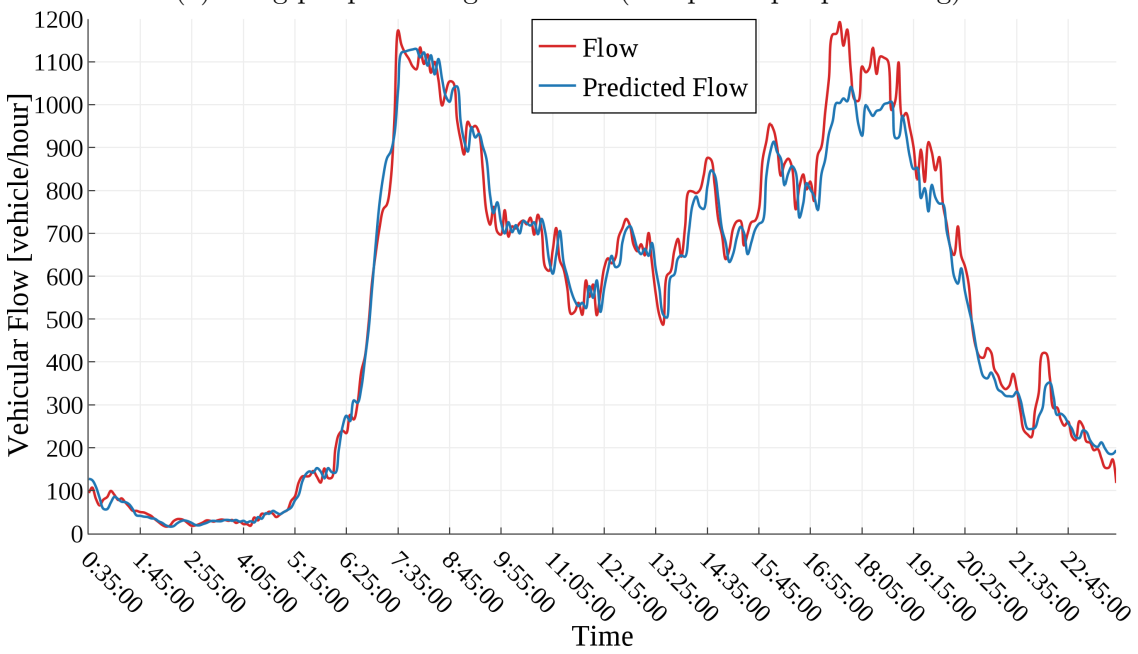
Figure 6.10 shows the comparison between the predicted and true value of vehicular flow for the same dataset with moving average and without timestamp as an input. In this case, a single learner was trained to predict the entire week's

Training data	Testing data	Moving average	No timestamp input	Shuffled order	Window size [min]	RMSE	MRE	COR
2 weekdays	1 weekday				55	134.50	24.50%	93.60%
		✓			45	47.67	7.36%	99.33%
		✓	✓		60	40.41	11.00%	99.53%
		✓	✓	✓	15	45.73	8.92%	99.24%
			✓		20	126.51	28.52%	94.31%
			✓	✓	60	135.29	28.85%	94.27%
2 weekends	1 weekend				30	122.93	27.22%	91.81%
		✓			30	26.69	9.05%	99.42%
		✓	✓		25	24.22	5.39%	99.53%
		✓	✓	✓	35	35.01	8.23%	98.99%
			✓		15	100.11	27.27%	92.60%
			✓	✓	20	97.66	27.75%	92.71%
2 weeks	1 week				40	129.99	29.03%	93.54%
		✓			40	21.05	5.80%	99.84%
		✓	✓		25	43.50	8.79%	99.27%
		✓	✓	✓	30	42.34	8.82%	99.27%
			✓		35	118.40	27.60%	94.56%
			✓	✓	60	121.98	28.47%	94.23%

Table 6.2: Summary of vehicular flow prediction errors on different datasets using Deep Learning with different preprocessing schemes.



(a) using pre-processing scheme A (No special pre-processing)



(b) using pre-processing scheme B (Moving average)

Figure 6.8: Predicted traffic flow vs actual traffic flow for CGA intersection VF northbound dataset consisting of a weekday.

vehicular flow. Please note that the x-axis of the Figure was just added for illustration purposes since the actual dataset does not contain any time frame reference.

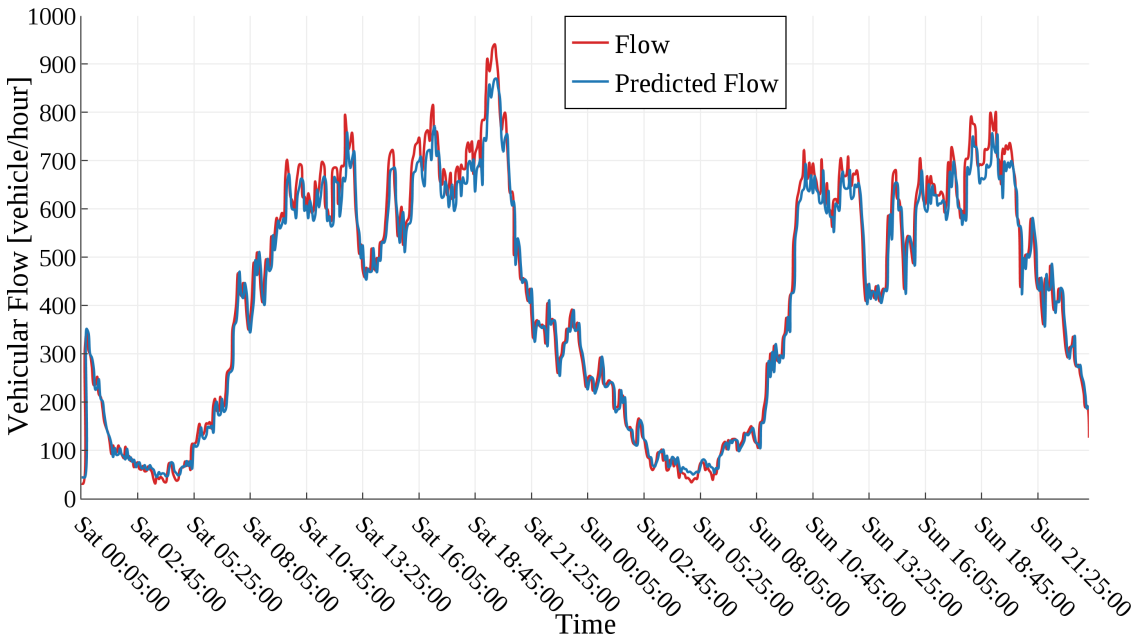


Figure 6.9: Predicted traffic flow vs actual traffic flow for CGA intersection VF northbound dataset consisting of weekends (Saturday and Sunday) using pre-processing scheme B (Moving average).

The best overall results are achieved by pre-processing schemes B and C.

My assumption is that the learner can correctly predict the flow of traffic without timestamp or index or any information about the order or sequence of a prediction. Results show that this assumption is true since on average for all datasets, with and without shuffled order schemes show negligible difference in error measures (4 RMSE, 0.4% MRE and 0.18% COR). This proves that the learner predicts every prediction individually and does not rely on previous or future predictions.

As far as the window size is concerned, results show that it does not have a major impact on the performance of the learner. The best three combinations of hyper-parameters show negligible difference in error measures with a major variation in the value of window size. Overall, in my experience, a window size of 10 to 30 minutes is reasonable for generating high accuracy predictions.

To further demonstrate the robustness of my approach, a cross-examination among different datasets was performed. A machine trained with one type of dataset (such as two weekdays) was used to predict the traffic flow of a different dataset (such as a weekend). Table 6.3 shows a summary of best results achieved after this cross-examination. Please note that all tests are performed using pre-processing scheme C (moving average without timestamp input). Results suggest that a machine trained with an entire week of traffic flow is the most suited to predict traffic flow on any day of the week. This hypothesis is confirmed by the

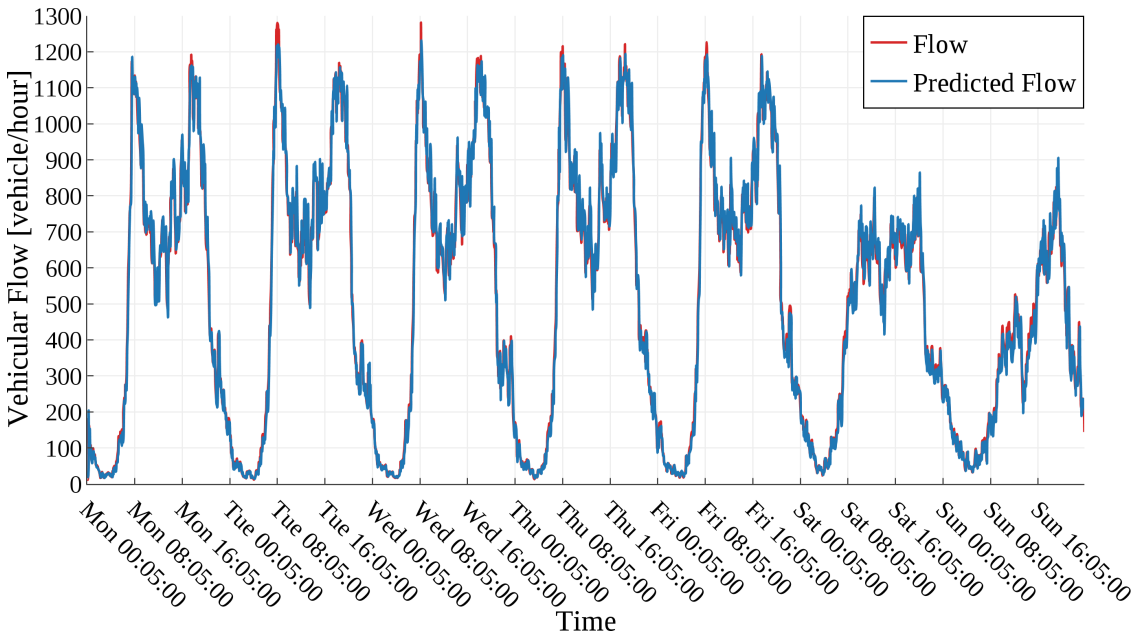


Figure 6.10: Predicted traffic flow vs actual traffic flow for CGA intersection VF northbound dataset consisting of an entire week using pre-processing scheme C (Moving average without timestamp input).

Training data	**Testing data**	**RMSE**	**MRE**	**COR**
2 weekdays	1 weekend	36.41	8.49%	98.90%
	1 week	43.30	9.03%	99.20%
2 weekends	1 weekday	61.97	10.57%	98.90%
	1 week	64.61	10.65%	98.70%
2 weeks	1 weekday	41.78	7.99%	99.30%
	1 weekend	35.60	8.19%	99.00%

Table 6.3: Summary of vehicular flow prediction errors with cross-examination among different datasets using Deep Learning.

machine trained using a set of data covering two full weeks instead of two weekdays or two weekends. This machine shows an improvement of on average 12.88 in RMSE, 1.6% in MRE and 0.23% in COR.

6.3.1 Comparison with state-of-the-art

In comparison, the best results achieved by Deep Learning Architecture (DLA) based forecasting method by [48] are around 90% Mean Accuracy (MA) ($MA = 1 - MRE$) using three hidden layers with 128 hidden units each and 40 epochs with data aggregated using a window size of 60 mins. Authors claim that these results are better than those of the ARIMA model [7] (which is a simple model-based predictor using auto-regression and moving average over past data values), the Bayesian model [72], the SVR model [8], the Locally Weighted Learning (LWL) model [67], the multivariate non-parametric regression model [10], the NN model [69], and the NN-S model [46]. My solution outperforms these results with MA of 94.2% while being much less complex (2 hidden layers with 50 hidden units each with data aggregated using a window size of 25).

The best results achieved by SAE based Deep Learning forecasting method by [83] are around 6.48% MRE using 3 hidden layers with 400 hidden units each with data aggregated using a window size of 15 mins. According to authors, these results are better than those achieved by the Back Propagation Neural Network (BPNN), the Random Walk (RW), the Support Vector Machine (SVM) and the Radial Basis Function Neural Network (RBFNN). Again, my solution outperforms these results in terms of higher accuracy and lower complexity (MRE of 5.8%; two hidden layers with 50 hidden units each).

6.4 Conclusions

This Chapter proposes a system for forecasting urban traffic over a short time period using Deep Learning. The prediction is accurate while being simple in terms of complexity. It also proposes and analyses the effects of multiple pre-processing schemes to improve the accuracy of forecasting. From experiments on a real traffic flow dataset from the City of Turin, it is evident that my Deep Learning machine performs better than state-of-the-art DLAs. Results show that my solution outperforms other DLAs with nearly 4% accuracy improvements while being much simpler in terms of complexity (hidden layers and hidden neurons). The most effective way to pre-process data is to use a simple moving average without timestamp as an input. This means that the machine can generate a prediction with only the traffic flow of past few minutes without any knowledge of the time. Moreover, to establish the robustness of my approach, a cross-examination of my architecture with different datasets was done. Results show that a single Deep Learning machine with only two hidden layers of 50 units each trained with traffic flow data of a week can predict the flow on any day of the week as well as holidays with remarkable accuracy.

The results are not only accurate, but they have significant applications. By accurately determining the intensity of traffic in the near future, ATCS can optimise

the traffic light control programme to minimise congestion. Moreover, when applied at multiple intersections, a forecasted traffic congestion can be distributed over multiple intersections to lower the overall impact. Furthermore, when using a multi-plan based traffic light control (*e.g.* low, medium, high ...) for an urban area, accurate predictions can be used as an indication to swap the current plan with a more appropriate plan. Although my focus was on the forecasting of urban traffic, the devised methodology for pre-processing and prediction of traffic intensity time series can be easily adapted to other time series with minor modifications as per the domain of prediction.

Chapter 7

Traffic forecasting with VIL

The concept of VIL for real-time traffic sensing was introduced in Chapter 5 and the use of machine learning techniques for traffic forecasting in Chapter 6. The next question to be investigated is if VIL, operating alongside with traffic prediction, would work with real traffic flow on a real road network. To validate this idea, a setup site needs to be installed and data should be collected from vehicles. The validation was done using simulation tools due to the complexity and cost of such a demanding task. To make the simulation scenario as close as possible to reality, part of a real road network was modelled and simulated with real traffic flow. Later, data was collected from this simulation and various tests were performed to find the best methodology for traffic forecasting in a semi-realistic scenario.

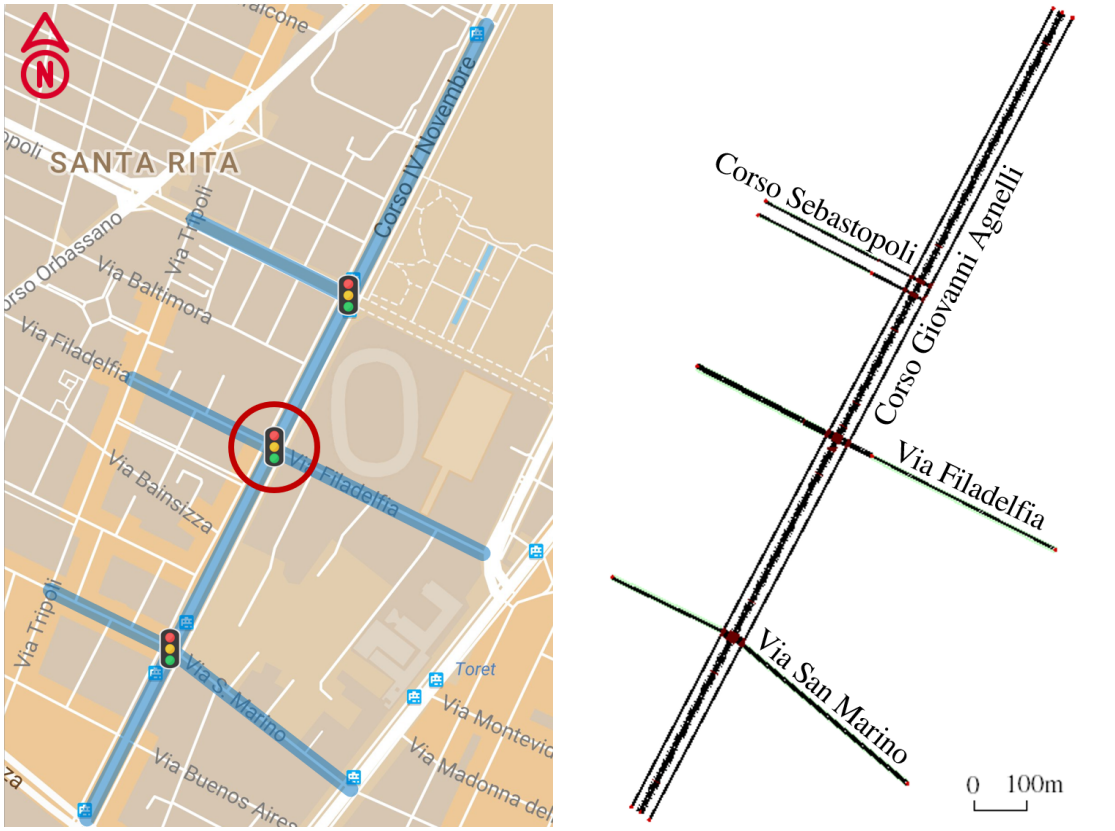
Several relevant traffic sensing state-of-the-art techniques are mentioned in Chapter 6. There are other traffic sensing techniques as well that rely on a completely different methodology of detection and prediction. One recent example is the use of distributed optical fibre belonging to an operational telecom network to detect vehicle speed and density [79]. However, the focus of this work is on the use of machine learning techniques applied to the data collected from smartphones for traffic sensing.

7.1 Simulation setup

The entire simulation setup is described in details in the following sections. In summary, the steps involved are simulation network creation, traffic routes generation and data collection scripting and simulation run.

7.1.1 Network creation

In order to to create a realistic simulation network for Simulation of Urban MObility (SUMO), the basic road network structure was exported from OpenStreetMap (OSM). OSM is a free and editable collaborative mapping project. For



(a) Selected traffic network for traffic simulation marked on Google Maps.

(b) Reconstructed simulation network from the real network on SUMO.

Figure 7.1: Visual comparison of simulation network.

this part of the work, the intersection between Corso Giovanni Agnelli (CGA) and Via Filadelfia (VF) and traffic light were selected. To make the traffic queues at the traffic light more realistic, the intersections and traffic lights between CGA and Corso Sebastopoli (CS) on one side and CGA and Via San Marino (VSM) on the other side were included. Figure 7.1a shows the section of network of interest. SUMO provides a useful script called *osmWebWizard.py* [17]. This script opens a web browser and allows selecting a geographic region on a map to be converted to a SUMO compatible simulation network.

The exported network was manually cleaned to keep only the main features of the network including CGA, VSM, VF and CS. Again, to better reflect the ground truth, some features to the network were added including:

1. *Early convergence:* Due to the nature of the intersections and the width of lanes on CGA, moving traffic usually moves in a single lane, while queues are double-laned. Moreover, when vehicles start moving after a traffic light, they usually merge together in a single lane after a few metres. To mimic

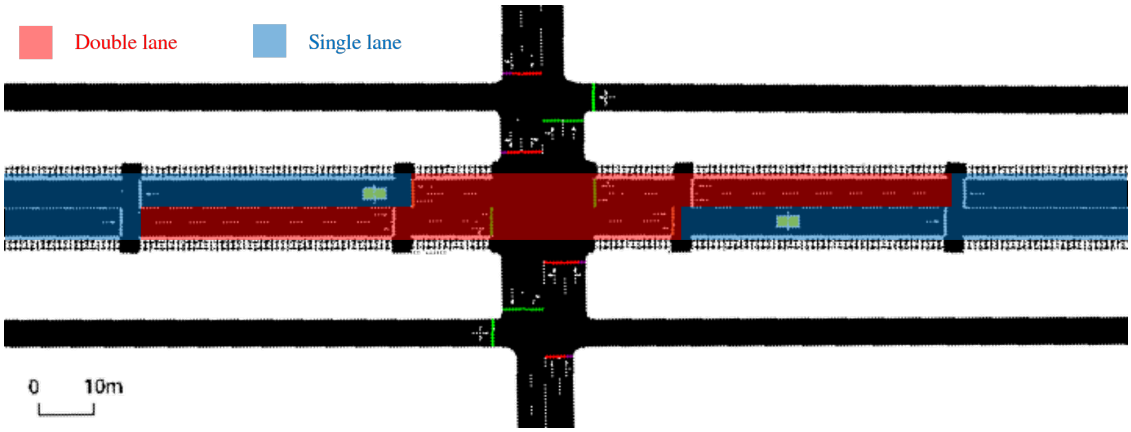


Figure 7.2: SUMO simulation network: Early convergence at CGA and VF intersection.

this driver's behaviour in the simulation model, early convergence at the intersection was added. The main segments of CGA are single-laned while the waiting queue area and entry into the next segment right after the intersection is double laned. Figure 7.2 represents the two areas with different colours.

2. *Neighbour lanes:* To enable opposite-direction-driving on the segments on CGA between all traffic lights, the adjacency information for opposite direction lanes was defined in the network file. This was done using the <*neigh*> element in the edge file.

3. *Parking lanes on secondary entry roads:* the secondary entry roads like VSM, VF and CS have diagonal and parallel parking lanes on them. These features are also represented in the simulation network.

4. *Traffic light phases:* Three traffic light plans are designed to ensure a smooth flow of traffic. Table 7.1, Table 7.2 and Table 7.3 shows the plans for intersections at VF, VSM and CS respectively. Figure 7.3 shows the TLS link index numbers as an example. The other two intersections are also numbered in a similar fashion (clockwise; outwards). All three TLSs have a period of 99 sec. The time offset between VSM and VF is 30 sec while it is 22 sec between VF and CS. Please refer to SUMO's documentation for the meaning of TLS characters [18].

The final simulation networks looks like Figure 7.1b. Figure 7.4 provides a close up look of the intersection between CGA and VF. The final simulation network file is publically available [42].

TLS link index																																											Phase duration [s]
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	
G	G	G	G	s	r	r	r	r	r	G	G	G	G	s	r	r	r	r	r	G	G	G	G	g	g	g	s	r	r	s	r	r	G	G	G	G	G	g	g	g	g	g	60
Y	Y	Y	Y	s	r	r	r	r	r	Y	Y	Y	Y	s	r	r	r	r	r	Y	Y	Y	Y	g	g	g	s	r	r	s	r	r	Y	Y	Y	Y	Y	g	g	g	g	g	3
s	r	r	r	G	G	G	G	g	g	s	r	r	r	G	G	G	G	g	g	s	r	r	r	G	G	g	G	G	G	G	G	G	s	r	r	r	r	G	g	g	G	g	15
s	r	r	r	G	G	G	G	g	g	s	r	r	r	Y	Y	Y	Y	y	y	s	r	r	r	G	G	g	Y	Y	Y	G	G	G	s	r	r	r	r	Y	y	y	Y	g	3
s	r	r	r	G	G	G	G	G	G	s	r	r	r	r	r	r	r	r	r	s	r	r	r	G	G	G	s	r	r	G	G	G	s	r	r	r	r	r	r	r	r	g	15
s	r	r	r	Y	Y	Y	Y	Y	Y	s	r	r	r	r	r	r	r	r	r	s	r	r	r	Y	Y	g	s	r	r	Y	Y	Y	s	r	r	r	r	r	r	r	r	g	3

Table 7.1: Traffic light phase plan for intersection of CGA and VF.

TLS link index																																		Phase duration [s]
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	
G	G	G	s	r	r	r	r	r	G	G	G	s	r	r	r	r	r	G	G	G	g	s	r	r	s	r	r	G	G	G	G	g	g	60
Y	Y	Y	s	r	r	r	r	r	Y	Y	Y	s	r	r	r	r	r	Y	Y	Y	g	s	r	r	s	r	r	Y	Y	Y	Y	y	g	3
r	r	r	s	s	r	r	r	r	s	r	r	G	G	G	G	G	G	r	r	r	g	G	G	G	s	r	r	s	r	r	r	G	G	15
r	r	r	s	s	r	r	r	r	s	r	r	Y	Y	Y	Y	Y	Y	r	r	r	g	Y	Y	Y	s	r	r	s	r	r	r	Y	g	3
s	r	r	G	G	G	G	G	G	s	r	r	r	r	r	r	r	r	r	r	r	G	s	r	r	G	G	G	s	r	r	r	r	g	15
s	r	r	Y	Y	Y	Y	Y	Y	s	r	r	r	r	r	r	r	r	r	r	r	g	s	r	r	Y	Y	Y	s	r	r	r	r	g	3

Table 7.2: Traffic light phase plan for intersection of CGA and VSM.

TLS link index																																																												Phase duration [s]
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	
G	G	G	G	s	r	r	r	r	r	G	G	g	g	G	G	g	g	G	G	G	G	s	r	r	r	r	r	G	G	G	g	g	g	g	g	g	G	g	g	G	G	G	G	g	g	G	G	G	s	r	r	r	r	G	G	g	g	g	g	60
Y	Y	Y	Y	s	r	r	r	r	r	Y	Y	y	y	Y	Y	y	y	Y	Y	Y	Y	s	r	r	r	r	r	Y	Y	Y	g	g	g	g	g	g	G	g	g	G	G	G	Y	y	y	Y	Y	Y	s	r	r	r	r	Y	Y	g	g	g	g	3
s	r	r	r	G	G	G	G	G	G	r	r	r	r	r	r	r	r	s	r	r	r	G	G	G	G	G	G	s	r	r	G	g	g	G	G	G	G	g	g	G	G	G	g	g	g	g	g	g	G	G	g	g	G	r	r	g	g	G	G	33
s	r	r	r	Y	Y	Y	Y	Y	Y	r	r	r	r	r	r	r	r	s	r	r	r	Y	Y	Y	Y	Y	Y	s	r	r	Y	y	y	Y	Y	Y	G	g	g	G	G	G	g	g	g	g	g	g	Y	Y	y	y	Y	r	r	g	g	G	G	3

Table 7.3: Traffic light phase plan for intersection of CGA and CS.

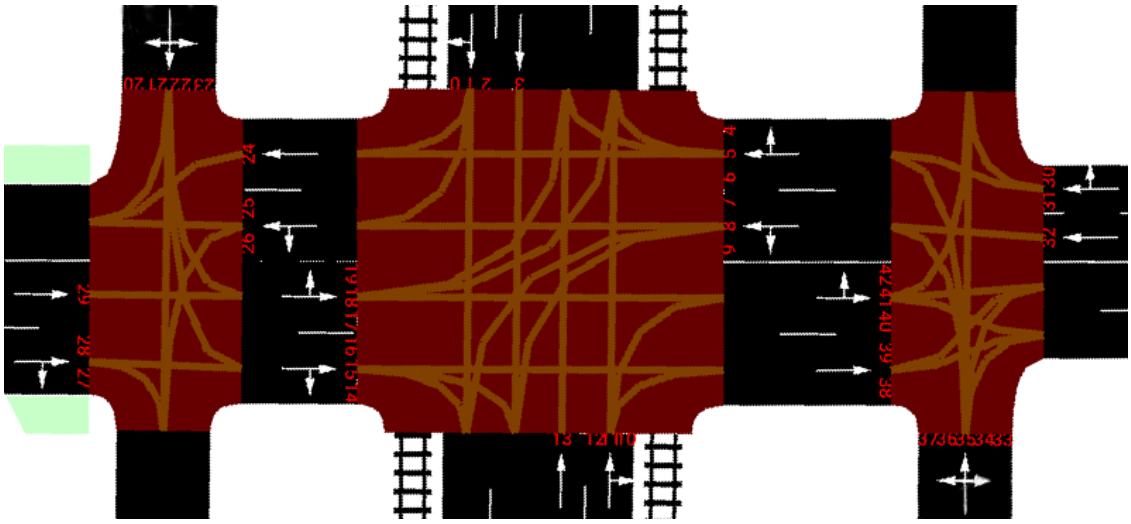


Figure 7.3: Traffic Light Sequence (TLS) link indexes between CGA and VF intersection. Figure also highlights all inter-lane connections.

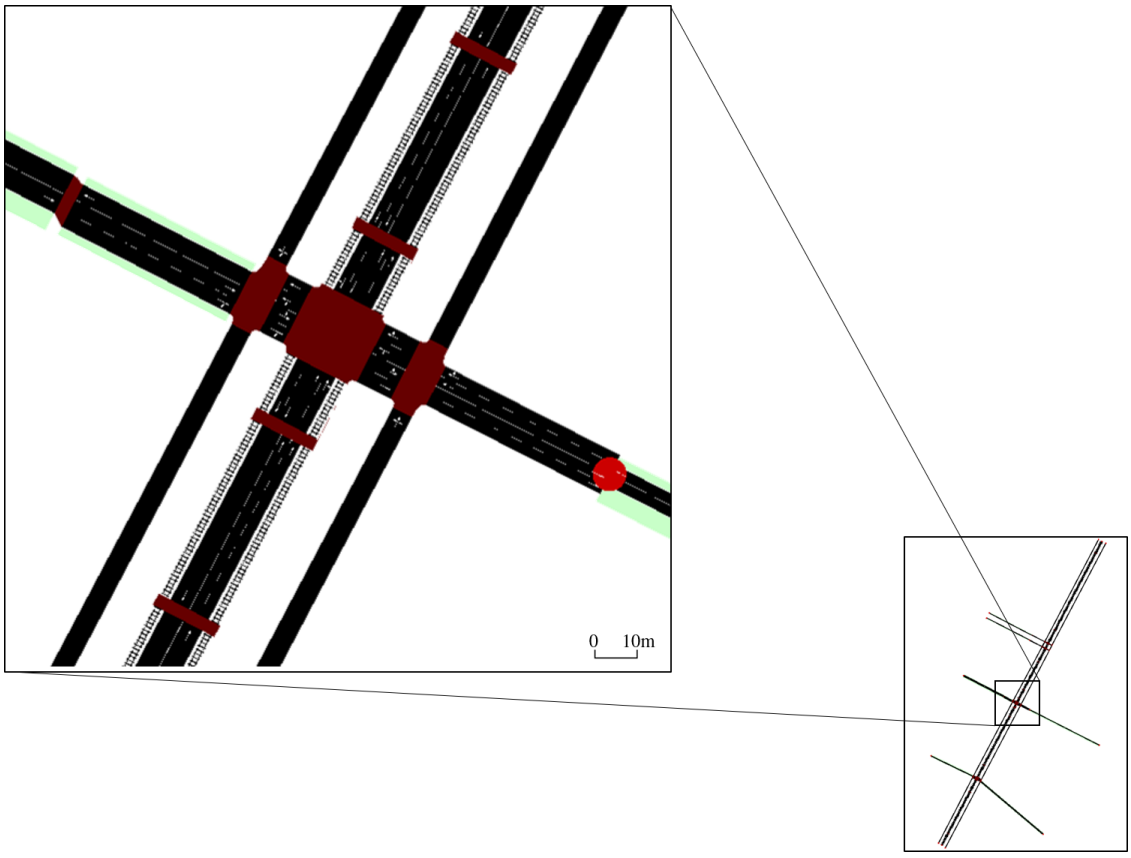


Figure 7.4: SUMO simulation network: Zoom up view of CGA and VF intersection.

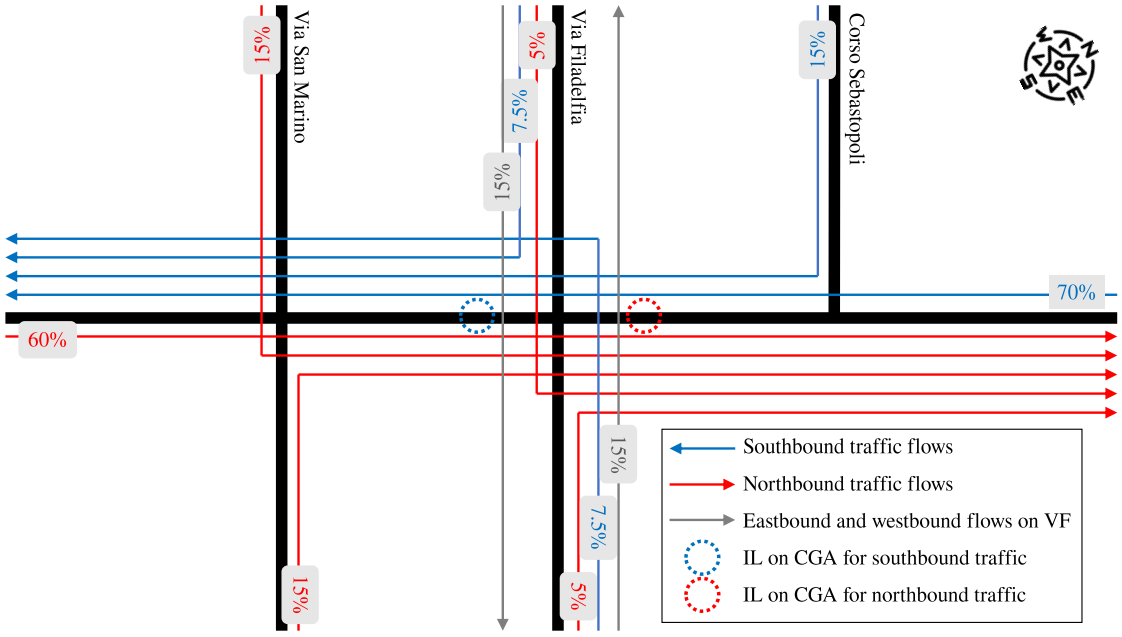


Figure 7.5: SUMO simulation network: Distribution of traffic flow.

7.1.2 Routes generation

In 5T open traffic data, there are two ILs available on CGA (intersection with VF) for north and southbound traffic. The real traffic intensities of these ILs are broken down into multiple flows emerging from VSM, VF and CS as shown in Figure 7.5. In the Figure, all blue flows are monitored by the southbound IL and red flows by northbound IL. Furthermore, to improve the realism in simulation, especially the east and westbound queues on VF, extra traffic flows were added. These extra or dummy flows are arbitrarily chosen to be equal to 15% of the northbound traffic and are not accounted for by any of the ILs. A sample routes file is publically available [42].

7.1.3 Data collection script

Once the simulation network and route files are ready, the simulation is started using a simple Python script. The Python script interacts with SUMO using Traffic Control Interface (TraCI). TraCI allows access to a running road traffic simulation to retrieve values of simulated objects and to manipulate their behaviour online. Every simulation is started with a simulation step size of 1 second and a maximum simulation length of approximately 24 hours. The logic behind the approximation is that all routes are generated from a real 24 hour day. The vehicles generated during the last part of the day may take a few extra minutes to leave the network. Every route file (generated from a specific real day) is assigned a unique random

number to be used as the seed for all random processes in the simulation. The Python script samples the timestamp, ID, speed, x (or latitude) & y (or longitude) positions and bearing of all vehicles at every step of the simulation. The Python script for running the simulation and collecting data is publically available [42].

7.2 Data processing

After the simulation runs are completed, the raw data (contained in the CSV file) collected from SUMO is stored in a MySQL table. The data are processed by virtually placing VILs at multiple points on the network. This entire process described below is done by a data processer which I implemented. The source code for this processor are publically available [41].

7.2.1 VIL positions

All VILs are identified by a latitude, longitude, radius and direction (or bearing). Figure 7.6a shows a simple placement plan of 8 VILs. This plan can be used to calculate average speeds at all VILs as well as the time difference (delta time) between passage of a vehicle over two VILs. Figure 7.6b and 7.6c show more advanced VIL placement plans with 15 and 17 VILs respectively. The *INT* VIL is an omni-directional VIL, hence all vehicles passing through the intersection in any direction are captured by it. All other VILs are uni-directional.

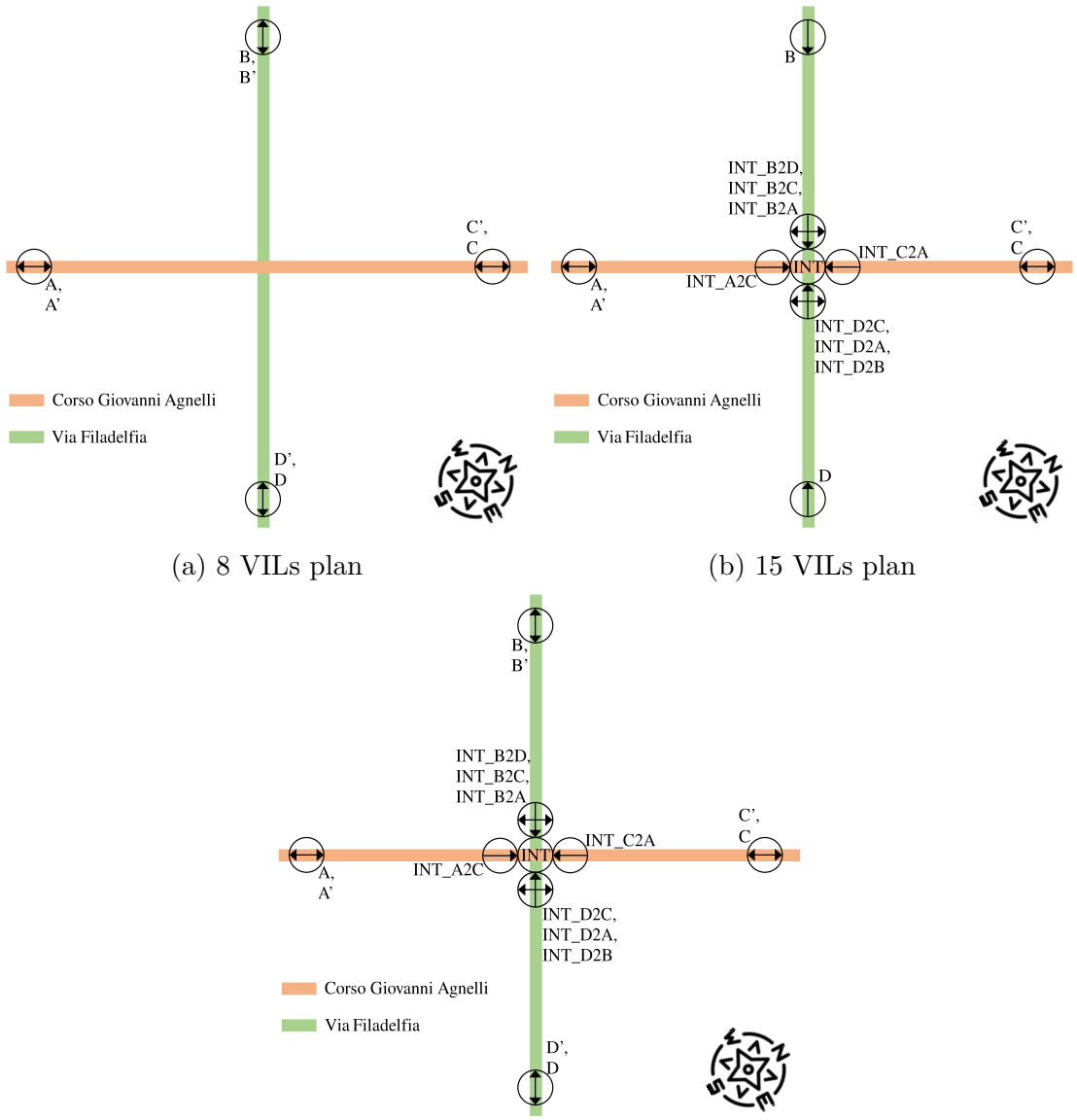
7.2.2 Headings and margins

The intersection between CGA and VF is orthogonal. CGA is oriented at approximately 28° from North while VF is 118°. CGA heading towards North is named as H1, VF heading towards East is named as H2 and so on in a clockwise fashion. The upper and lower bounds for all bearing angles are ±15° (*i.e.* $\alpha = 30°$). Figure 7.7 shows all headings and their upper and lower bounds.

7.2.3 Sample SQL statements

Temporary tables are created from raw simulation data to identify all passages over VILs. Structured Query Language (SQL) statement for creating temporary table containing IDs, transit times and transit speeds of all passing vehicles over a VIL is shown below.

```
DROP TEMPORARY TABLE IF EXISTS TABLE_NAME;
CREATE TEMPORARY TABLE TABLE_NAME ENGINE=MEMORY
SELECT id, AVG(step) AS transitTime, AVG(speed) AS
    transitSpeed
```



(a) 8 VILs plan

(b) 15 VILs plan

(c) 17 VILs plan

Figure 7.6: Multiple VIL plans for CGA and VF intersection.

```
FROM INPUT_TABLE_NAME
WHERE ST_DISTANCE(POINT(positionX, positionY), POINT(VIL_X,
    VIL_Y)) < VIL_RADIUS AND (IF((angle) >= H1_LOWER OR (
    angle) < H1_UPPER, "H1", IF((angle) >= H2_LOWER AND (
    angle) < H2_UPPER, "H2", IF((angle) >= H3_LOWER AND (
    angle) < H3_UPPER, "H3", IF((angle) >= H4_LOWER AND (
    angle) < H4_UPPER, "H4", "HX"))))) LIKE VIL_HEADING
```

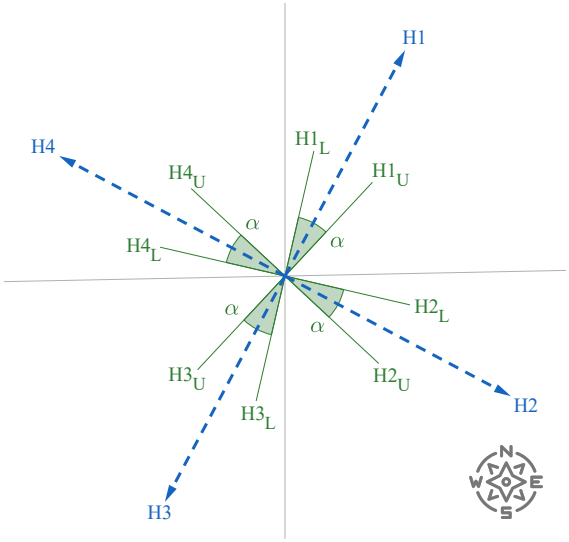


Figure 7.7: Heading names for CGA and VF intersection.

```
GROUP BY id;
```

`TABLE_NAME`, `INPUT_TABLE_NAME`, `VIL_X`, `VIL_Y`, `VIL_RADIUS` and `VIL_HEADING` must be provided to the statement. `Hx_UPPER` and `Hx_LOWER` are shown in Figure 7.7.

VIL data can be calculated after the passages table is ready. SQL statement for calculating average speed of all vehicles passing over a VIL during a given time period is shown below.

```
SELECT AVG(transitSpeed) AS averageSpeed
FROM VIL_TABLE_NAME
WHERE transitTime > STEP_FROM AND transitTime <= STEP_TO
    AND id REGEXP ID_REG_EXP;
```

`TABLE_NAME`, `VIL_TABLE_NAME`, `STEP_FROM`, `STEP_TO` and `ID_REG_EXP` must be provided to the statement.

Similarly, delta of transit time can also be calculated from the passages table. SQL statement for calculating delta of transit time for a vehicle between two VILs during a given time period is shown below.

```
SELECT AVG(EXIT_VIL_TABLE_NAME.transitTime -
    ENTRY_VIL_TABLE_NAME.transitTime) AS deltaTime
FROM EXIT_VIL_TABLE_NAME
LEFT JOIN ENTRY_VIL_TABLE_NAME ON EXIT_VIL_TABLE_NAME.id =
    ENTRY_VIL_TABLE_NAME.id
WHERE EXIT_VIL_TABLE_NAME.transitTime > STEP_FROM AND
    EXIT_VIL_TABLE_NAME.transitTime <= STEP_TO;
```

ENTRY_TABLE_NAME, EXIT_TABLE_NAME, STEP_FROM and STEP_TO must be provided to the statement.

This processing results in two processed data tables; one for average speed and another for delta time. The table for average speed with 15 VILs (similar to Figure 7.6b) has the following structure:

STEP_FROM	VIL-B	VIL-INT	VIL-INT-B2A
STEP_TO	VIL-C	VIL-INT-A2C	VIL-INT-D2A
FLOW_NORTH	VIL-D	VIL-INT-B2C	VIL-INT-B2D
FLOW_SOUTH	VIL-E	VIL-INT-D2C	VIL-INT-D2B
VIL-A	VIL-F	VIL-INT-C2A	

The table for delta time with 8 VILs (similar to Figure 7.6a) has the following structure:

STEP_FROM	FLOW_SOUTH	DT-D2C	DT-B2A
STEP_TO	DT-A2C	DT-C2A	DT-B2D
FLOW_NORTH	DT-B2C	DT-D2A	DT-D2B

For both tables, the data type of all of the columns is FLOAT except for STEP_FROM and STEP_TO which are INT.

7.3 Prediction workflow

The prediction workflow basically involves generating training and testing dataset(s), finding the best model for training dataset and testing the model with testing dataset(s). Figure 7.8 shows prediction workflow for training and predicting traffic intensity from VIL data. The steps included in the prediction workflow are listed below.

1. The labelled training dataset is read from the database.
2. A Gradient Boosted Machine (GBM) is trained with 90% of the training dataset.
3. The trained GBM model is applied to the remaining 10% of training dataset (also called *validation dataset*).
4. Training performance is evaluated (called *validation performance*).
5. Step 2, 3 and 4 are repeated 10 times in order to complete a *10-fold cross-validation* such that if the training dataset is divided into 10 equal subsets, each set is used exactly once as the validation dataset.

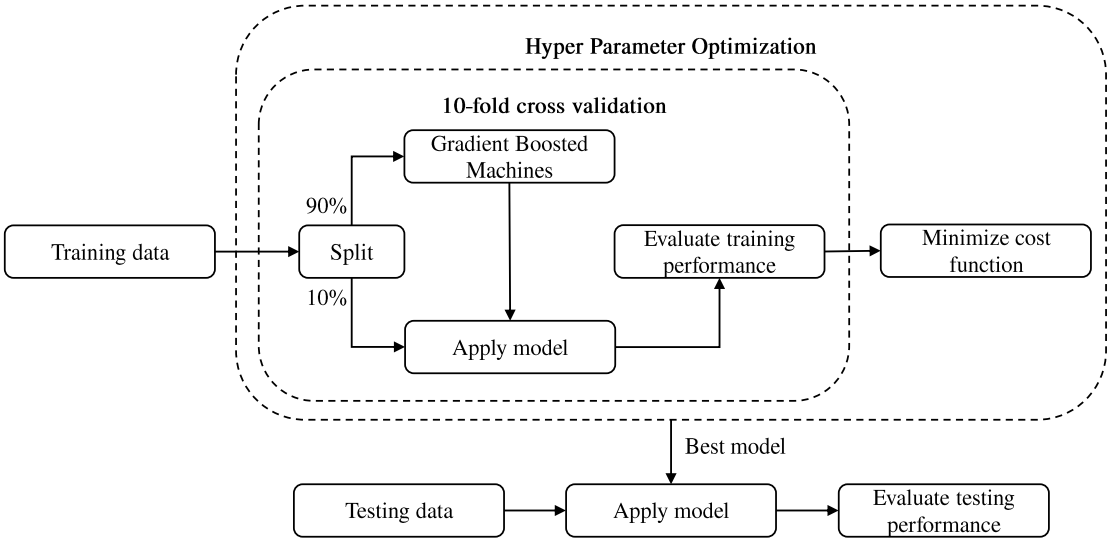


Figure 7.8: Prediction workflow

6. 10 validation performance results are averaged to create the *inner performance*.

7. Step 2, 3, 4, 5 and 6 are repeated using a different combination of hyper-parameters to perform *hyper-parameter optimisation*.

8. The hyper-parameter optimisation continues until the cost function (*i.e.* 1 - inner performance) is minimised. The GBM model trained with a set of hyper-parameters achieving highest performance is called the *best model*.

9. The testing dataset is read from the database.

10. The best model is applied to the testing dataset to generate predictions.

11. Testing performance is evaluated (called *outer performance*).

All training run with a time-out limit of 15 minutes. All tests go through 10-fold cross-validation and use GBM algorithm for training and testing. The motivation of using cross-validation is to test the model's ability to predict new data that was not used in estimating it, to flag problems like overfitting or selection bias [9]. Cross-validation significantly helps to remove biases from learning. The direction of traffic is always northbound. PR is the percentage of vehicles equipped with VIL so that they report data to the system.

7.3.1 Performance measures

Regression

The following performance measures are used to evaluate the accuracy of trained models for all regression-based tasks.

- **RMSE** is the averaged root-mean-squared error such that:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{f}_i - f_i)^2} \tag{7.1}$$

  where $\hat{f}_i$ is the i^{th} predicted value, f_i is the i^{th} true value and N is the total number of readings.

- **Mean Absolute Error (MAE)** is the averaged absolute deviation of the prediction from the actual value such that:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}\left|\hat{f}_i - f_i\right| \tag{7.2}$$

  where $\hat{f}$ is the i^{th} predicted value, f is the i^{th} true value and N is the total number of readings.

- **MRE** is the averaged absolute deviation of the prediction from the actual value divided by actual value such that:

$$MRE = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{\hat{f}_i - f_i}{f_i}\right| \tag{7.3}$$

  where $\hat{f}$ is the i^{th} predicted value, f is the i^{th} true value and N is the total number of readings.

- **Mean Lenient Relative Error (MLRE)** is the averaged absolute deviation of the prediction from the actual value divided by the maximum between the actual value and the predicted value such that:

$$MLRE = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{\hat{f}_i - f_i}{\max(\hat{f}_i, f_i)}\right| \tag{7.4}$$

  where $\hat{f}$ is the i^{th} predicted value, f is the i^{th} true value and N is the total number of readings.

- **Mean Strict Relative Error (MSRE)** is the averaged absolute deviation of the prediction from the actual value divided by the minimum between the actual value and the predicted value such that:

$$MSRE = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{\hat{f}_i - f_i}{\min(\hat{f}_i, f_i)}\right| \tag{7.5}$$

  where $\hat{f}$ is the i^{th} predicted value, f is the i^{th} true value and N is the total number of readings.

- **COR** is the averaged correlation coefficient between the label and prediction attributes such that:

$$corr(f, \hat{f}) = \frac{\sum_{i=1}^{N}(f_i - \overline{f})(\hat{f}_i - \overline{\hat{f}})}{\sqrt{\sum_{i=1}^{N}(f_i - \overline{f})^2 \sum_{i=1}^{N}(\hat{f}_i - \overline{\hat{f}})^2}} \tag{7.6}$$

  where $\hat{f}$ is the i^{th} predicted value, f is the i^{th} true value, $\overline{\hat{f}}$ is the mean of all predicted values, $\overline{f}$ is the mean of all true values and N is the total number of readings.

Classification (Binary class)

The following performance measures are used to evaluate the accuracy of trained models for all binary classification-based tasks.

- **Accuracy (ACC)** is the ratio between the total true positive classifications and negative classifications and total population, such that:

$$ACC = \frac{\sum True\ positive + \sum True\ negative}{\sum Total\ population} \tag{7.7}$$

- **True Positive Rate (TPR)** measures the proportion of positives that are correctly identified, such that:

$$TPR = \frac{\sum True\ positive}{\sum Condition\ positive} \tag{7.8}$$

- **False Positive Rate (FPR)** measures the proportion of negatives that are incorrectly identified as positives, such that:

$$FPR = \frac{\sum False\ positive}{\sum Condition\ negative} \tag{7.9}$$

- **False Negative Rate (FNR)** measures the proportion of positives that are incorrectly identified as negatives, such that:

$$FNR = \frac{\sum False\ negative}{\sum Condition\ positive} \tag{7.10}$$

- **True Negative Rate (TNR)** measures the proportion of negatives that are correctly identified, such that:

$$TNR = \frac{\sum True\ negative}{\sum Condition\ negative} \tag{7.11}$$

7.3.2 Workbench configuration

Hardware

A CentOS based computer with 2 × Intel® Xeon® CPU E5-2640 v4 @ 2.40GHz (10 Cores & 20 Threads each) with 128 GB DDR4 Random-Access Memory (RAM) was used for all training and testing. The operating system was fully up-to-date and throughout the tests, the computer was only utilised by the system or test related processes.

Software

For the experimentation, H_2O version 3.18.0.8 [38] was used. H_2O is an open source in-memory platform for distributed and scalable machine learning. Specifically, H2O - Flow was used which is an interactive notebook [37].

7.4 Results

In the following sections detailed results, analysis and discussion are presented for several data types and test schemes. Overall, regression and classification style prediction tests were conducted. Prediction schemes where training and testing datasets have different simulation seeds but same (static) PR are referred to as Cross Random Seed (CRS) tests. Schemes where training and testing datasets have different simulation seeds and different (but static) PR are called Cross Penetration Rate (CPR) tests. Lastly, to simulate realistic scenarios tests were run with Variable Penetration Rate (VPR) with random simulation seeds.

7.4.1 Basic CRS tests

AS dataset

For the first few tests, traffic patterns were generated for an average weekday by taking an average of all weekdays. This synthetic day was simulated using 3

different random seeds. All models are trained using the dataset of the first two seeds while the last seed (unseen by the model) is used for testing, hence the name CRS. The type of data used here for training and testing is Average Speed (AS). Following are the test configurations:

- PR [%] : 10, 25, 50
- Data : AS
- Step size [s] : 30, 60, 120
- Window size [#] : 5, 10, 20
- Window size [s] : 150, 300, 600, 1200, 2400
- VILs [#] : 15
- Attributes [#] : 75, 150, 300
- Training seed(s) : 100 & 200
- Testing seed(s) : 300

Figure A.1 shows results grouped by PR in box plot representation. As expected, all error measures improve as the PR increases. This trend is clear for all performance parameters.

DT dataset

Tests were performed with Delta Time (DT) data to better understand what type of data for training and testing is best suited. The deep learning machine was trained and tested using the same hypothetical averaged weekdays as Section 7.4.1. Again the machine was tested with a third unseen day. Following are the test configurations:

- PR [%] : 10, 25, 50
- Data : DT
- Step size [s] : 30, 60, 120
- Window size [#] : 5, 10, 20
- Window size [s] : 150, 300, 600, 1200, 2400
- VILs [#] : 8
- Attributes [#] : 40, 80, 160
- Training seed(s) : 100 & 200
- Testing seed(s) : 300

Figure A.2 shows results grouped by PR in box plot representation. As expected, all error measures improve as the PR increases. This trend is clear for all performance parameters. Comparatively speaking, overall better results are achieved using AS as data rather than DT.

AS&DT dataset

To better understand to what extent the performance of the machine can be improved, next tests were performed with both AS and DT data. As before, the

deep learning machine was trained and tested using the same hypothetical averaged weekdays as Section 7.4.1. Again, the machine was tested with a third unseen day. Following are the test configurations:

- PR [%] : 10, 25, 50
- Data : AS&DT
- Step size [s] : 30, 60, 120
- Window size [#] : 5, 10, 20
- Window size [s] : 150, 300, 600, 1200, 2400
- VILs [#] : 17
- Attributes [#] : 85, 170, 340
- Training seed(s) : 100 & 200
- Testing seed(s) : 300

Figure A.3 shows results grouped by PR in box plot representation. As expected, all error measures improve as the PR increases. This trend is clear for all performance parameters. The expectation from these results is that they should be at least as good as results presented in Section 7.4.1, which proved to be true. Using both data types as input does provide some improvement, but not significant. On the other hand, it increases the number of attributes to as much as 340, which incurs a complexity cost for the machine.

7.4.2 Variable importance estimation

In order to reduce system complexity, a variable importance assessment was performed for GBM Algorithm. Figure 7.9 shows a bar chart for scaled attribute importance for all attributes. The list of attributes includes 15 attributes from AS data and 8 attributes from DT data (a total of 23 attributes). Following are the test configurations:

- PR [%] : 10, 25, 50
- Data : AS&DT
- Step size [s] : 30, 60, 120
- Window size [#] : 1
- Window size [s] : 30, 60, 120
- VILs [#] : 17
- Attributes [#] : 23
- Training seed(s) : 100 & 200
- Testing seed(s) : 300

Results show that, in general, AS data are more important than DT data. Moreover, each of the 5 most important attributes have an importance higher than 0.1 regardless of the step size. These attributes include AS-VIL-INT, DT-A2C, AS-VIL-A, AS-VIL-INT-A2C and AS-VIL-C. This resulted in a new type of data, referred to as Important Attributes (IA) composed only of the 5 most important attributes. Figure 7.10 shows VIL implementation plan for collecting data for these

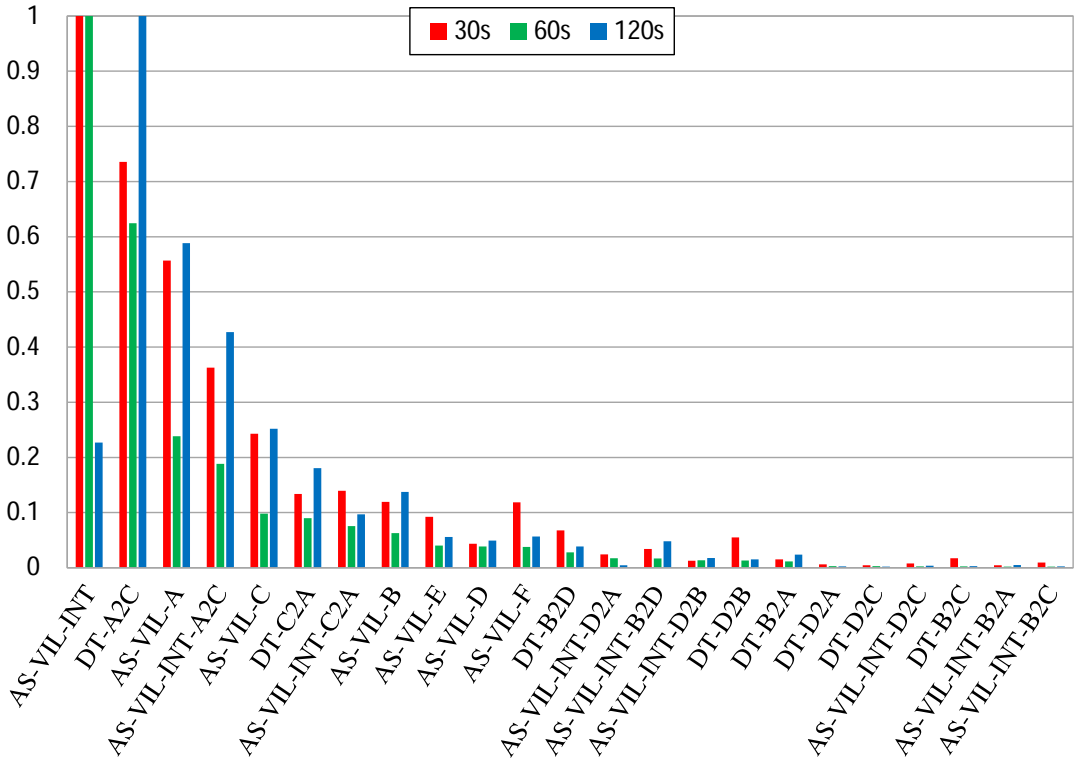


Figure 7.9: Scaled attribute importance for GBM (0 to 1, 1 being highest importance).

important attributes. In total the plan includes 4 uni-directional and 1 omni-directional VILs.

IA dataset

The same tests as Sections 7.4.1, 7.4.1 and 7.4.1 were performed to analyse the possible performance degradation due to the use of IA instead of AS, DT or AS&DT. Following are the test configurations:

- PR [%] : 10, 25, 50
- Data : IA
- Step size [s] : 30, 60, 120
- Window size [#] : 5, 10, 20
- Window size [s] : 150, 300, 600, 1200, 2400
- VILs [#] : 5
- Attributes [#] : 25, 50, 100
- Training seed(s) : 100 & 200
- Testing seed(s) : 300

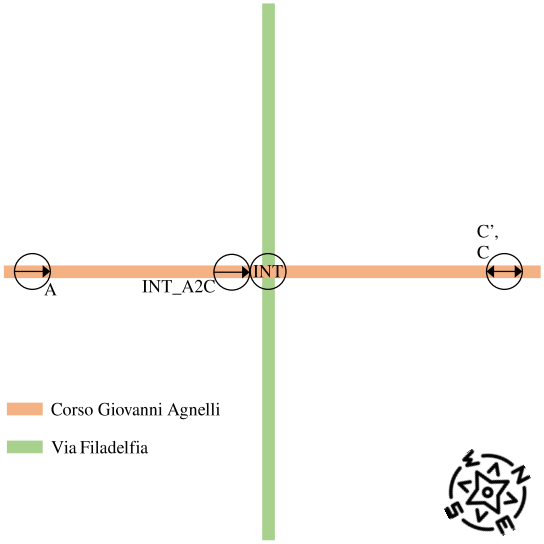


Figure 7.10: VIL plan implementation for 5 most important attributes.

Figure 7.11 shows results grouped by PR in box plot representation. The performance of prediction achieved by IA data is very much comparable to the best performance achieved amongst AS, DT and AS&DT data. Even after reducing the number of attributes from as much as 340 to as low as 25, on average, the performance in terms of MRE only reduces by 1.67%. This is a very reasonable trade-off and in the majority of the work that follows only IA data will be used.

7.4.3 Basic CPR tests

AS&DT dataset

All of the tests conducted until now were done using the same PR for training and testing. This assumes that the PR remains constant during training and testing phases, which may not be necessarily correct. A cross-check was performed between PRs by training machine with one PR and testing with another, hence the name CPR. This was done for all combinations of PRs mentioned below. Following are the test configurations:

- PR [%] : 5, 10, 15, 25, 50, 75, 100
- Data : AS&DT
- Step size [s] : 60
- Window size [#] : 10
- Window size [s] : 600
- VILs [#] : 17
- Attributes [#] : 170
- Training seed(s) : 100, 200
- Testing seed(s) : 300

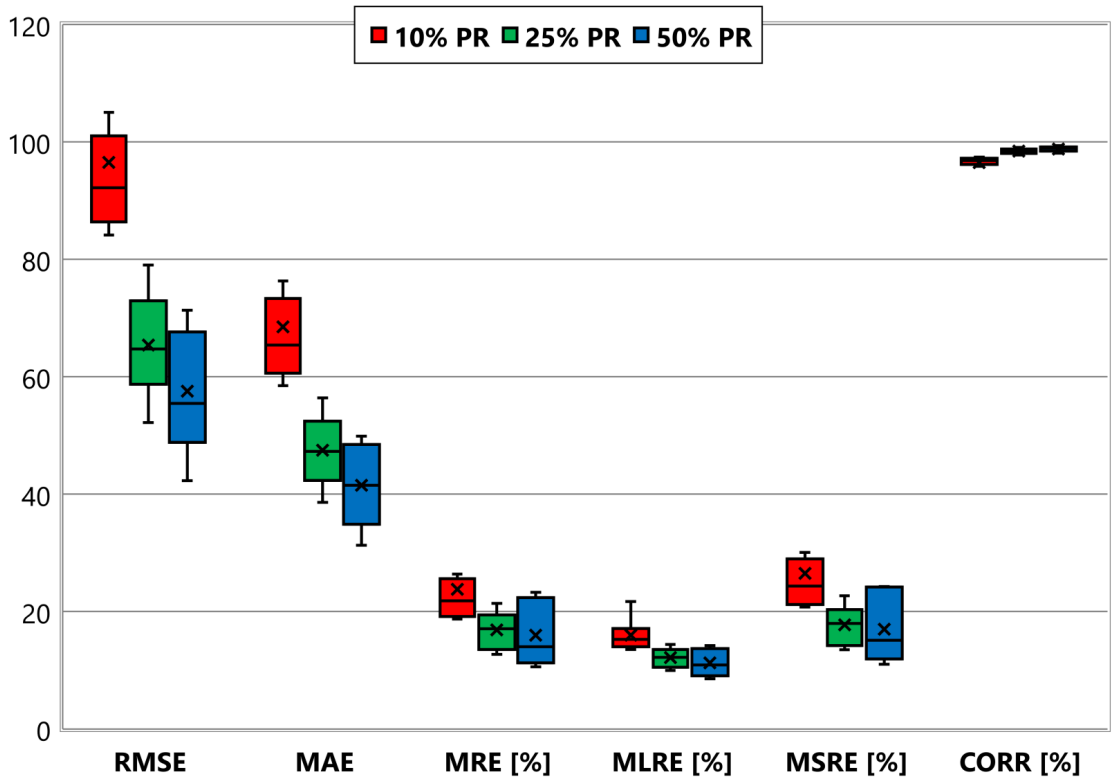


Figure 7.11: IA dataset CRS test results.

Figure 7.12 shows results grouped by training PR in box plot representation. Figure 7.13 shows results grouped by testing PR in box plot representation. Performance parameters followed by * represent parameters which are calculated only for the more important part of the day. After analysing the daily traffic flow, it was concluded that the important part of the day is the part where a prediction of traffic might be useful in order to act upon it (by some means of traffic congestion control such as traffic light plan change). In my analysis, for this particular intersection, that part is from 06:30 until 20:30.

Results show that the performance of the prediction machine increases as the difference between the training and testing PR reduces. For each individual training PR, the best performance is achieved when the testing PR is the same as the training PR. For example, if a machine is trained with 5% PR and the testing PR is 100%, it does not achieve better performance than if the testing PR is 5%.

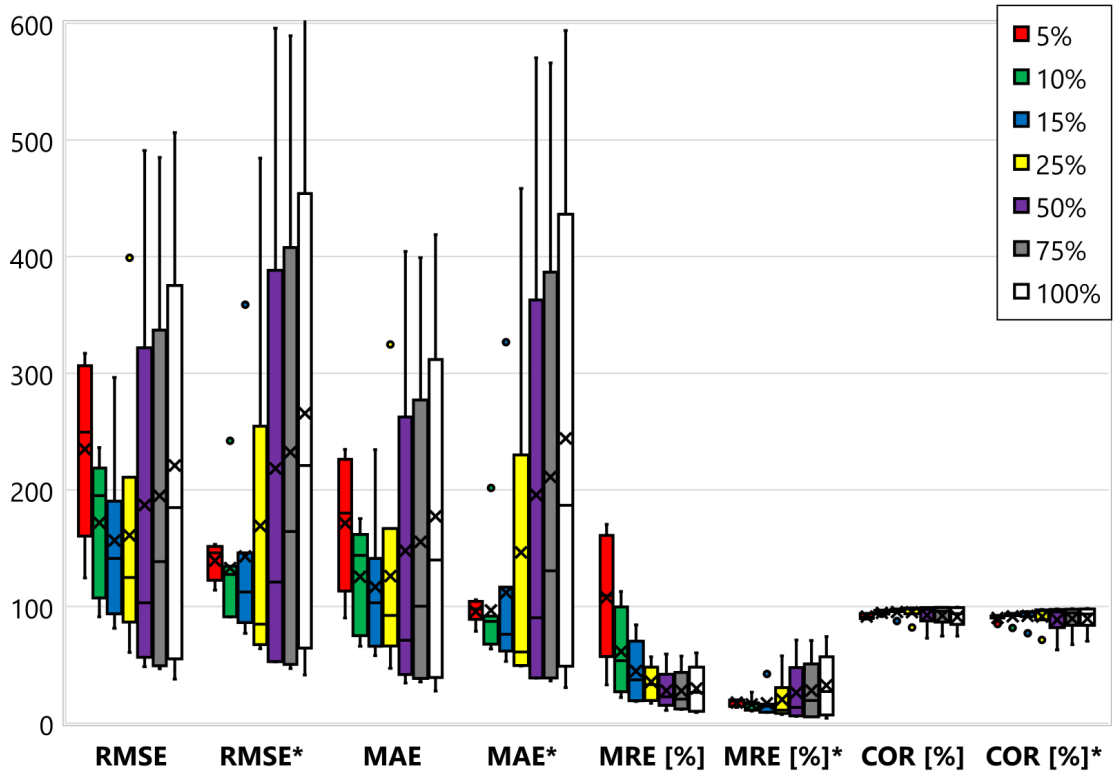


Figure 7.12: AS&DT dataset CPR test results box plot grouped by training PR.

7.4.4 CRS tests with hour of the day

Some tests were performed to understand how will adding 'hour of the day' to the dataset affect the prediction performance. The 'hour of the day' provides some context to the machine about which part of the day does this data belong to. This does not, however, specify exactly what is the timestamp of data.

AS&DT&H dataset

First, 'hour of the day' (H) was added to AS&DT data to create AS&DT&H dataset. Following are the test configurations:

- PR [%] : 10, 25, 50
- Data : AS&DT&H
- Step size [s] : 30, 60, 120
- Window size [#] : 5, 10, 20
- Window size [s] : 150, 300, 600, 1200, 2400
- VILs [#] : 17
- Attributes [#] : 90, 180, 360
- Training seed(s) : 100, 200
- Testing seed(s) : 300

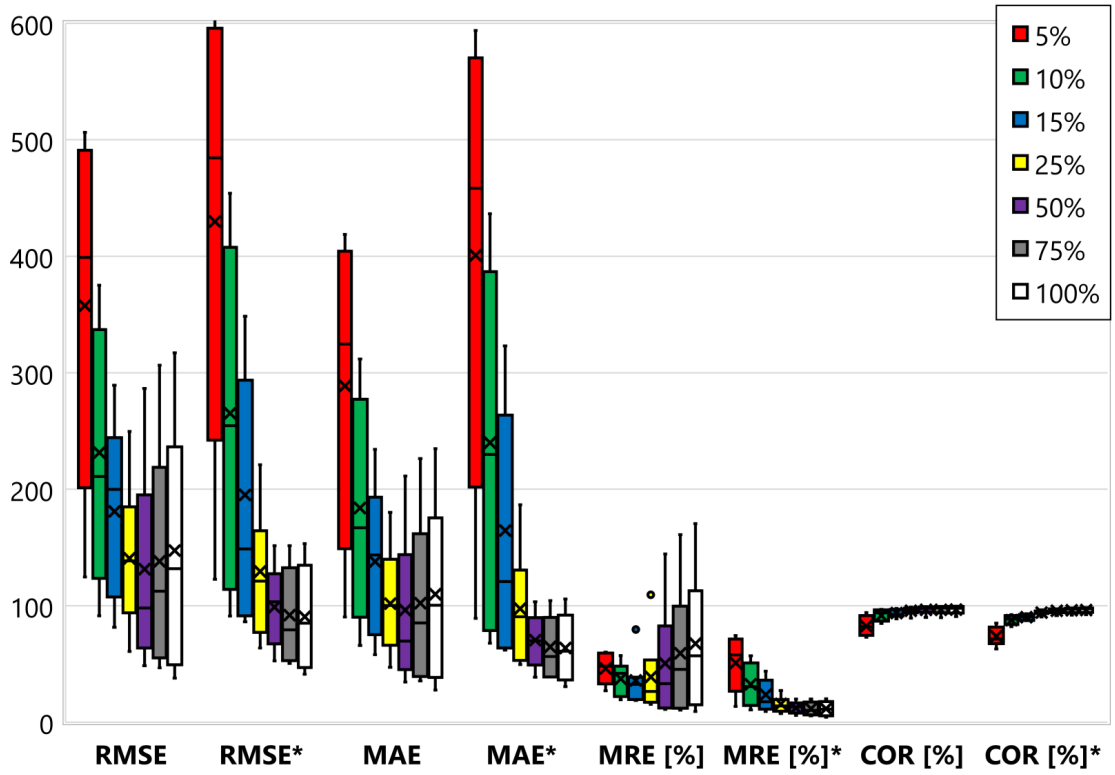


Figure 7.13: AS&DT dataset CPR test results box plot grouped by testing PR.

Figure A.4 shows results grouped by PR in box plot representation. Results show a significant improvement in terms of all performance parameters. When compared to results presented in Section 7.4.1, overall, on average an improvement of 48% in RMSE was seen. The problem, however, is that the machine relies too much on the hour attribute and mostly ignore VIL attributes. The result is that the predicted output of the machine does not reflect the actual situation of traffic collected by VIL.

IA&H dataset

To test the effect of 'hour of the day' (H) to IA dataset, H was added to IA data to create IA&H dataset. Following are the test configurations:

- PR [%] : 10, 25, 50
- Data : IA&H
- Step size [s] : 30, 60, 120
- Window size [#] : 5, 10, 20
- Window size [s] : 150, 300, 600, 1200, 2400
- VILs [#] : 5

- Attributes [#] : 30, 60, 120
- Training seed(s) : 100, 200
- Testing seed(s) : 300

Figure A.5 shows results grouped by PR in box plot representation. The analysis of these results reconfirm what is concluded earlier in Section 7.4.4. When compared to results presented in Section 7.4.2, overall, on average an improvement of 49% in RMSE was noticed. Since the machine rely too much on 'hour of the day' and less on the situation of traffic reflected by VIL data, it was concluded that it is not advantageous to pursue this track.

7.4.5 Classification tests (CRS)

All prediction schemes until now are regression-based predictions in which the intensity of traffic itself was predicted. Here, a classification-based analysis is conducted instead to classify the current intensity of traffic into a number of classes.

IA_3C dataset

The first classification dataset created classifies traffic intensity into 3 different classes *i.e.* low, medium, high. The maximum traffic intensity in the AS&DT dataset was 1100 which was equally distributed among the 3 classes. Following are the test configurations:

- PR [%] : 10, 25, 50
- Data : IA_3C
- Classes [#] : 3
  - low $\stackrel{\text{def}}{=}$ $[0, 367)$ veh/hour
  - medium $\stackrel{\text{def}}{=}$ $[367, 733)$ veh/hour
  - high $\stackrel{\text{def}}{=}$ $[733, \infty)$ veh/hour
- Step size [s] : 30, 60, 120, 300
- Window size [#] : 5, 10, 20
- Window size [s] : 150, 300, 600, 1200, 1500, 2400, 3000, 6000
- VILs [#] : 5
- Attributes [#] : 25, 50, 100
- Training seed(s) : 100, 200
- Testing seed(s) : 300

Results show an average accuracy of 85.25% with 10% of PR, 88.6% with 25% PR and 91.03% with 50% PR. In general, accuracy increases as PR increases, as step size decreases and as window size increases.

IA_2CH dataset

Traffic intensity was split into 2 different classes *i.e.* 0 and 1 with hysteresis and a binning window to create the dataset referred to as IA_2CH_BW. The double

threshold-based hysteresis is applied to the average traffic flow calculated using the binning window. The traffic flow is classified using the following equation:

$$c_n = \begin{cases} 0, & \text{if } c_{n-1} = 0 \ \& \ f_n < 300 \\ 0, & \text{if } c_{n-1} = 1 \ \& \ f_n < 200 \\ 1, & \text{if } c_{n-1} = 0 \ \& \ f_n \geq 300 \\ 1, & \text{if } c_{n-1} = 1 \ \& \ f_n \geq 200 \end{cases}$$

where c_n is the current class, c_{n-1} is the previous class and f_n is the average traffic flow of the current binning window. Following are the test configurations:

- PR [%] : 10
- Data : IA_2CH_BW
- Classes [#] : 2
- Step size [s] : 30, 60, 120, 300
- Window size [#] : 5, 10, 20
- Window size [s] : 150, 300, 600, 1200, 1500, 2400, 3000, 6000
- Hysteresis binning window [s] : 600, 1200
- VILs [#] : 5
- Attributes [#] : 25, 50, 100
- Training seed(s) : 100, 200
- Testing seed(s) : 300

Figure 7.14 shows results grouped by the step size in box plot representation. The accuracy is quite impressive and stays higher than 90% in all cases. Results show the accuracy of classification increases as the step size increases. Additionally, the machine is better able to recognise class 1 compared to class 0 (TPR > TNR). Figure 7.15 shows results grouped by window size in box plot representation. Results show that accuracy improves as window size increases. Figure A.6 shows results grouped by binning window size in box plot representation. Results show better accuracy at smaller binning window size. So the ideal configuration for classification should be: step size of 120s, windows size of 20 and binning window size of 1200s.

7.4.6 Real days dataset

Until now, all of the tests were conducted on a hypothetical weekday made using average traffic flow during weekdays during a certain time period. In order to check the performance of the prediction mechanism with traffic situation of real days and make the simulation more realistic, a very large dataset was created consisting of 56 real days. This dataset consists of all types of days identified earlier. Between the months of October-2017 and April-2018, the first available (in my archive) working weekday from each month was selected. Similarly, the first

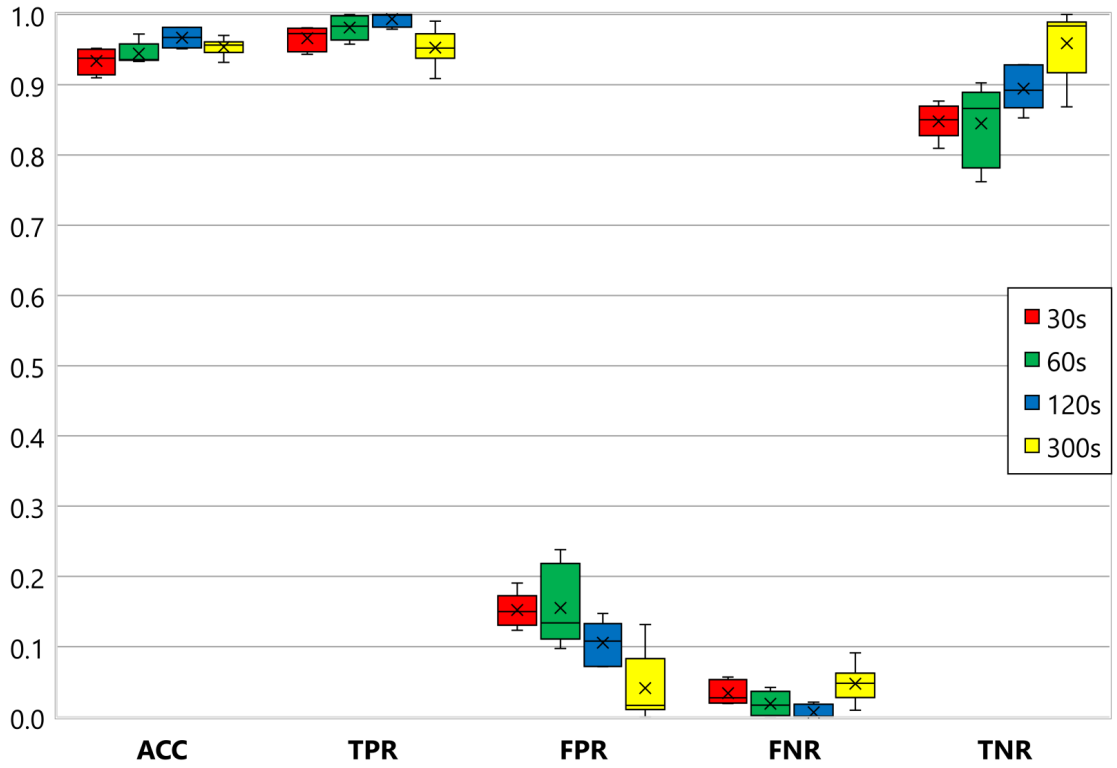


Figure 7.14: IA_2CH dataset CRS test results box plot grouped by step size.

available weekend days and seven national and local holidays that occurred during this period were chosen. Table 7.4 shows all details relating to these selected 56 days. These days were chosen and simulated using unique random seeds *i.e.* all of these simulated days have a different random seed. In the sections that follow, these days will be referred to by their code names mentioned in the Table. The entire simulation dataset (174 files; 13.5 GB in total) is available online with open access [43].

Table 7.4: Details of selected days for real days dataset.

Date	Day	Month	Category	Code Name	Random Seed
2017-10-01	Sunday	October	Sunday	Su1	20000
2017-10-02	Monday	October	Monday	M1	100
2017-10-07	Saturday	October	Saturday	Sa1	13000
2017-10-10	Tuesday	October	Tuesday	Tu1	27000
2017-10-11	Wednesday	October	Wednesday	W1	34000
2017-10-12	Thursday	October	Thursday	Th1	41000

Date	Day	Month	Category	Code Name	Random Seed
2017-10-13	Friday	October	Friday	F1	800
2017-11-01	Wednesday	November	Holiday	H1	6000
2017-11-02	Thursday	November	Thursday	Th2	42000
2017-11-03	Friday	November	Friday	F2	900
2017-11-04	Saturday	November	Saturday	Sa2	14000
2017-11-05	Sunday	November	Sunday	Su2	21000
2017-11-06	Monday	November	Monday	M2	200
2017-11-07	Tuesday	November	Tuesday	Tu2	28000
2017-11-08	Wednesday	November	Wednesday	W2	35000
2017-12-01	Friday	December	Friday	F3	1000
2017-12-02	Saturday	December	Saturday	Sa3	15000
2017-12-03	Sunday	December	Sunday	Su3	22000
2017-12-04	Monday	December	Monday	M3	300
2017-12-05	Tuesday	December	Tuesday	Tu3	29000
2017-12-06	Wednesday	December	Wednesday	W3	36000
2017-12-07	Thursday	December	Thursday	Th3	43000
2017-12-08	Friday	December	Holiday	H2	7000
2017-12-25	Monday	December	Holiday	H3	8000
2017-12-26	Tuesday	December	Holiday	H4	9000
2018-01-04	Thursday	January	Thursday	Th4	44000
2018-01-05	Friday	January	Friday	F4	2000
2018-01-13	Saturday	January	Saturday	Sa4	16000
2018-01-14	Sunday	January	Sunday	Su4	23000
2018-01-15	Monday	January	Monday	M4	400
2018-01-16	Tuesday	January	Tuesday	Tu4	30000
2018-01-17	Wednesday	January	Wednesday	W4	37000
2018-02-02	Friday	February	Friday	F5	3000
2018-02-03	Saturday	February	Saturday	Sa5	17000
2018-02-11	Sunday	February	Sunday	Su5	24000
2018-02-12	Monday	February	Monday	M5	500
2018-02-13	Tuesday	February	Tuesday	Tu5	31000
2018-02-14	Wednesday	February	Wednesday	W5	38000
2018-02-15	Thursday	February	Thursday	Th5	45000
2018-03-01	Thursday	March	Thursday	Th6	46000
2018-03-02	Friday	March	Friday	F6	4000
2018-03-03	Saturday	March	Saturday	Sa6	18000
2018-03-04	Sunday	March	Sunday	Su6	25000
2018-03-05	Monday	March	Monday	M6	600

Date	Day	Month	Category	Code Name	Random Seed
2018-03-06	Tuesday	March	Tuesday	Tu6	32000
2018-03-07	Wednesday	March	Wednesday	W6	39000
2018-03-30	Friday	March	Holiday	H5	10000
2018-04-01	Sunday	April	Holiday	H6	11000
2018-04-02	Monday	April	Holiday	H7	12000
2018-04-03	Tuesday	April	Tuesday	Tu7	33000
2018-04-04	Wednesday	April	Wednesday	W7	40000
2018-04-05	Thursday	April	Thursday	Th7	47000
2018-04-06	Friday	April	Friday	F7	5000
2018-04-07	Saturday	April	Saturday	Sa7	19000
2018-04-08	Sunday	April	Sunday	Su7	26000
2018-04-09	Monday	April	Monday	M7	700

M_AS&DT dataset (CRS)

The first real days tests conducted were between all Mondays in the database. 2 consecutive Mondays from the selected days were selected to train the machine, then this machine was used to predict traffic intensity on all remaining Mondays. This meant doing 5 tests per configuration scheme. The test configurations are below:

- PR [%] : 10
- Data : AS&DT
- Step size [s] : 60
- Window size [#] : 5, 10, 20
- Window size [s] : 300, 600, 1200
- VILs [#] : 17
- Attributes [#] : 85, 170, 340
- Training seed(s) : Multiple
- Testing seed(s) : Multiple

Figure A.7 shows results from 90 tests grouped by window size in box plot representation. Results look very promising from all performance matrices perspective. If results obtained from training and testing multiple different Mondays are comparable with results obtained with training and testing a hypothetical day, it would imply that the machine is very well capable of predicting real traffic flow. In other words, results presented here should be comparable with results presented in Section 7.4.1. This stands true for all parameters including RMSE, MAE, MRE, and COR. For some parameters, a very negligible performance degradation was noticed.

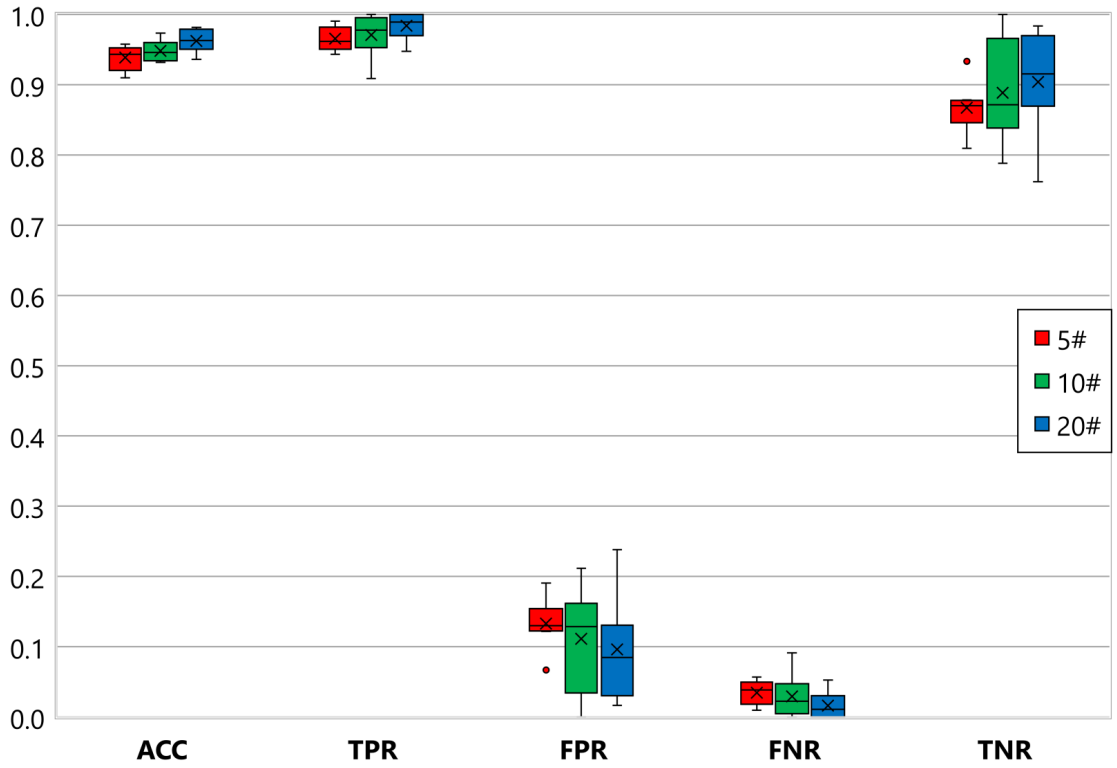


Figure 7.15: IA_2CH dataset CRS test results box plot grouped by window size.

IA 5-categories dataset (CRS)

To better explore how the training of machine can be improved a set of trails were conducted by increasing the size of the training dataset. The idea is that if the machine is able to train using more data and diverse data (different types of days), the performance may improve. To test this theory, a dataset was created consisting of all selected Mondays, Fridays, Saturdays, Sundays and Holidays. This came up to 35 real days of data in total. Experiments were done with multiple combinations to better understand which combination of days improve prediction performance. The machine was trained using 4 different combinations of training dataset including:

1. A Monday, Friday, Saturday, Sunday and a Holiday (MFSaSuH)
2. Two Mondays (MM)
3. Two Mondays and two Saturdays (MMSaSa)
4. Four Mondays (MMMM)

For each training dataset combination three separate tests were performed with different selected days and each test consisted of predicting all five types of days. So, in summary, this totalled to 60 tests. The test configurations are below:

- PR [%] : 10
- Data : IA
- Step size [s] : 60
- Window size [#] : 10
- Window size [s] : 600
- VILs [#] : 5
- Attributes [#] : 50
- Training seed(s) : Multiple
- Testing seed(s) : Multiple

Figure A.8 shows results grouped by training dataset type in box plot representation. Figure A.9 shows results grouped by testing dataset type in box plot representation. A machine trained with different type of days is expected to outperform a machine trained only with a specific type of day. This is confirmed by analysing results from MFSaSuH and MMSaSa against MM and MMMM. Considering all parameters, MFSaSuH dataset training performs best. However, it is noteworthy and interesting to see that MMSaSa shows a very minimal performance difference compared to the best.

IA 8-categories dataset (CRS)

One of the conclusions from Section 7.4.6 is that a machine trained with different types of days is, in general, better able to predict traffic. For this next test, a dataset consisting of all 8 categories of selected days including Mondays, Tuesdays, Wednesdays, Thursdays, Fridays, Saturdays, Sundays and holidays was created. This came up to 56 real days of data in total. Experiments were performed using 12 different combinationsto better understand which combination of days improve prediction performance including:

1. Two Mondays (MM)
2. A Monday, Saturday and a holiday (MSaH)
3. Four Mondays (MMMM)
4. Two Mondays and two Saturdays (MMSaSa)
5. Two Mondays, two Saturdays, two Sundays and two holidays (MMSaSaSuSuHH)
6. A Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday and a holiday belonging to the same month (MTuWThFSaSuH)

7. A Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday and a holiday belonging to the different months (MTuWThFSaSuH*)
8. A Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, and Sunday belonging to the same month (MTuWThFSaSu)
9. A Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, and Sunday belonging to the different months (MTuWThFSaSu*)
10. Two Mondays, two Tuesdays, two Wednesdays, two Thursdays and two Fridays (MMTuTuWWThThFF)
11. A Monday, Tuesday, Wednesday, Thursday, Friday and Saturday belonging to the same month (MTuWThFSa)
12. A Monday, Tuesday, Wednesday, Thursday, Friday and Saturday belonging to the different months (MTuWThFSa*)

For each training dataset combination three separate tests were conducted with different selected days and each test consisted of predicting all eight types of days. So, in summary, this totalled to 288 tests. The test configurations are below:

- PR [%] : 10
- Data : IA
- Step size [s] : 60
- Window size [#] : 10
- Window size [s] : 600
- VILs [#] : 5
- Attributes [#] : 50
- Training seed(s) : Multiple
- Testing seed(s) : Multiple

Figure 7.16 shows results grouped by training dataset type in box plot representation. Figure 7.17 shows results grouped by testing dataset type in box plot representation. Considering all performance parameters, the best overall performance is achieved by the dataset containing an entire week of traffic with a holiday (MTuWThFSaSuH*). In fact, in general, datasets containing a lower number of days perform worse than datasets including a higher number of days. One factor for this could be the fact that the number of training samples in a dataset increases as the number of days in the dataset increases. Analysing the Figure with results grouped by testing dataset type (the type of day), the machine is able to predict traffic intensities on all weekdays with similar accuracy. However, on weekends and holidays, the performance suffers a little bit. In a realistic scenario, this should not cause a serious problem since the rate of occurrence of a weekend is only 28%. Similarly, the number of holidays in a year is quite limited, compared to the number of weekdays.

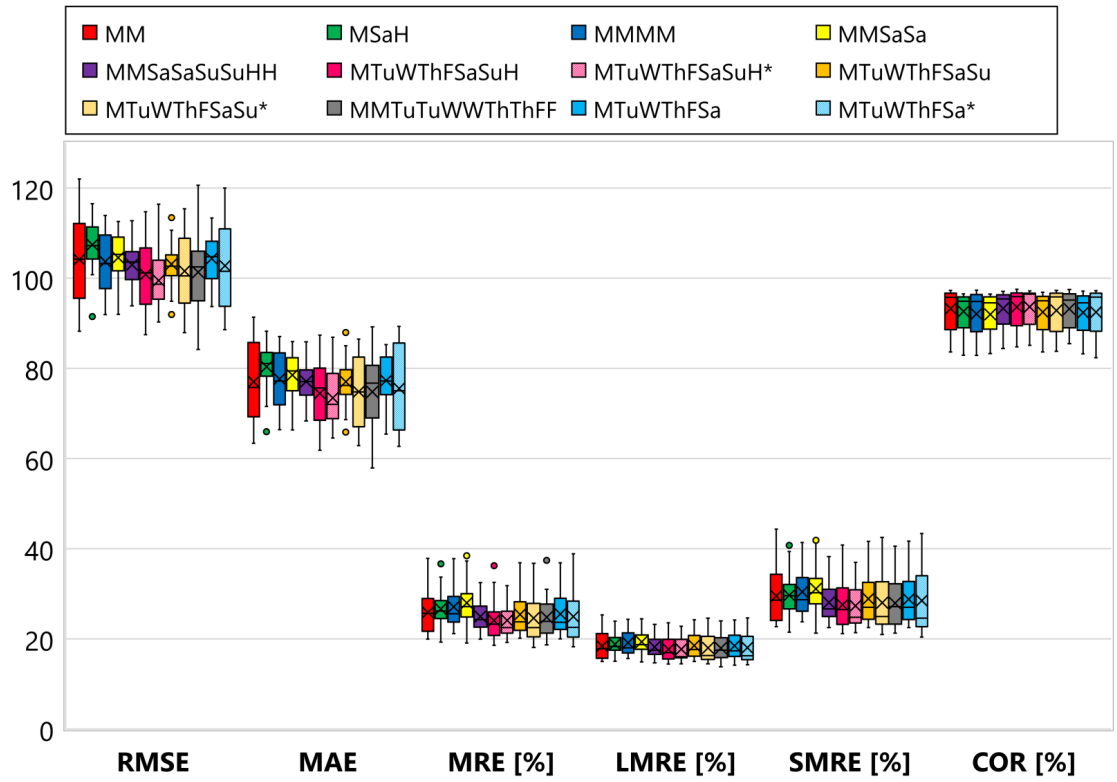


Figure 7.16: IA 8-categories dataset CRS test results box plot grouped by training PR.

IA 8-categories dataset (CRS)

Similar to Section 7.4.3, a CPR test was done with this dataset. This elaborates on how the predictions will be affected by a change in the difference between the training and testing PRs. For testing PRs somewhat realistic PRs were selected including 5%, 10% and 15%. For training the machine the single PRs already selected for testing were chosen along with their combinations such as [10%, 5%] and [15%, 10%, 5%]. Sets of PR written between squared brackets signify one combined dataset. As far as the combination of the type of days is concerned, the MTuWThFSaSu combination was chosen for three different months. For each combination of training and testing PR, two different training day types and eight testing days each were singled out. So in total, each PR combination was tested 16 times resulting in a total of 192 predictions. Please note that the training and testing days always belonged to different months. Other test parameters were as follows:

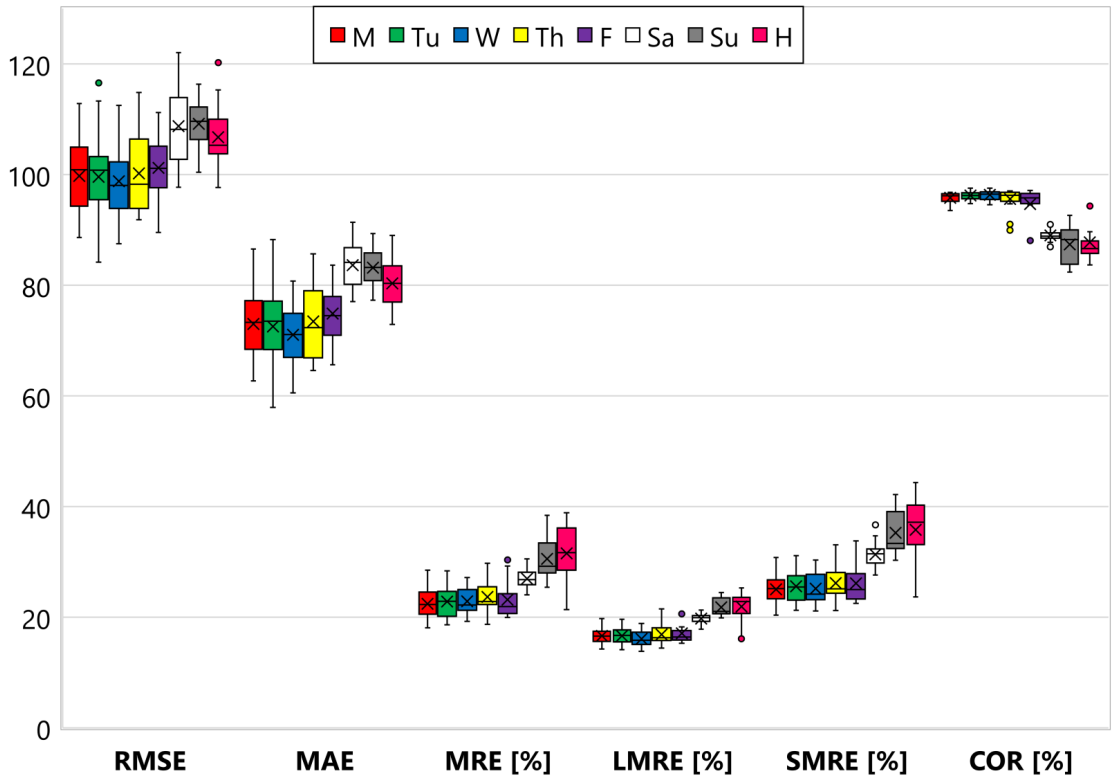


Figure 7.17: IA 8-categories dataset CRS test results box plot grouped by testing PR.

- Data : IA
- Step size [s] : 60
- Window size [#] : 10
- VILs [#] : 5
- Attributes [#] : 50
- Training seed(s) : Multiple
- Testing seed(s) : Multiple
- Training days : MTuWThFSaSu (2, 4 & 6)

Figure A.10 shows results grouped by training PR in box plot representation. Figure A.11 shows results grouped by training and testing PR combination in box plot representation. The first takeaway result from the test is that training sets composed of multiple training PRs outperform training sets composed of a single PR. In my tests, training PRs [10%, 5%] and [15%, 10%, 5%] outperformed all others and were similar to each other. Secondly, as previously experienced in Section 7.4.3, the performance of prediction is indirectly proportional to the difference between the training and testing PRs. Best results (in case of single training

PRs) are achieved when the training and testing PRs are the same. As the difference between the training and testing PRs starts increasing, the accuracy starts decreasing.

7.4.7 DPR datasets

From a practical point of view, the proposed traffic prediction system would consist of two stages *i.e.* training stage and testing stage. The training stage would be temporary and may last just a few weeks (two to three). During the training stage, I propose to use some temporary traffic sensing instrument such as pneumatic tubes, video detection or radar detection. The temporary sensing instrument will sense more or less 100% of the passing traffic. Additionally, data coming from VIL system will be collected during this stage. After the training stage is over, the temporary sensing system can be removed.

The dataset created during the training stage can be considered a 100% PR dataset. But depending on the PR of the VIL system, the PR during the testing stage will be much lower. In theory, it is possible to derive lower PR datasets from higher PR datasets. This is true since statistically, all lower PR datasets will be subsets of the 100% PR dataset. To verify this hypothesis, multiple lower PR datasets were derived from the 100% PR dataset called Derived Penetration Rate (DPR) dataset.

IA 8-categories DPR dataset (CRS)

A very detailed CPR test was conducted using all DPR. Eight PRs were deribed from 100% PR dataset (*i.e.* 20%, 15%, 12%, 10%, 8%, 5%, 2.5%, 1%). To do this, x% of vehicles were randomly selected from 100% of the vehicles using the same seed as the one used for the simulation of that particular day. Tests were performed by using all DPRs as testing PRs. For training PRs single DPRs were used as well as combinations of multiple DPRs (*i.e.* [40%, 15%, 10%, 5%], [20%, 15%, 10%, 5%], [20%, 15%, 10%], [15%, 10%, 5%], [10%, 5%], 15%, 10%, 5%, 2.5%, and 1%). Sets of PR written between squared brackets signify one combined dataset. Each combination of training and testing PR was tested with two different sets of training days. Each set of training day was used to predict all eight types of days. Please note that, as before, the testing and the training days were never overlapping in time (for example belong to the same month). This summed up to 1280 individual predictions. Other test configurations are as follows:

- Data : IA
- Step size [s] : 60
- Window size [#] : 10
- VILs [#] : 5
- Attributes [#] : 50
- Training seed(s) : Multiple

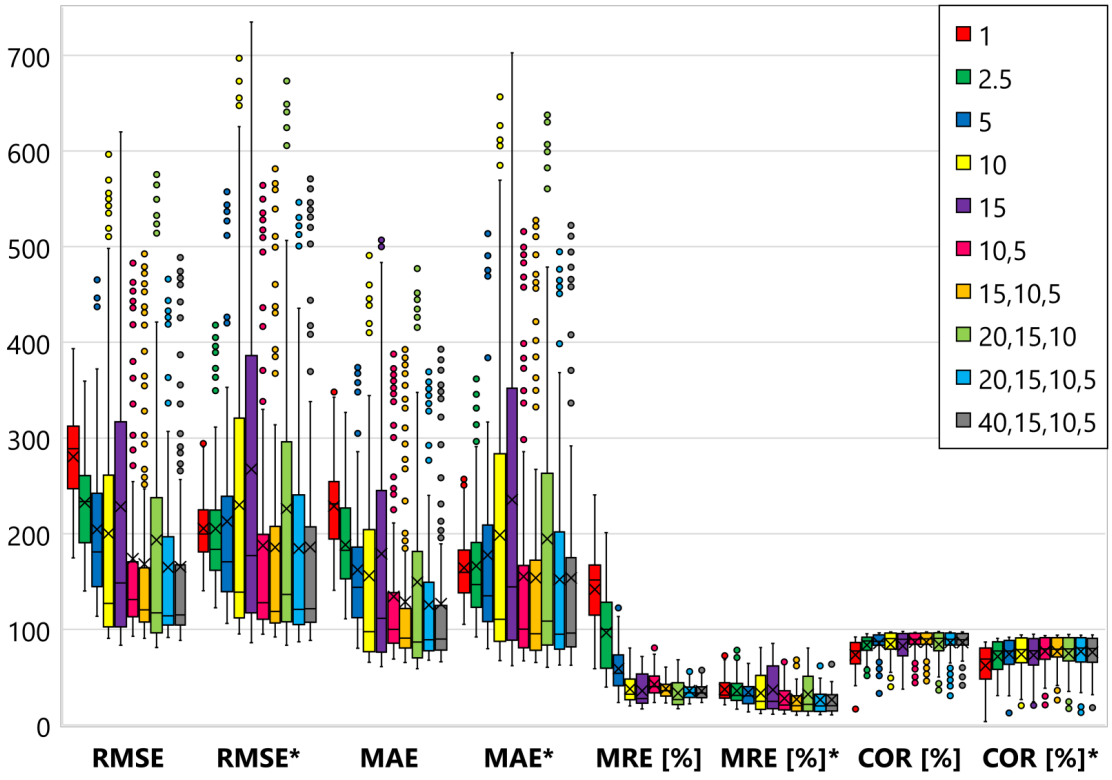


Figure 7.18: IA 8-categories DPR dataset CPR test results box plot grouped by training PR.

- Testing seed(s) : Multiple
- Training days : MTuWThFSaSu (2, 4 & 6)

Figure 7.18 shows results grouped by training PR in box plot representation. Figure 7.19 shows results grouped by testing PR in box plot representation. Results show that, in general, single PR training sets perform worst than combined PR training sets. In fact, these results are very much comparable with results presented in Section 7.4.6. Previous findings can also be confirmed here, such as prediction performance improves as the PR increases. As far as testing PR is concerned, the same remains true until the PR increases to about 10%. After that, the performance remains stable (doesn't get worse) but contains more outliers. This can be because of the combination of training and testing PRs. Figure A.12, 7.20, and A.13 further investigates what would be the best training PR(s) dataset to predict somewhat realistic testing PRs (1%, 2.5% and 5%).

Figure A.12, 7.20, and A.13 show results grouped by training PR in box plot representation only for tests conducted with testing PR of 1%, 2.5%, and 5% respectively. In all three cases, best results are achieved when the training PR is the

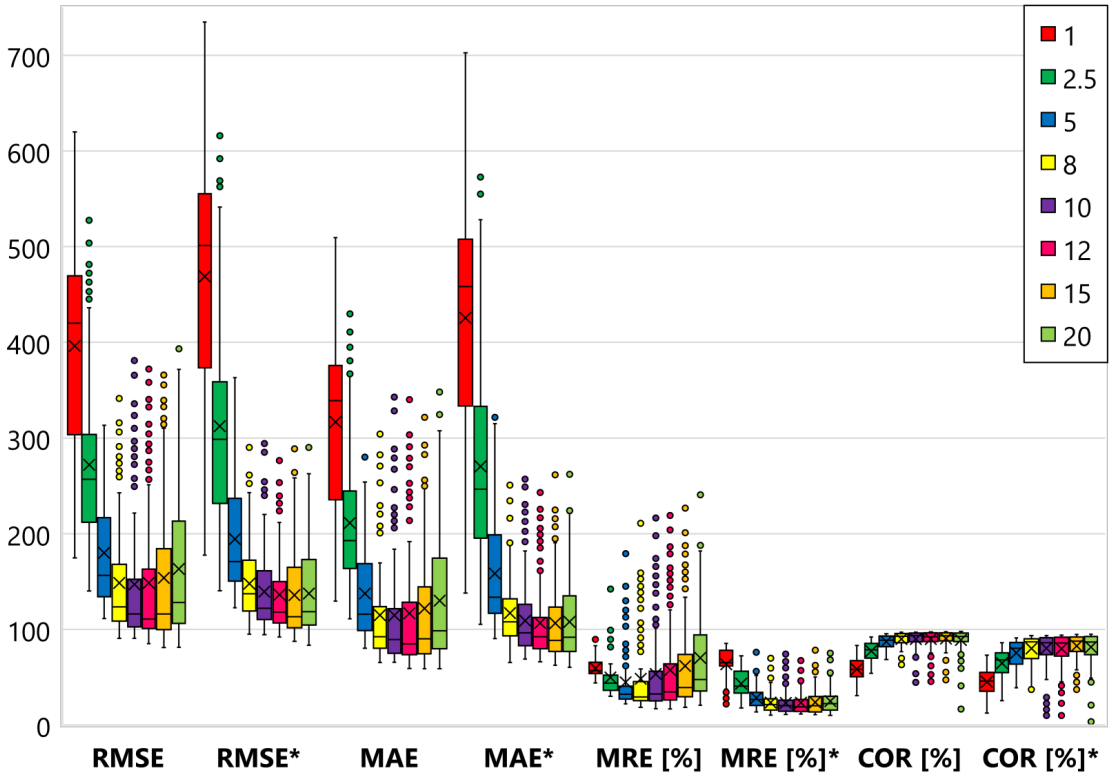


Figure 7.19: IA 8-categories DPR dataset CPR test results box plot grouped by testing PR.

same as the testing PR. Moreover, all combined PR training sets which contain lower PRs (*e.g.* [40%, 15%, 10%, 5%], [20%, 15%, 10%, 5%], [15%, 10%, 5%], [10%, 5%]) perform better than others (*e.g.* [20%, 15%, 10%]). This again concludes that the machine is better able to predict traffic when the difference between training and testing PRs is low.

7.4.8 CPR tests with VPR (DPR) dataset

All tests conducted until now consider a constant PR throughout the day. This is an assumption taken to simplify the exhaustive training and testing procedure. However, in reality, the actual PR can vary throughout the day. To understand how the machine will behave in such a situation, extensive CPR tests were conducted with a new dataset created from DPR having VPR. Following are the testing VPR datasets that were made to simulate a semi-realistic day:

1. 1-2.5-5-7.5-10
    - Low PR: random between 1%, 2.5% and 5%

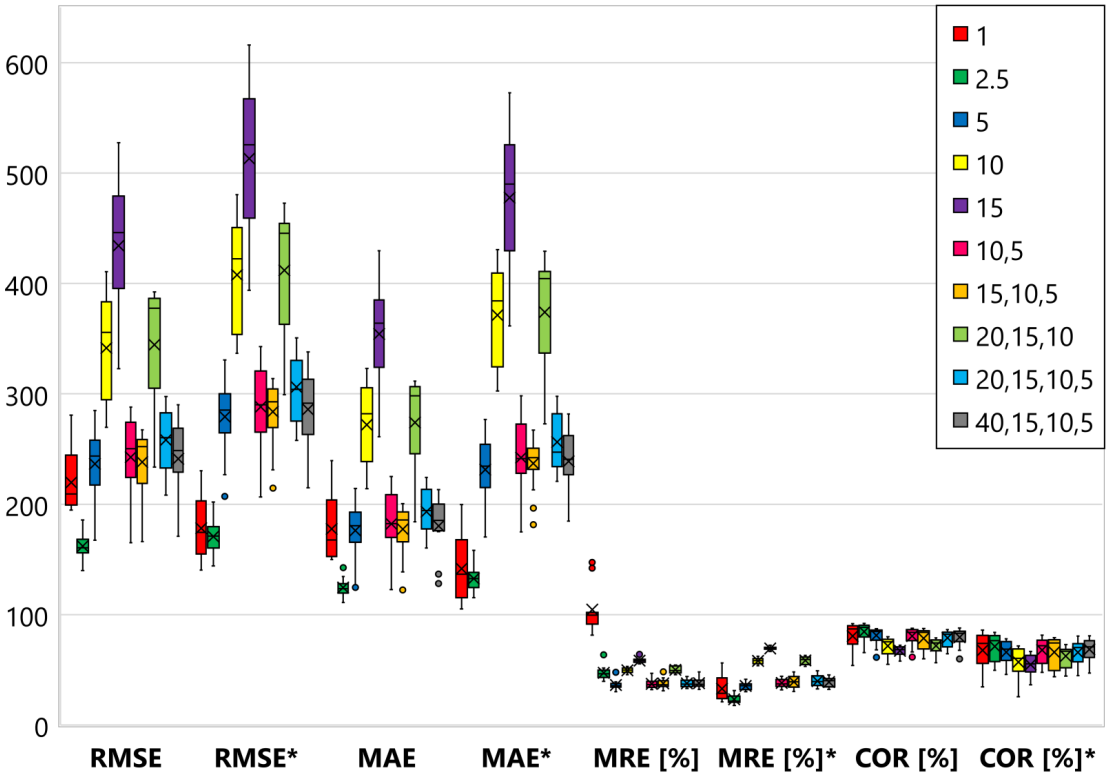


Figure 7.20: IA 8-categories DPR dataset CPR test results box plot testing only with 2.5% PR.

- Medium PR: random between 2.5%, 5% and 7.5%
- High PR: random between 5%, 7.5% and 10%

2. 1-5-10
   - Low PR: 1%
   - Medium PR: 5%
   - High PR: 10%
3. 5-10-15
   - Low PR: 5%
   - Medium PR: 10%
   - High PR: 15%
4. 5-8-10-12-15

- Low PR: random between 5%, 8% and 10%
- Medium PR: random between 8%, 10% and 12%
- High PR: random between 10%, 12% and 15%

5. 8-10-12
    - Low PR: 8%
    - Medium PR: 10%
    - High PR: 12%

In all datasets, PR changes every 60 mins. Low, medium and high hourly average traffic intensities thresholds used here are the following:

- low $\stackrel{\text{def}}{=}$ $[0, 300)$ veh/hour
- medium $\stackrel{\text{def}}{=}$ $[300, 600)$ veh/hour
- high $\stackrel{\text{def}}{=}$ $[600, \infty)$ veh/hour

Figure 7.21 shows an example of how the VPR scheme works. To predict the traffic flow of these datasets, machines were trained with the following training PRs:

1. 1,2.5,5,7.5,10
    - Combined PR [1%, 2.5%, 5%, 7.5% and 10%]
2. 1-2.5-5-7.5-10
    - Low PR: random between 1%, 2.5% and 5%
    - Medium PR: random between 2.5%, 5% and 7.5%
    - High PR: random between 5%, 7.5% and 10%
3. 5,10,15
    - Combined PR [5%, 10% and 15%]
4. 5,7.5,10
    - Combined PR [5%, 7.5% and 10%]
5. 5,8,10,12,15
    - Combined PR [5%, 8%, 10%, 12% and 15%]
6. 5-10-15

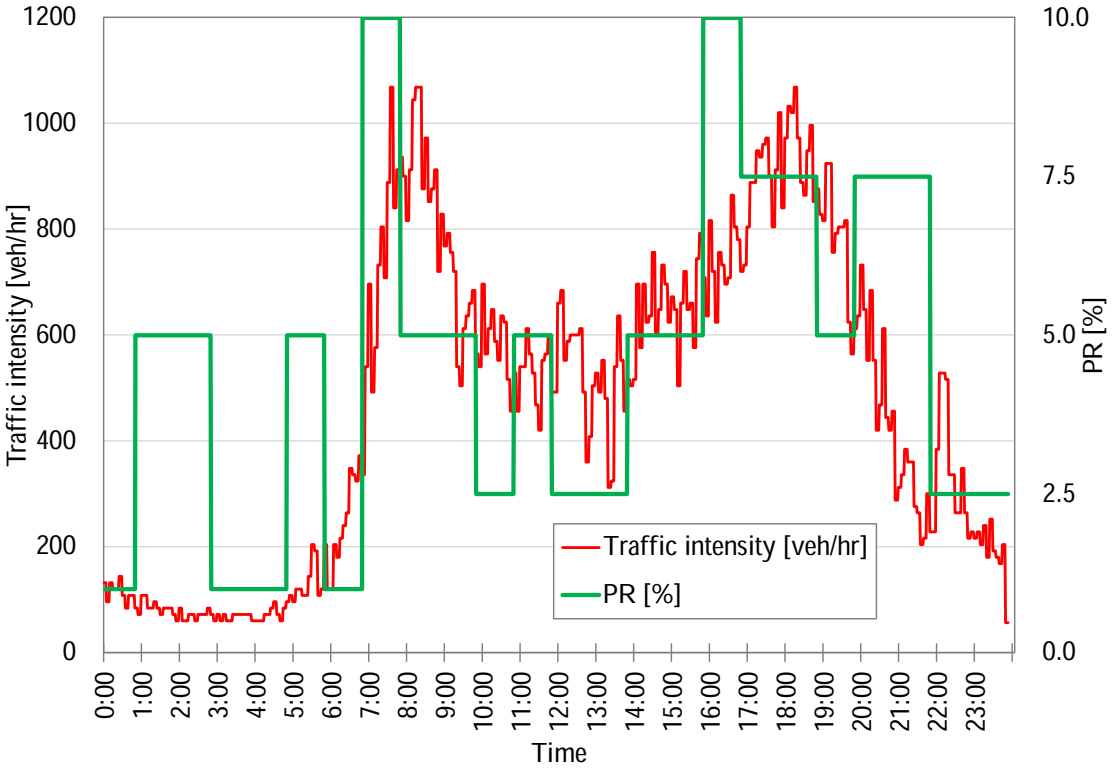


Figure 7.21: IA 8-categories derived and variable PR dataset scheme.

- Low PR: 5%
- Medium PR: 10%
- High PR: 15%

7. 5-10-15,8-10-12
   - Combined PR [Item 6 and Item 10]
8. 5-8-10-12-15
   - Low PR: random between 5%, 8% and 10%
   - Medium PR: random between 8%, 10% and 12%
   - High PR: random between 10%, 12% and 15%
9. 8,10,12
   - Combined PR [8%, 10% and 12%]
10. 8-10-12

- Low PR: 8%
- Medium PR: 10%
- High PR: 12%

Each training dataset was used to predict all eight types of days. Please note that, as before, the testing and the training days were never overlapping in time (for example belong to the same month). This summed up to 800 individual predictions. Other test configurations are as follows:

- Data : IA
- Step size [s] : 60
- Window size [#] : 10
- VILs [#] : 5
- Attributes [#] : 50
- Training seed(s) : Multiple
- Testing seed(s) : Multiple
- Training days : MTuWThFSaSu (2, 4 & 6)

Figure 7.22 shows results grouped by training PR in box plot representation. Figure 7.23 shows results grouped by testing PR in box plot representation. Results show that, in general, training sets with VPR perform better than their combined PR training sets equivalents. Secondly, these results are very much comparable with results presented in Section 7.4.6. As far as testing PR is concerned, the performance improves (as expected) when average PR increases. Figure A.14 further investigates what would be the best training PR(s) dataset to predict a somewhat realistic day represented by 1-2.5-5-7.5-10 testing PRs (Item 1).

Figure A.14 shows results grouped by training PR in box plot representation only for tests conducted with testing PR of 1-2.5-5-7.5-10 (Item 1). The best performance is achieved when the testing PR is the same as training PR. In general, training PR datasets with a higher average PR perform worst. This can be explained again as before due to the high difference between training and testing PRs.

7.4.9 Training dataset size

The difference in training dataset sizes might lead to unfair comparison. For example the training dataset size of combined PR dataset (Item 1) is roughly 5 times higher than its VPR equivalent (Item 2). This is simply true because this combined PR dataset contains 5 versions with different PRs of one week (equal to 35 days). While in the case of VPR dataset, it only contains one VPR version of one week (equal to 7 days). To eliminate this, 5 different versions of VPR datasets were created and simulated with different random seeds. This ensures that the training dataset sizes are the same between combined PR dataset and its VPR equivalent.

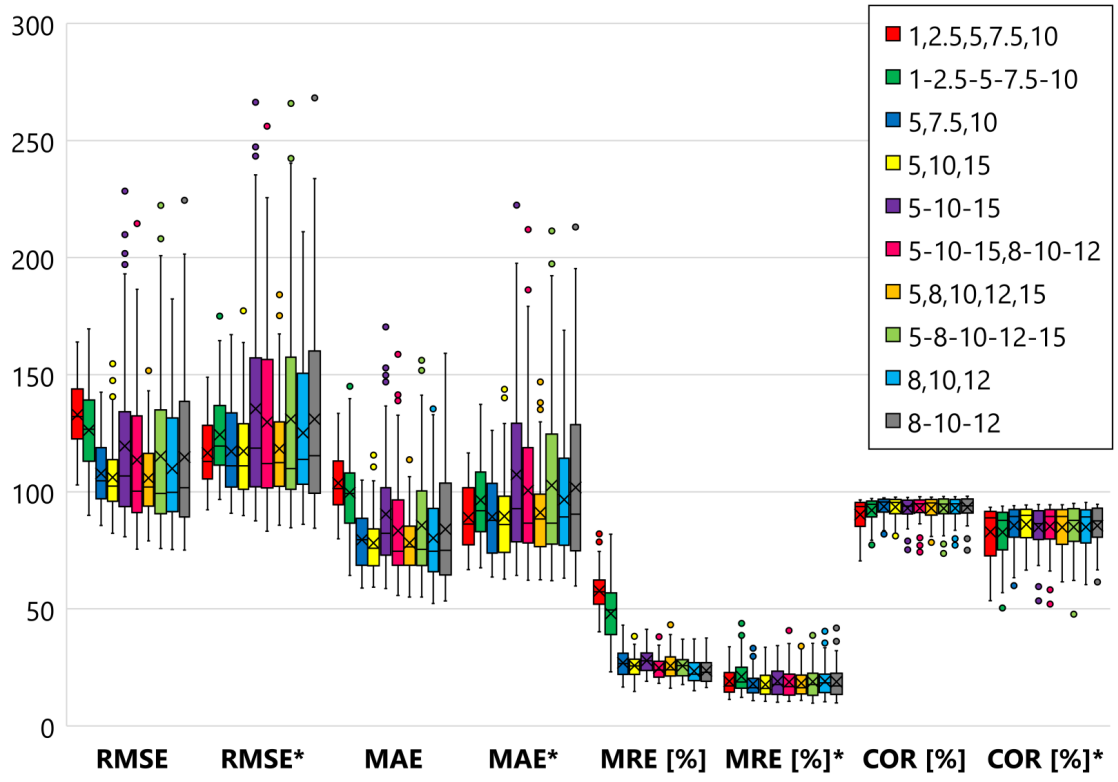


Figure 7.22: IA 8-categories derived and variable PR dataset CPR test results box plot grouped by training PR.

Figure 7.24 shows results grouped by training PR in box plot representation. The size of combined PR dataset (1,2.5,5,7.5,10) is 50,556 samples, VPR single version (1-2.5-5-7.5-10) is 10,102 samples and VPR multiple versions (1-2.5-5-7.5-10*) is 50,534 samples. The multiple version VPR dataset outperforms all other training datasets. This concludes that the ideal training dataset should consist of multiple versions of VPR-based traffic data collected during one week.

7.5 Conclusions

In this Chapter, modelling, simulation and validation of traffic forecasting system is presented based on VIL using SUMO for simulation and GBM for traffic modelling and prediction. A real traffic intersection from the City of Turin wasselected to be modelled in detail in SUMO. This network was simulated with real traffic flow archived from 5T open data. VIL data was collected from the simulation by creating multiple scenarios using multiple parameters. The collected data was used to train and test GBM models. Experiments were conducted with AS, DT

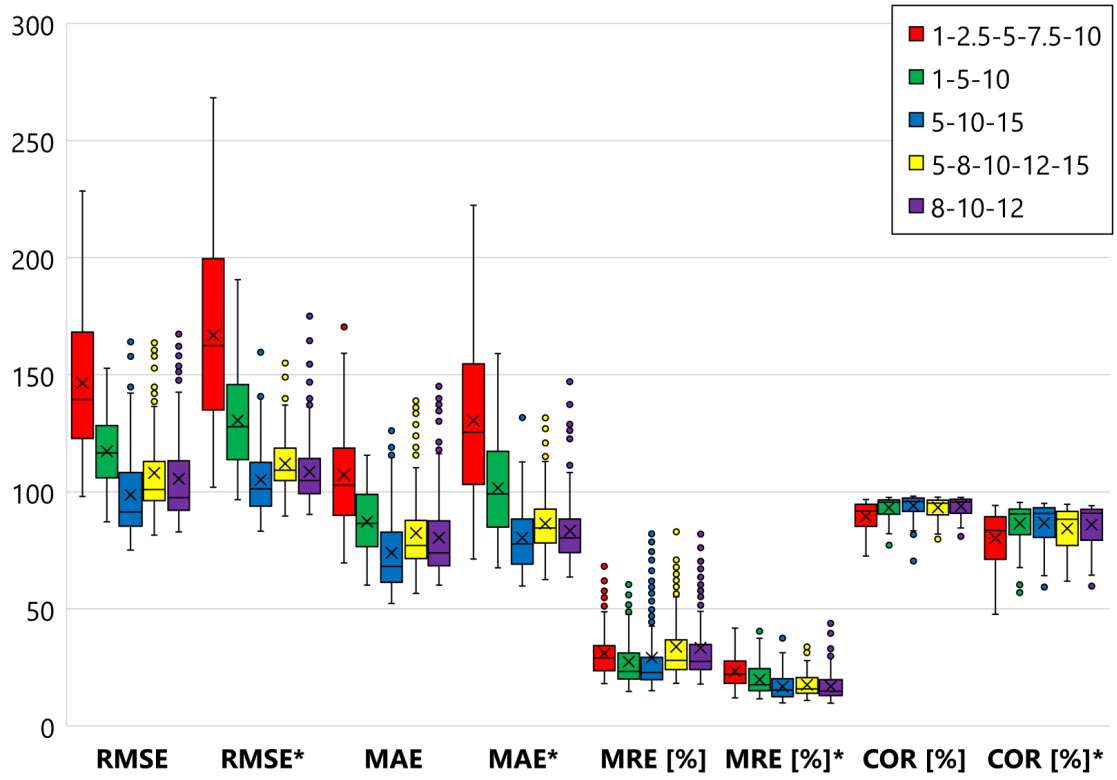


Figure 7.23: IA 8-categories derived and variable PR dataset CPR test results box plot grouped by testing PR.

and AS&DT data and concluded that using only 5 Important Attributes (IA) is sufficient for performance. CRS and CPR style tests were performed for all data types with multiple PRs. Classification style prediction schemes were also tested apart from regression style schemes. Multiple tests with real days dataset validated the concept of VIL in the simulation of a real traffic network. DPR datasets proved that during the training phase of the real system, VIL data can be collected once only at a fixed PR and later other lower PRs can be derived from it. This makes the implementation of the system feasible. VPR datasets were created since the real PR during an actual day might be variable. This enabled the simulation of the most realistic replica of a real day. The system was able to forecast traffic in all scenarios with reasonable accuracies considering realistic PRs.

Results from CRS tests show that a GBM machine trained with AS data can achieve on average an RMSE of 98.74 and an MRE of 24% with 10% of PR. The same averages for a GBM machine trained with DT data are RMSE of 97.74 and an MRE of 27.17%. When AS and DT are combined the averages reach an RMSE of 93.98 and an MRE of 22.97%. IA data drastically simplifies the model while

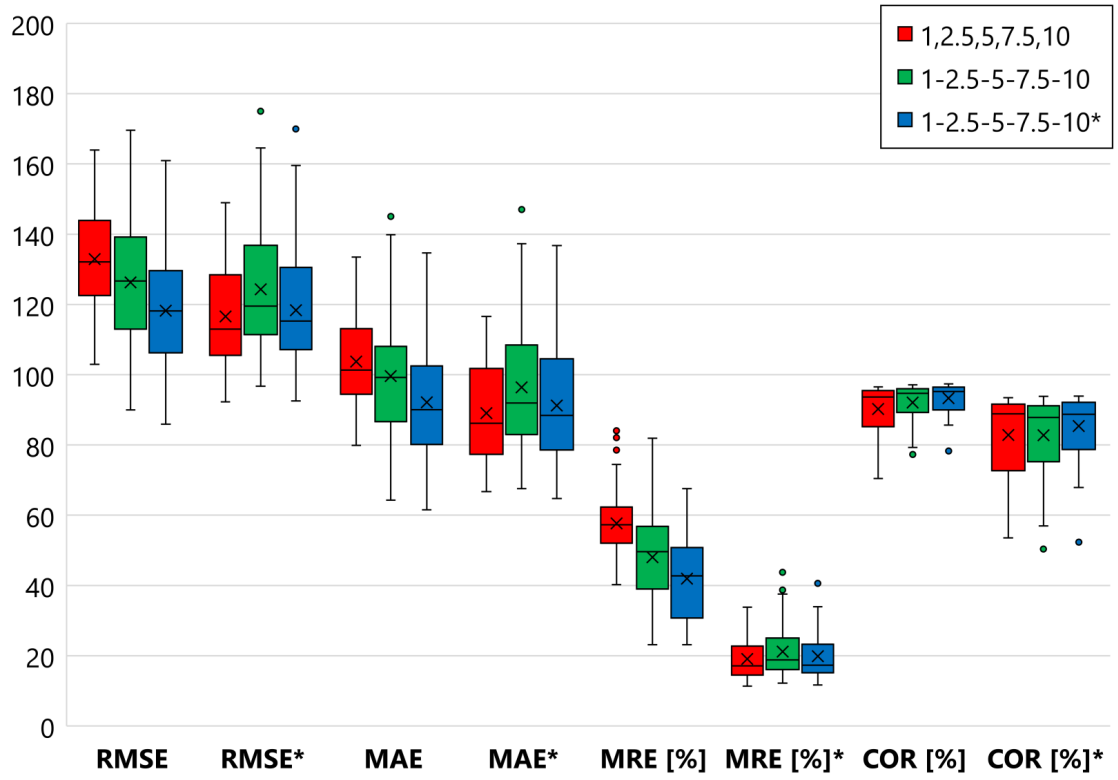


Figure 7.24: Training dataset size comparison shown by box plot grouped by training PR.

achieving an average RMSE of 96.51 and an MRE of 23.77%. CPR tests concluded that the performance of the prediction machine increases as the difference between the training and testing PR reduces. As far as classification is concerned, a GBM machine can on average achieve an accuracy of 94.96% with 2 classes. CRS tests on real days dataset showed that best scheme to train machines is with data from weeks from different months plus holidays. This machine on average gets an RMSE of 99.47 and an MRE of 24.06%. CPR tests with the same dataset show that machines trained with multiple training PRs outperform those which are trained with a single PR. Simulation of semi-realistic real days using VPR dataset showed that it is better to train a machine with VPR to get better results. A single GBM machine can forecast traffic for a case which is closest to reality in terms of PR with an MRE* of 20%. Lastly, as far as the size of the training dataset is concerned, it was concluded that the ideal training dataset should consist of multiple versions of VPR-based traffic data collected during one week.

Chapter 8

Conclusions

To conclude, this thesis discussed two major points in the ITS domain.

1. The collection and processing of data from road traffic using smartphones and other devices (such as On-Board Units (OBUs)).

   (a) I proposed a procedure to convert any measure taken by a smartphone sensor into the vehicle coordinate system in real-time in Chapter 2. It uses information from low power IMU (accelerometer and magnetometer) and GPS to perform data conversion, applying first a 3D rotation (from smartphone to Earth coordinates) and then a 2D rotation (from Earth to vehicle coordinates). With this procedure, the driver of a vehicle is no more constrained to place their smartphone in a cradle all along a trip but can leave it in a bag, pocket or even handle it for short periods. To obtain this result, accuracy was traded with usability; reducing the first one to increase the second one. The result is a very low penalty in accuracy, negligible in most ITS applications, and a very high increase in usability which is a factor of paramount importance for any customer-oriented application. This is the first solution, to the best of my knowledge, that can achieve real-time axis remapping with reasonable accuracy without placing any restrictions on the state of the device or driver. My approach enables the implementation of numerous ITS applications without installing dedicated hardware and using only already available mobile devices.

   (b) Chapter 3 describes in detail the idea and the proposed structure of a modular platform for the collection and processing of data for ITS related applications. The platform called VDAP consists of a smartphone and a back-end server section. VDAP is highly customisable and can be adapted to provide a number of ITS related services. More details about the design and implementation of two applications using VDAP are available in Chapter 4 and Chapter 5.

(c) Chapter 4 demonstrates a sample use case of VDAP for the recognition and analysis of the driving style of a driver with a freely placed smartphone. The algorithm utilises VDDA for full-calibration and data rotation of IMU. Additionally, it discusses in detail the APIs used from Android's SDK. DSA is essentially a two-part approach consisting of: 1) recognition of driving events or manoeuvres; 2) assignment of driving scores. DSA is able to recognise left/right turns, acceleration, deceleration, lateral acceleration and stop events with quite a high accuracy. The driving scores are constructed using the ranks (or intensities) of individual events and a comparison with other drivers' trips on that particular stretch of road. A scoring scheme is used to penalise higher or lower compared values (such as average speed, event ranks etc.).

2. The use of machine learning techniques to predict the intensity of traffic using crowd-sourced data from a small fraction of the traffic.

   (a) Chapter 5 presents a completely software-based solution with already available and ubiquitous hardware, *i.e.* smartphones. It was proven that the feasibility of using smartphones to collect data and provide highly accurate information about a passage of a user over pre-defined goals. Simulation results based on real vehicular traces show negligible timestamp calculation errors with nearly 100% success rate. Moreover, due to the flexible and scalable nature of VIL, the goals can be defined in real-time to focus on areas of special interest. A practical integration scheme is proposed to allow UTC systems, such as UTOPIA, to benefit from VIL.

   (b) Chapter 6 formulates a system for forecasting urban traffic over a short time period using Deep Learning. The prediction is accurate while being simple in terms of complexity. It also proposes and analyses the effects of multiple pre-processing schemes to improve the accuracy of forecasting. From experiments on a real traffic flow dataset from the City of Turin, it is evident that my Deep Learning machine performs better than state-of-the-art DLAs. Results show that my solution outperforms other DLAs with nearly 4% accuracy improvements while being much simpler in terms of complexity (hidden layers and hidden neurons). The most effective way to pre-process data is to use a simple moving average without timestamp as an input. This means that the machine can generate a prediction with only the traffic flow of past few minutes without any knowledge of the time. Moreover, to establish the robustness of my approach, a cross-examination of my architecture with different datasets was done. Results show that a single Deep Learning machine with only two hidden layers of 50 units each trained with traffic flow data of a

week can predict the flow on any day of the week as well as holidays with remarkable accuracy.

(c) In Chapter 7, modelling, simulation and validation of traffic forecasting system was presented based on VIL using SUMO for simulation and GBM for traffic modelling and prediction. A real traffic intersection from the City of Turin was selected to be modelled in detail in SUMO. This network was simulated with real traffic flow archived from 5T open data. VIL data was collected from the simulation by creating multiple scenarios using multiple parameters. The collected data was used to train and test GBM models. Experiments were conducted with AS, DT and AS&DT data and concluded that using only 5 Important Attributes (IA) is sufficient for performance. CRS and CPR style tests were performed for all data types with multiple PRs. Classification style prediction schemes were also tested apart from regression style schemes. Multiple tests with real days dataset validated the concept of VIL in the simulation of a real traffic network. DPR datasets proved that during the training phase of the real system, VIL data can be collected once only at a fixed PR and later other lower PRs can be derived from it. This makes the implementation of the system feasible. VPR datasets were created since the real PR during an actual day might be variable. This enabled the simulation of the most realistic replica of a real day. The system was able to forecast traffic in all scenarios with reasonable accuracies considering realistic PRs.

Appendix A

Traffic forecasting with VIL results

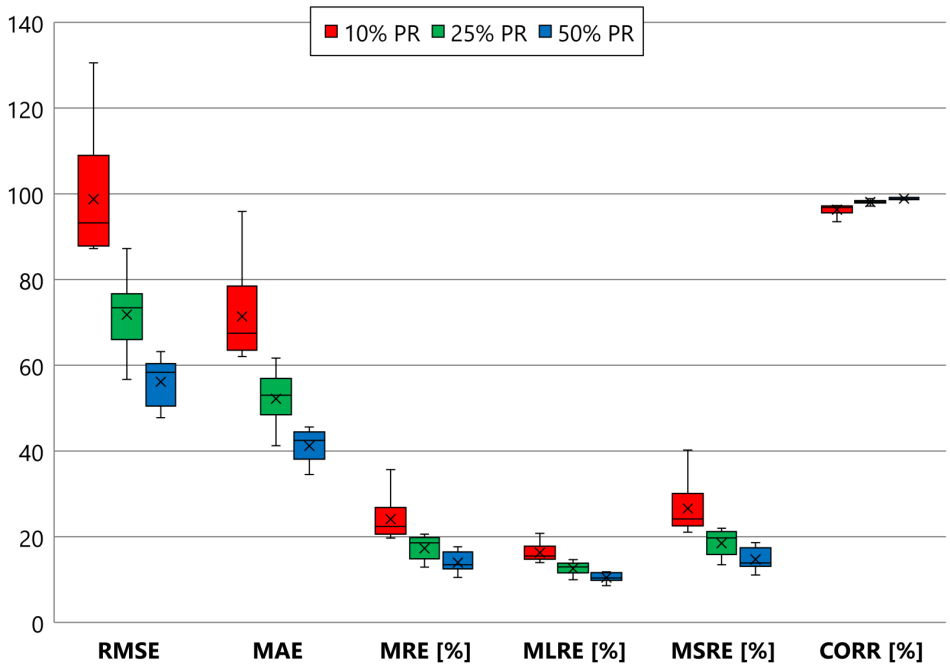


Figure A.1: AS dataset CRS test results.

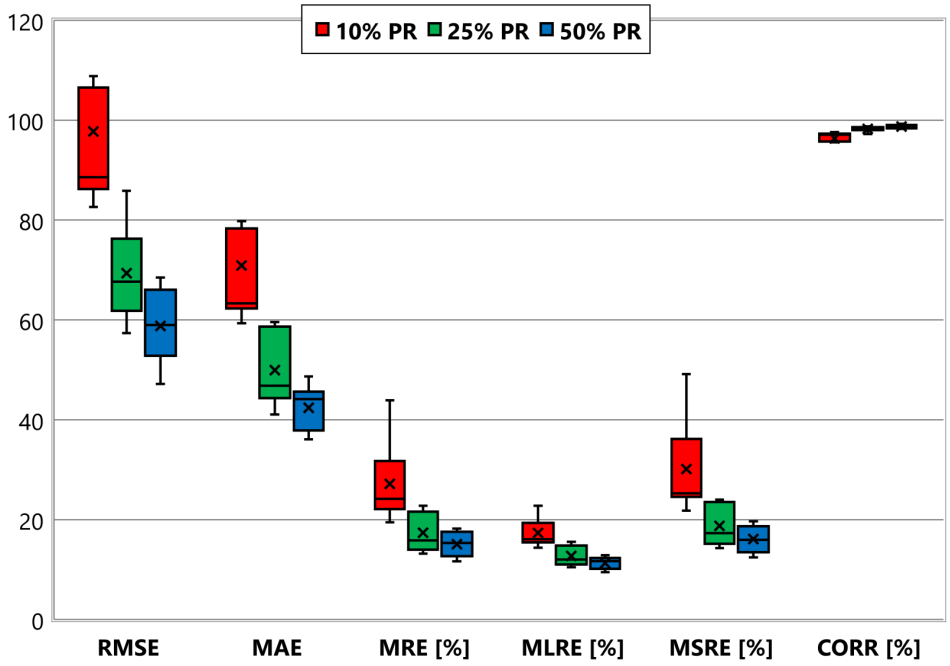


Figure A.2: DT dataset CRS test results.

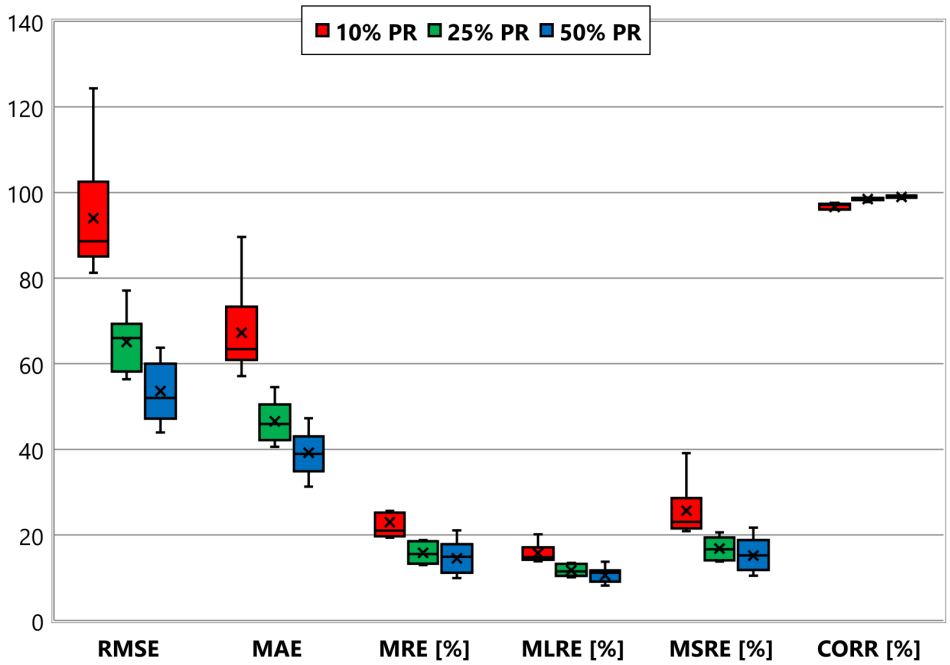


Figure A.3: AS&DT dataset CRS test results.

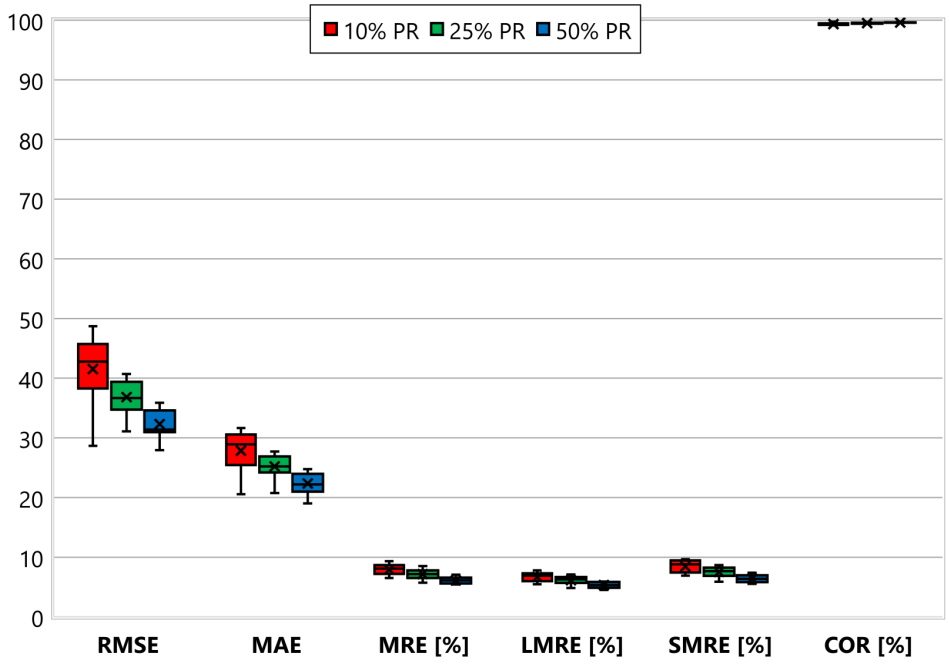


Figure A.4: AS&DT&H dataset CRS test results.

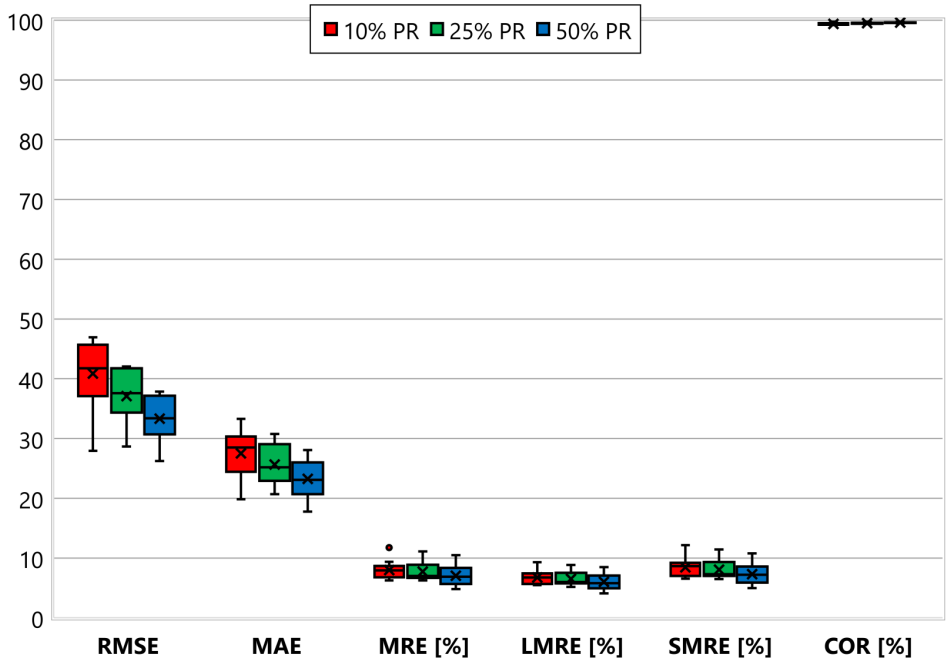


Figure A.5: IA&H dataset CRS test results.

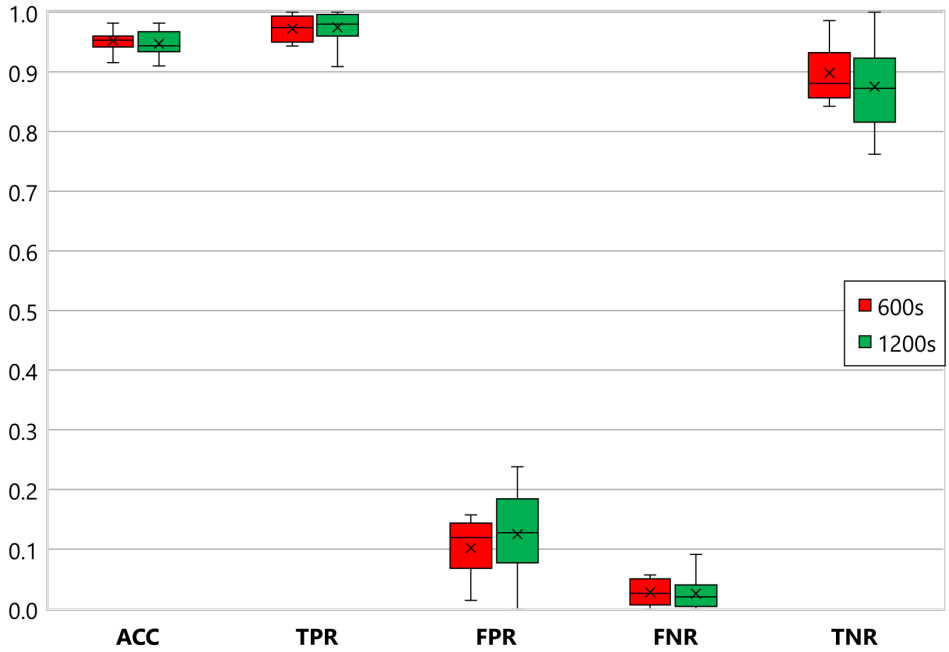


Figure A.6: IA_2CH dataset CRS test results box plot grouped by binning window size.

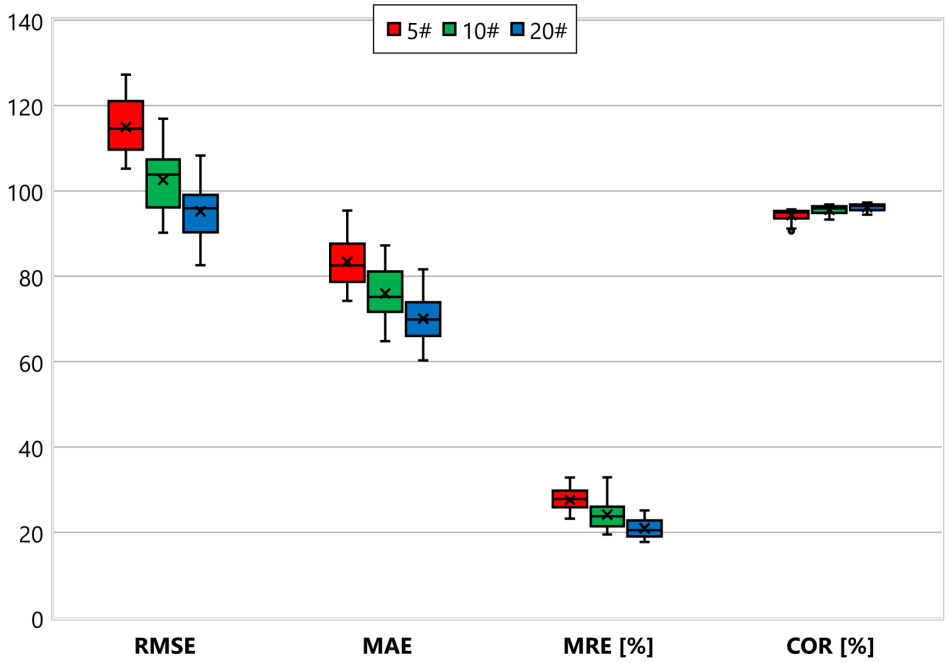


Figure A.7: M_AS&DT dataset CRS test results.

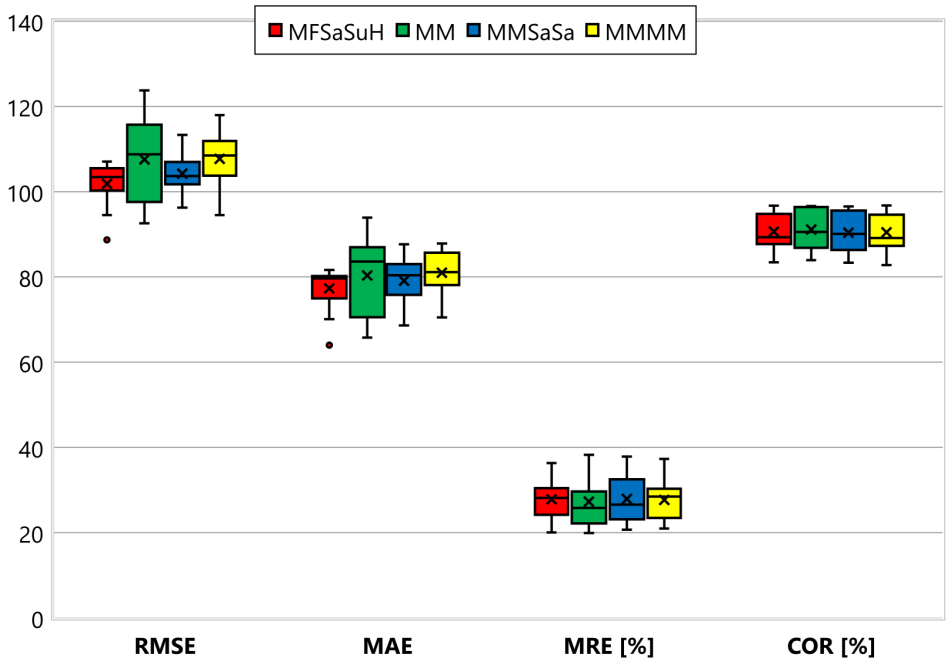


Figure A.8: IA 5-categories dataset CRS test results box plot grouped by training PR.

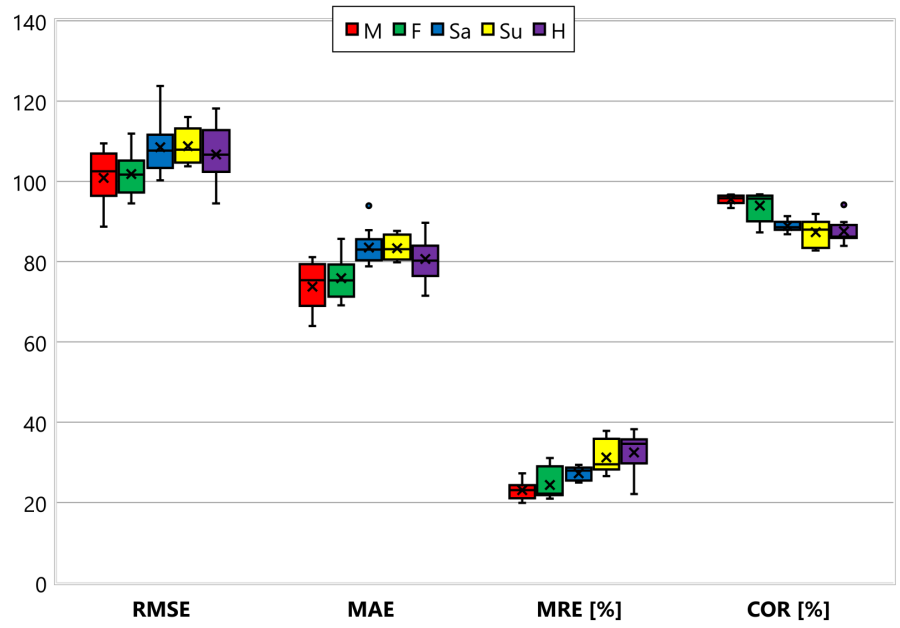


Figure A.9: IA 5-categories dataset CRS test results box plot grouped by testing PR.

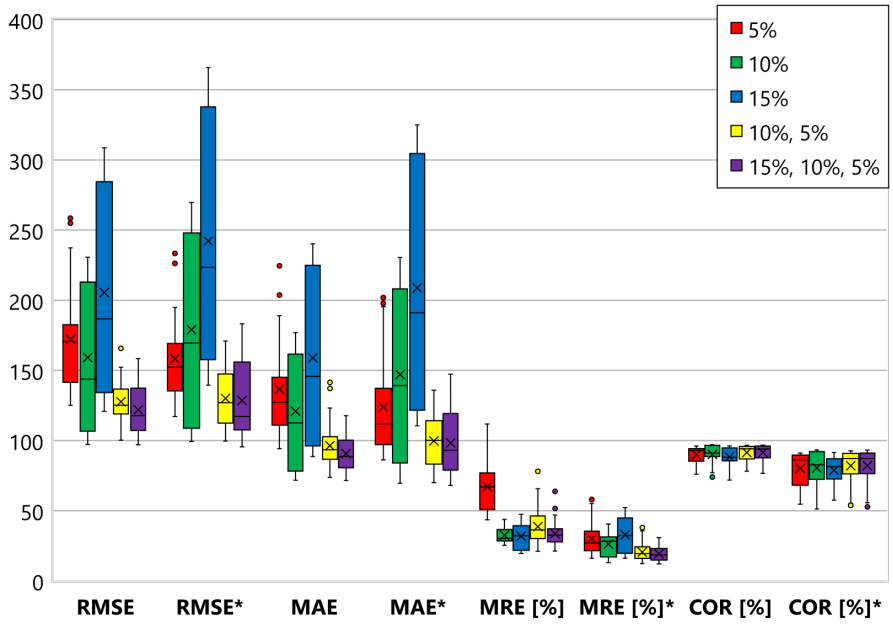


Figure A.10: IA 8-categories dataset CPR test results box plot grouped by training PR.

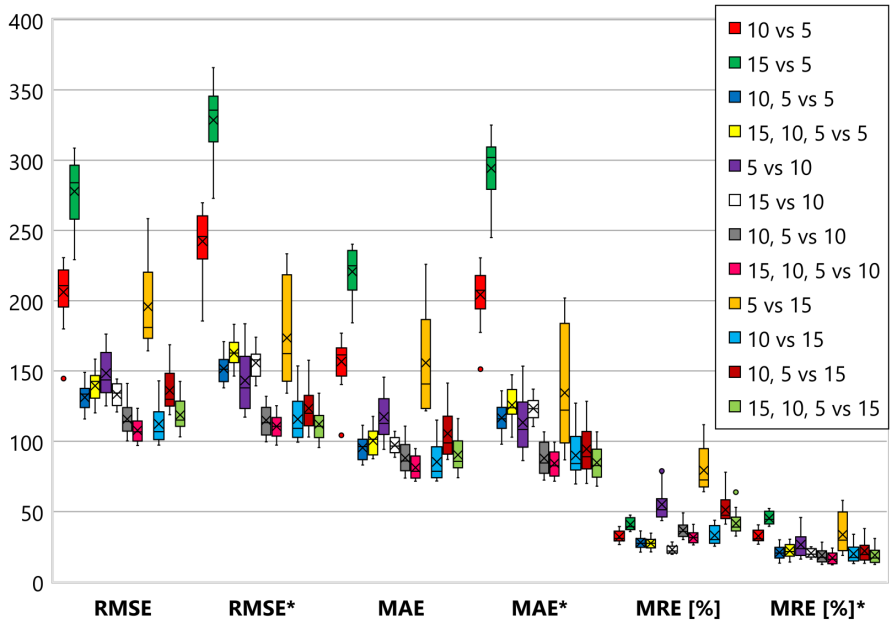


Figure A.11: IA 8-categories dataset CPR test results box plot grouped by testing PR.

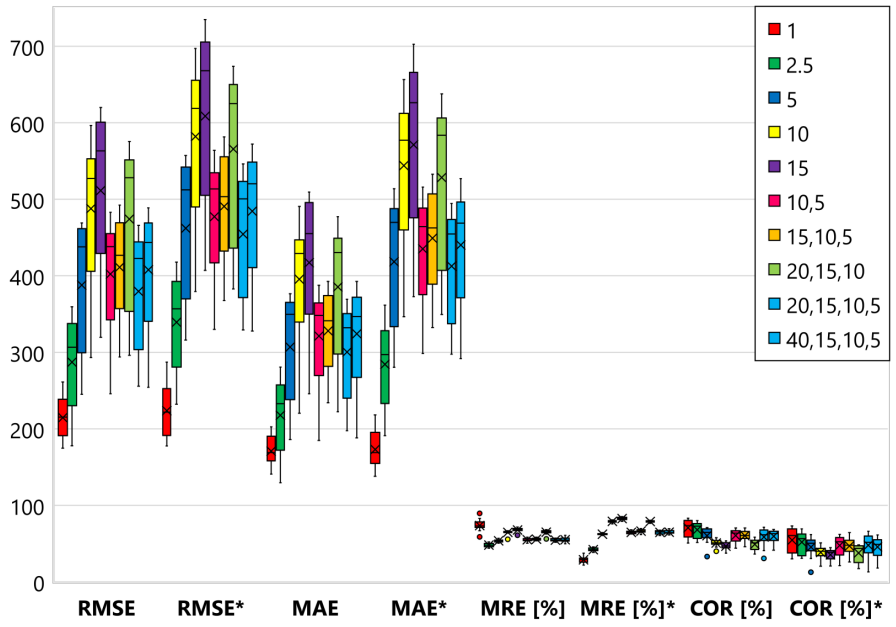


Figure A.12: IA 8-categories DPR dataset CPR test results box plot testing only with 1% PR.

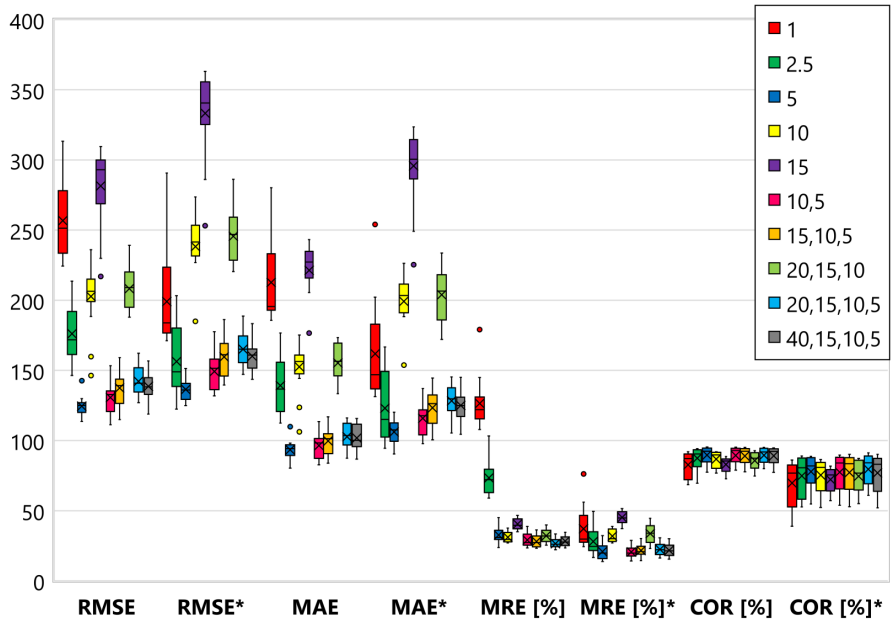


Figure A.13: IA 8-categories DPR dataset CPR test results box plot testing only with 5% PR.

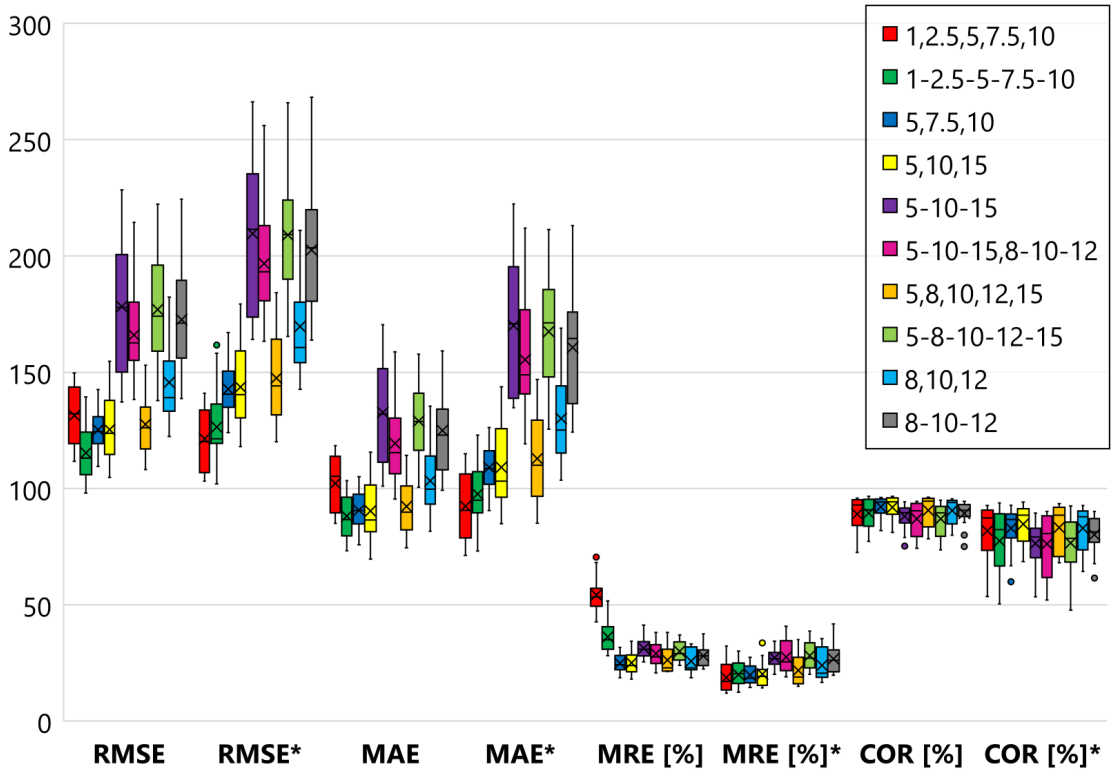


Figure A.14: IA 8-categories derived and variable PR dataset CPR test results box plot testing only with 1-2.5-5-7.5-10 VPR.

Acronyms

ACC Accuracy.

AI Artificial Intelligence.

ANN Artificial Neural Network.

API Application Programming Interface.

ARIMA Auto-Regressive Integrated Moving Average.

AS Average Speed.

ATCS Adaptive Traffic Control System.

ATMS Advanced Traffic Management System.

BLE Bluetooth Low Energy.

BPNN Back Propagation Neural Network.

CAN Controller Area Network.

CGA Corso Giovanni Agnelli.

COR Correlation.

CPR Cross Penetration Rate.

CPU Central Processing Unit.

CRS Cross Random Seed.

CS Corso Sebastopoli.

CSV Comma-Separated Values.

DBMS DataBase Management System.

DLA Deep Learning Architecture.

DPR Derived Penetration Rate.

DSA Driving Style Analysis.

DT Delta Time.

EU European Union.

FCM Firebase Cloud Messaging.

FNR False Negative Rate.

FPR False Positive Rate.

GBM Gradient Boosted Machine.

GCM Google Cloud Messaging.

GDP Gross Domestic Product.

GPS Global Positioning System.

HTML HyperText Markup Language.

HTTP HyperText Transfer Protocol.

IA Important Attributes.

IL Induction Loop.

IMU Inertial Measurement Unit.

INS Inertial Navigation System.

ITS Intelligent Transportation System.

JSON JavaScript Object Notation.

LWL Locally Weighted Learning.

MA Mean Accuracy.

MAE Mean Absolute Error.

MEMS Micro-Electro-Mechanical System.

MIROAD Mobile sensor platform for Intelligent Recognition of Aggressive Driving.

MLRE Mean Lenient Relative Error.

MRE Mean Relative Error.

MSRE Mean Strict Relative Error.

NN Neural Network.

OBD On-Board Diagnostics.

OBU On-Board Unit.

OS Operating System.

OSM OpenStreetMap.

PR Penetration Rate.

RAM Random-Access Memory.

RBFNN Radial Basis Function Neural Network.

REST REpresentational State Transfer.

RMSE Root Mean Square Error.

RSU Road Side Unit.

RW Random Walk.

SAE Stacked Auto-Encoder.

SDK Software Development Kit.

SMA Simple Moving Average.

SOAP Simple Object Access Protocol.

SQL Structured Query Language.

SUMO Simulation of Urban MObility.

SVM Support Vector Machine.

SVR Support Vector Regression.

TLS Traffic Light Sequence.

TMC Traffic Message Channel.

TNR True Negative Rate.

TPR True Positive Rate.

UART Universal Asynchronous Receiver-Transmitter.

URL Uniform Resource Locator.

USB Universal Serial Bus.

UTC Urban Traffic Control.

UTOPIA Urban Traffic OPtimisation by Integrated Automation.

VDAP Vehicle Data Acquisition Platform.

VDDA Vehicle Dynamics Data Acquisition.

VF Via Filadelfia.

VIL Virtual Induction Loop.

VPR Variable Penetration Rate.

VSM Via San Marino.

WGS World Geodetic System.

Wi-Fi Wireless Fidelity.

XML eXtensible Markup Language.

Bibliography

[1] International Energy Agency. *World Energy Outlook 2017*. International Energy Agency, Nov. 14, 2017. URL: https://www.iea.org/weo2017/ (visited on 11/14/2017).

[2] J. Almazan et al. "Full auto-calibration of a smartphone on board a vehicle using IMU and GPS embedded sensors". In: *Proc. IEEE Intelligent Vehicles Symposium*. 2013, pp. 1374–1380. DOI: 10.1109/IVS.2013.6629658.

[3] R. L. Anderson. "Electromagnetic Loop Vehicle Detectors". In: *IEEE Trans. on Vehicular Technology* VT-19.1 (1970), pp. 23–30. DOI: 10.1109/T-VT.1970.23428.

[4] R. Araújo et al. "Driving coach: A smartphone application to evaluate driving efficient patterns". In: *Proc. IEEE Intelligent Vehicles Symposium*. 2012, pp. 1005–1010. DOI: 10.1109/IVS.2012.6232304.

[5] G. Bishop and G. Welch. "An Introduction to the Kalman Filter". In: *Proc of SIGGRAPH, Course* 8.27599-3175 (2001), p. 41. URL: https://www.cs.unc.edu/~welch/media/pdf/kalman_intro.pdf.

[6] E. Bolshinsky and R. Freidman. *Traffic Flow Forecast Survey*. Tech. rep. Computer Science Department, Technion, 2012, pp. 1–15. URL: http://www.cs.technion.ac.il/users/wwwb/cgi-bin/tr-get.cgi/2012/CS/CS-2012-06.pdf.

[7] G. E. P. Box and D. A. Pierce. "Distribution of Residual Autocorrelations in Autoregressive-Integrated Moving Average Time Series Models". In: *American Statistical Association Stable* 65.332 (2008), pp. 1509–1526. DOI: 10.2307/2284333.

[8] M. Castro-neto et al. "SVR for short-term traffic flow prediction under typical and atypical traffic conditions". In: *Expert Systems With Applications* 36.3 (2009), pp. 6164–6173. DOI: 10.1016/j.eswa.2008.07.069.

[9] G. C. Cawley and N. L. C. Talbot. "On over-fitting in model selection and subsequent selection bias in performance evaluation". In: *Journal of Machine Learning Research* 11 (2010), pp. 2079–2107. URL: https://ueaeprints.uea.ac.uk/3640/.

[10] S. Clark. "Traffic Prediction Using Multivariate Nonparametric Regression". In: *Journal of Transportation Engineering* 129.2 (2003), pp. 161–168. DOI: 10.1061/(ASCE)0733-947X(2003)129:2(161).

[11] J. Eriksson et al. "The pothole patrol". In: *Proc. 6th Int. Conf. on Mobile systems, applications, and services - MobiSys '08* (2008), p. 29. DOI: 10.1145/1378600.1378605.

[12] European Commission. *Roadmap to a Single European Transport Area - Towards a competitive and resource efficient transport system*. URL: https://ec.europa.eu/transport/themes/strategies/2011_white_paper_en.

[13] Eurostat. *News Release - Internet use by individuals*. URL: https://ec.europa.eu/eurostat/documents/2995521/7771139/9-20122016-BP-EN.pdf.

[14] Eurostat. *Passenger cars in the EU*. European Commission, Apr. 1, 2018. URL: https://ec.europa.eu/eurostat/statistics-explained/pdfscache/25886.pdf (visited on 04/01/2018).

[15] jQuery Foundation. *jQuery API Documentation*. URL: https://api.jquery.com/ (visited on 03/31/2018).

[16] K. Fox and S. Clark. "Evaluating the benefits of a responsive UTC system using microsimulation". In: *Institute for Transport Studies, University of Leeds, Leeds, UK.* (1998). URL: http://www.its.leeds.ac.uk/projects/flows/utsgkaf.pdf.

[17] German Aerospace Center - Institute of Transportation Systems. *Eclipse SUMO - Simulation of Urban MObility source code*. Github.com (12 May 2019). Jan. 2019. URL: https://github.com/eclipse/sumo/blob/master/tools/osmWebWizard.py.

[18] German Aerospace Center - Institute of Transportation Systems. *Simulation/Traffic Lights - Sumo*. URL: http://sumo.dlr.de/wiki/Simulation/Traffic_Lights#Signal_state_definitions (visited on 05/21/2018).

[19] B. Ghosh, B. Basu, and M. O'Mahony. "Bayesian Time-Series Model for Short-Term Traffic Flow Forecasting". In: *Journal of Transportation Engineering* 133.3 (2007), pp. 180–189. DOI: 10.1061/(ASCE)0733-947X(2007)133:3(180).

[20] statcounter GlobalStats. *Operating System Market Share Worldwide | StatCounter Global Stats*. 2018. URL: http://gs.statcounter.com/os-market-share (visited on 02/08/2019).

[21] Google. *ActivityRecognitionClient | Google APIs for Android | Google Developers*. URL: https://developers.google.com/android/reference/com/google/android/gms/location/ActivityRecognitionClient (visited on 03/31/2018).

[22] Google. *BluetoothDevice | Android Developers*. URL: https://developer.android.com/reference/android/bluetooth/BluetoothDevice.html (visited on 03/31/2018).

[23] Google. *Google Maps JavaScript API | Google Developers*. URL: https://developers.google.com/maps/documentation/javascript/ (visited on 03/31/2018).

[24] Google. *Google Places API Web Service | Google Developers*. URL: https://developers.google.com/places/web-service/ (visited on 03/31/2018).

[25] Google. *Motion Sensors | Android Developers*. URL: https://developer.android.com/guide/topics/sensors/sensors_motion.html (visited on 03/31/2018).

[26] Google. *Overview of Google Play Services | Google APIs for Android | Google Developers*. URL: https://developers.google.com/android/guides/overview (visited on 03/31/2018).

[27] Google. *Place IDs | Google Places API | Google Developers*. URL: https://developers.google.com/places/place-id (visited on 03/31/2018).

[28] Google. *Position Sensors | Android Developers*. URL: https://developer.android.com/guide/topics/sensors/sensors_position.html (visited on 03/31/2018).

[29] Google. *Sensor | Android Developers*. URL: https://developer.android.com/reference/android/hardware/Sensor.html (visited on 03/31/2018).

[30] Google. *Sensor Manager - Android Developers - getRotationMatrix*. Feb. 2017. URL: https://developer.android.com/reference/android/hardware/SensorManager.html#getRotationMatrix(float[],%20float[],%20float[],%20float[]) (visited on 05/14/2019).

[31] Google. *SensorManager | Android Developers*. URL: https://developer.android.com/reference/android/hardware/SensorManager.html (visited on 03/31/2018).

[32] Google. *Sensors Overview - Android Developers*. Feb. 2017. URL: https://developer.android.com/guide/topics/sensors/sensors_overview.html (visited on 01/13/2019).

[33] Google. *Snap to Roads | Google Maps Roads API | Google Developers*. URL: https://developers.google.com/maps/documentation/roads/snap (visited on 03/31/2018).

[34] M. Gramaglia, C. J. Bernardos, and M. Calderon. "Virtual induction loops based on cooperative vehicular communications". In: *Sensors* 13.2 (2013), pp. 1467–1476. DOI: 10.3390/s130201467.

[35] M. S. Grewal, V. D. Henderson, and R. S. Miyasako. "Application of Kalman Filtering to the Calibration and Alignment of Inertial Navigation Systems". In: *IEEE Trans. on Automatic Control* 36.1 (1991), pp. 3–13. DOI: 10.1109/9.62283.

[36] GSM Association. *GSMA Mobile Economy 2018 - Europe 2018*. GSM Association, 2018. URL: https://www.gsmaintelligence.com/research/?file=884c77f3bc0a405b2d5fd356689be340&download.

[37] H2O.ai. *H2O - Flow interactive notebook software, Version 3.18.0.8*. H2O.ai (22 May 2018). 2018. URL: http://docs.h2o.ai/h2o/latest-stable/h2o-docs/flow.html.

[38] H2O.ai. *H2O Open-source Library Software, Version 3.18.0.8*. H2O.ai (22 May 2018). 2018. URL: https://www.h2o.ai/h2o/.

[39] H2O.ai. *H2O Open-source Library Software, Version 3.8.2.6*. H2O.ai (24 May 2016). May 2016. URL: https://www.h2o.ai/h2o/.

[40] W. Hassan. *5T Data Logger source code*. Github.com (12 May 2019). May 2019. URL: https://github.com/hassanwaqar00/5t_traffic_data_logger.

[41] W. Hassan. *5T-SUMO data processor source code*. Github.com (12 May 2019). May 2019. URL: https://github.com/hassanwaqar00/5t_sumo_processor.

[42] W. Hassan. *SUMO simulation network, routes file and Python script*. Github.com (12 May 2019). May 2019. URL: https://github.com/hassanwaqar00/5t_sumo_simulation.

[43] W. Hassan. *Traffic forecasting with Virtual Induction Loops - SUMO simulation dataset*. May 2019. DOI: 10.5281/zenodo.2648393.

[44] J. C. Herrera et al. "Evaluation of traffic data obtained via GPS-enabled mobile phones: The Mobile Century field experiment". In: *Transportation Research Part C: Emerging Technologies* 18.4 (2010), pp. 568–583. DOI: 10.1016/j.trc.2009.10.006.

[45] C. P. V. Hinsbergen, J. W. V. Lint, and F. M. Sanders. "Short Term Traffic Prediction Models". In: *Proc. 14th World Congress on Intelligent Transport Systems (ITS)* (Oct. 2007), p. 18. DOI: 10.1037/11584-022.

[46] S. Hochreiter et al. "Neural-network-based models for short-term traffic flow forecasting using a hybrid exponential smoothing and levenberg-marquardt algorithm". In: *IEEE Trans. on Intelligent Transportation Systems* 13.2 (2012), pp. 644–654. DOI: 10.1109/TITS.2011.2174051.

[47] W. C. Hong et al. "Hybrid evolutionary algorithms in a SVR traffic flow forecasting model". In: *Applied Mathematics and Computation* 217.15 (2011), pp. 6733–6747. DOI: 10.1016/j.amc.2011.01.073.

[48] W. Huang et al. "Deep Architecture for Traffic Flow Prediction: Deep Belief Networks With Multitask Learning." In: *IEEE Trans. on Intelligent Transportation Systems* 15.5 (2014), pp. 2191–2201. DOI: 10.1109/TITS.2014.2311123.

[49] IDC. *IDC - Smartphone Market Share - OS*. 2018. URL: https://www.idc.com/promo/smartphone-market-share/os (visited on 03/12/2019).

[50] INRIX. *INRIX Global Traffic Scorecard*. INRIX, 2017. URL: http://inrix.com/scorecard/.

[51] D. A. Johnson and M. M. Trivedi. "Driving style recognition using a smartphone as a sensor platform". In: *Proc. IEEE Conf. on Intelligent Transportation Systems ITSC*. 2011, pp. 1609–1615. DOI: 10.1109/ITSC.2011.6083078.

[52] A. Krizhevsky, I. Sutskever, and G. Hinton. "ImageNet classification with deep convolutional neural networks". In: *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc., 2012, pp. 1097–1105. DOI: 10.1145/3065386.

[53] G. Leduc. "Road Traffic Data : Collection Methods and Applications". In: *Technical Note: JRC 47967* 1.55 (2008), pp. 1–55. URL: ftp://ftp.jrc.es/pub/EURdoc/JRC47967.TN.pdf.

[54] H. Link et al. *The costs of road infrastructure and congestion in Europe*. Springer Science & Business Media, 2012, p. 10. DOI: 10.1007/978-3-642-58660-6.

[55] Z. C. Lipton. "The mythos of model interpretability". In: *ICML Workshop on Human Interpretability in Machine Learning (WHI 2016)*. 2016. URL: https://arxiv.org/abs/1606.03490.

[56] V. C. Magaña and M. M. Organero. "Artemisa: An eco-driving assistant for Android Os". In: *Proc. IEEE Int. Conf. on Consumer Electronics*. 2011, pp. 211–215. DOI: 10.1109/ICCE-Berlin.2011.6031794.

[57] V. Mauro and C. D. Taranto. "UTOPIA". In: *Control, Computers, Communications in Transportation*. IFAC Symposia Series. Oxford: Pergamon, 1990, pp. 245–252. DOI: 10.1016/B978-0-08-037025-5.50042-6.

[58] S. Maus et al. *The US/World Magnetic Model for 2010-2015*. Technical Report. National Geophysical Data Center, 2010. DOI: 10.7289/V5TH8JNW.

[59] P. Mohan, V. N. Padmanabhan, and R. Ramjee. "Nericell: using mobile smartphones for rich monitoring of road and traffic conditions". In: *Proc. 6th ACM Conf. on Embedded network sensor systems - SenSys '08* (2008), p. 323. DOI: 10.1145/1460412.1460450.

[60] Newzoo. *Global Mobile Market Report*. Newzoo, 2018. URL: https://newzoo.com/insights/rankings/top-50-countries-by-smartphone-penetration-and-users/.

[61] I. Okutani and Y. J. Stephanedes. "Dynamic prediction of traffic volume through Kalman filtering theory". In: *Transportation Research Part B* 18.1 (1984), pp. 1–11. DOI: 10.1016/0191-2615(84)90002-X.

[62] M. Papageorgiou et al. "Review of Road Traffic Control Strategies". In: *Proc. IEEE* 91.12 (Dec. 2003), pp. 2043–2067. DOI: 10.1109/JPROC.2003.819610.

[63] T. F. Park and S. Lee. "Sensor based activity detection". 2013. URL: https://www.google.com/patents/US8560229.

[64] F. Qiao, H. Yang, and W. H. K. Lam. "Intelligent simulation and prediction of traffic flow dispersion". In: *Transportation Research Part B: Methodological* 35.9 (2001), pp. 843–863. DOI: 10.1016/S0191-2615(00)00024-2.

[65] RapidMiner. *RapidMiner Studio Software, Version 7.6.0*. RapidMiner (17 Aug 2017). Aug. 2017. URL: https://rapidminer.com/products/studio/.

[66] *The Global Competitiveness Report 2018*. World Economic Forum, Oct. 16, 2018. URL: http://www3.weforum.org/docs/GCR2018/05FullReport/TheGlobalCompetitivenessReport2018.pdf (visited on 10/16/2018).

[67] M. Shuai et al. "An online approach based on locally weighted learning for short-term traffic flow prediction". In: *Proc. 16th ACM SIGSPATIAL Int. Conf. on Advances in geographic information systems - GIS '08*. ACM. 2008, p. 1. DOI: 10.1145/1463434.1463490.

[68] S. Skehan. "Adaptive Traffic Control System". In: *Compendium of Technical Papers for the 66th ITE Annual MeetingInstitute of Transportation Engineers (ITE)*. 1996, pp. 203–207. URL: https://trid.trb.org/view/481594.

[69] B. L. Smith and M. J. Demetsky. "Short-term traffic flow prediction models-a comparison of neural network and nonparametric regression approaches". In: *Proc. IEEE Int. Conf. on Systems, Man and Cybernetics*. Vol. 2. IEEE. 1994, pp. 1706–1709. DOI: 10.1109/ICSMC.1994.400094.

[70] 5T srl. *City of Turin - Traffic flows (Real time)*. Feb. 2011. URL: http://opendata.5t.torino.it/get_fdt.

[71] L. Strahilevitz. "'How's My Driving?' for Everyone (and Everything?)" In: *New York University Law Review* 125.125 (2006), pp. 1699–1765. URL: https://ssrn.com/abstract=899144.

[72] S. Sun, C. Zhang, and G. Yu. "A Bayesian network approach to traffic flow forecasting". In: *IEEE Trans. on Intelligent Transportation Systems* 7.1 (2006), pp. 124–133. DOI: 10.1109/TITS.2006.869623.

[73] E. I. Vlahogianni, M. G. Karlaftis, and J. C. Golias. "Short-term traffic forecasting: Where we are and where we're going". In: *Transportation Research Part C: Emerging Technologies* 43 (2014), pp. 3–19. DOI: 10.1016/j.trc.2014.01.005.

[74] E. Vlahogianni and M. Karlaftis. "Temporal aggregation in traffic data: implications for statistical characteristics and model choice". In: *Transportation Letters* 3.1 (2011), pp. 37–49. DOI: 10.3328/TL.2011.03.01.37-49.

[75] J. Wahlstrom, I. Skog, and P. Handel. "IMU alignment for smartphone-based automotive navigation". In: *Proc. 18th Int. Conf. on Information Fusion*. IEEE. 2015, pp. 1437–1443. URL: https://ieeexplore.ieee.org/document/7266726.

[76] J. Wahlstrom, I. Skog, and P. Handel. "Smartphone-Based Vehicle Telematics: A Ten-Year Anniversary". In: *IEEE Trans. on Intelligent Transportation Systems* 18.10 (2017), pp. 2802–2825. DOI: 10.1109/TITS.2017.2680468.

[77] M. J. Walker. *Hype Cycle for Emerging Technologies, 2017*. Insight Report. Gartner, Inc., 2017. URL: https://www.gartner.com/document/3768572.

[78] P. W. Koh Wei and P. Liang. "Understanding black-box predictions via influence functions". In: *Proc. 34th Int. Conf, on Machine Learning*. Vol. 70. ICML'17. JMLR, 2017, pp. 1885–1894. URL: http://dl.acm.org/citation.cfm?id=3305381.3305576.

[79] G. Wellbrock et al. "First Field Trial of Sensing Vehicle Speed, Density, and Road Conditions by using Fiber Carrying High Speed Data". In: *2019 Optical Fiber Communications Conference and Exhibition (OFC)*. IEEE. 2019, pp. 1–3. DOI: 10.1364/OFC.2019.Th4C.7.

[80] J. White et al. "WreckWatch: Automatic traffic accident detection and notification with smartphones". In: *Mobile Networks and Applications* 16.3 (2011), pp. 285–303. DOI: 10.1007/s11036-011-0304-8.

[81] B. M. Williams and L. A. Hoel. "Modeling and Forecasting Vehicular Traffic Flow as a Seasonal ARIMA Process: Theoretical Basis and Empirical Results". In: *Journal of Transportation Engineering* 129.6 (2003), pp. 664–672. DOI: 10.1061/(ASCE)0733-947X(2003)129:6(664).

[82] K. Wood. "Urban traffic control; systems review." In: *TRL,Crowthorne, Tech. Rep. PR41* (1993), p. 53. ISSN: 0968-4093. URL: https://trl.co.uk/reports/PR41.

[83] L. Yisheng et al. "Traffic flow prediction with big data: a deep learning approach". In: *IEEE Trans. on Intelligent Transportation Systems* 16.2 (2015), pp. 865–873. DOI: 10.1109/TITS.2014.2345663.

[84] J. Zaldivar et al. “Providing accident detection in vehicular networks through OBD-II devices and android-based smartphones”. In: *Proc. Conf. on Local Computer Networks LCN*. 2011, pp. 813–819. DOI: 10.1109/LCN.2011.6115556.

[85] J. Zhang et al. “Data-driven intelligent transportation systems: A survey”. In: *IEEE Trans. on Intelligent Transportation Systems* 12.4 (2011), pp. 1624–1639. DOI: 10.1109/TITS.2011.2158001.

This Ph.D. thesis has been typeset by means of the TEX-system facilities. The typesetting engine was pdfLATEX. The document class was `toptesi`, by Claudio Beccari, with option `tipotesi=scudo`. This class is available in every up-to-date and complete TEX-system installation.