UNIVERSIDAD POLITECNICA DE VALENCIA

DEPARTAMENTO DE INFORMÁTICA DE SISTEMAS Y COMPUTADORES

Master Thesis
Academic year 2014/15
Master's degree in Computer Engineering

A STUDY OF WIRELESS DIGITAL POSTS AND TRAFFIC SIGNS USING SMARTPHONES

Author: **Carlos J Fernández Laguía**

Advisor: **Pietro Manzoni**

Abstract.

The importance of active safety has received a growing interest in recent years. Among the systems that compose the definition of active safety, traffic sign recognition is playing a vital role in a scenario where transportation systems are becoming increasingly autonomous. Most of the current implementations are focused on the use of cameras to capture and process images from the road. However due to some intrinsic limitations like weather, light conditions and partial occlusions there have been some proposals to enhance/replace camera systems with other technologies.

The present report presents and clasiffies some of the current proposals for traffic sign recognition. Analyzing the current and future situation and considering the investment on vehicles and road infrastructure an original system is proposed in this document, which is based on the smartphone as alternative platform to the in-build capture device. This system aims to close the gap between technical prerequisites and the requirement of potential users to easy an infrastructure change.

The results show that despite the smartphone constrains we have achieved a successful detection and recognition experience above 90 kilometers per hour. Moreover the system has been designed as a cost-effective solution that will be potentially upgraded in the future. Therefore flexibility and compatibility are important attributes that have underlined on every decision taken during the implementation process. Ultimately the project confirms that the use of smartphones represents an opportunity to expand wireless technology in the traffic sign recognition context.

“Vision is the art of seeing things invisible.”

Jonathan Swift

Content.

List of Figures

List of Tables

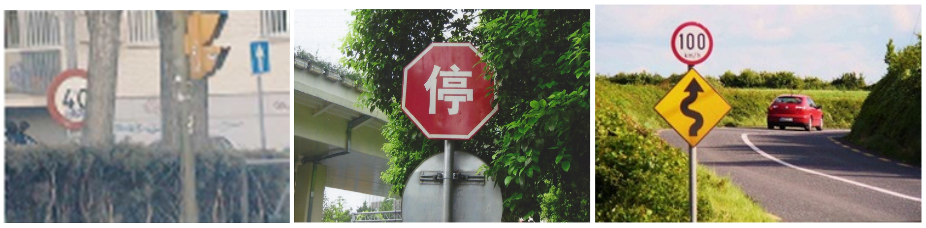
Abbreviations

ADB	Android Debug Bridge
AP	Access Point
API	Application Programming Interface
BSSID	Basic Service Set Identification
CALM	Communications Access for Land Mobile
CCMP	Counter Mode CBC-MAC Protocol
CPU	Central Processing Unit
DOT	U.S. Department of Transportation
GPU	Graphics Processing Unit
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DSRC	Dedicated Short Range Communications
EIRP	Effective Isotropic Radiated Power
FSM	Finite-State Machine
GLONASS	Global Navigation Satellite System
GPS	Global Positioning System
HAL	Hardware Abstraction Layer
IP	Internet Protocol
JNI	Java Native Interface
LTE	Long Term Evolution
MAC	Media Access Control
NFC	Near-Field Communication
OEM	Original Equipment Manufacturer
OS	Operating System
RFID	Radio-Frequency Identification
RSSI	Received Signal Strength
SD	Secure Digital
SDK	Software Development Kit
SoC	System on a Chip
SSID	Service Set Identification
TCP	Transmission Control Protocol
TSR	Traffic Sign Detection and Recognition
UI	User Interface
UMTRI	University of Michigan Transportation Research Institute
V2V	Vehicle-to-vehicle communications
VM	Virtual Machine
WAVE	Wireless Access in Vehicular Environment
WEXT	Wireless Extension
WLAN	Wireless Local Area Network
WPA	Wi-Fi Protected Access

Chapter 1.
Introduction

The future of transportation is outlined by more efficient, more autonomous and more connected vehicles. For this purpose the car will require an accurate knowledge about the environment. Vehicles are already being equipped with different set of sensors, cameras and transceivers with a common intention: collecting data from the "outside world". In this context traffic sign detection and recognition have drawn a lot of attention from the vehicle industry. The topic has inspired many publications in the literature, which has made their contributions to the concept in various different ways.

Most of these proposals are based on camera systems, which analyze images in order to detect and recognize a traffic sign or panel. In practice these systems seem to be a very good solution to detect speed limit indications, extending sensing capabilities of the driver in situations of lack of concentration or due to drowsiness. However there are some limitations on the nature of these systems that are critical in the future scenario where transportation aims to become an autonomous system.



(a) (b) (c)

Figure 1: Camera system limitations: (a) partial occlusions; (b) traffic sign not found in the database; (c) complex situations

The solutions that can be found in the market today are limited to detect the traffic signs stored on the database, which are typically speed limits and overtaking bans compliant with the Vienna convention. Even if these systems get upgraded to recognize any traffic sign that might come across on the way with the same efficiency, light and weather conditions would still play a critical role on the system reliability. Moreover the location of some traffic signs is sometimes not the most appropriate one causing partial occlusion. Unlike a computer, the driver easily overcomes these situations making decisions based on partial information. Also there are situation that are complex where the information from a traffic sign cannot be fulfilled to the letter, like figure 1 illustrates along with other examples where a camera-based system might fail to deliver a correct result.

The system proposed on the present document tries to overcome these limitations by the use of wireless technologies instead of cameras to capture the information from traffic signs and panels. We even go further developing a solution based on the smartphone platform, so drivers can adopt the solution not only on new vehicles, but also on the current stock of cars on the road.

1.1 Contributions

The five principal contributions of this thesis are: (1) Development of a complete hardware solution to equip traffic signs with wireless technology at a low cost; (2) a resource-efficient *Android* application with the implementation of the client solution, which would be installed on the driver's smartphone as proof of concept; (3) a novel traffic sign recognition taxonomy, which allows us to classify most of proposals in the literature; (4) a vehicle-to-infrastructure high level communication protocol draft, which can be extended in the future based on potential needs; and (5) reliable results collected from a simulation context and validated in a real environment.

1.2 Structure of the Thesis

The present document is organized in six chapters including the introduction. Chapter 2 makes reference to different traffic sign recognition methods proposed on the literature and classifies them using an original taxonomy based on the technology of each system. Chapter 3 describes the context where the project is developed and underlies the major factors that justify a different TSR solution. Chapter 4 presents the proposed architecture design in detail. Chapter 5 describes the experiences made to test the solution from different points of view, discusses the results obtained, and compares them with other solutions from the market. Finally chapter 6 presents the conclusions of the report and recommends future work considering a final product based on the solution that has been presented in this project.

Chapter 2.
State of the art review

Traffic sign detection and recognition (TSR) is receiving a lot of attention from Original Equipment Manufacturer (OEM). Companies and clients have understood the importance of the active safety systems and that traffic signs are a key factor in responding to this demand. A lot of work and research is being conducted to enhance traffic sign recognition capabilities by introducing new techniques and improving the current ones. This chapter analyses different approaches in the literature and tries to classify them in order to allow easy comparison.

2.1 A novel taxonomy

The importance given to TSR has been reflected on the literature. There are several proposals that bring traffic signs and panels information to the digital domain. Despite of the large amount of research works focusing on this topic, comparative and evaluation studies that attempt to group the proposals depending on technology usage are rare. This fact leads to the inability to quantify in a real working environment which proposal is better for the user.

In this chapter we will provide a basic taxonomy of the most popular technologies used or proposed for TSR implementation. They have been some excellent attempts to classify vision techniques and algorithms like [16] and [17], however in this document we propose to take a step back and analyze TSR as an abstract system.

When we consider the whole picture, a TSR system is composed not only by the element that is detecting and recognizing an object, but also the object itself and how its information gets propagated to the intended receivers. The most extended implementation is a camera installed in the vehicle interior or exterior that captures images, which are processed in real-time searching a known pattern, like a shape or a color. We define this technology as "Camera-based" system because the information is actually recorded by an optical instrument, which digitalizes an image that is propagated across the visible spectrum.

2.2 Camera-based systems.

The problem of geometric shape detection and "heavily constrained" image recognition might not be a big challenge for an engineer acquainted with the recent advances in computer vision. Traffic signs are rather simple objects in an age where a machine can pick up the movement of pupils when navigating a web page. However to apply the system on a real environment we must add additional variables that might highly influence the system performance and, hence, will definitely increase its complexity for a satisfactory result.

Road images will be acquired from a vehicle moving on the, often uneven, road surface with a variable speed. Road signs are frequently totally or partially occluded by other vehicles or objects like trees, lamp poles or other signs. Many other objects are also present in the scene, like pedestrians, bicycles, billboards with a similar shape and colors, etc. which make the sign detection hard. One example could be the one shown in figure 1a.

The camera-based technology has been extensively studied in the literature and it is definitely the most developed one among our classification. There are a lot of attributes that we can evaluate for categorization: the sort of stream, which could be a single picture [25] or a video stream [24]; the color segmentation, for example RGB [23] or YCbCr [1]; the different elements that handle algorithm solve load, for example a dedicated GPU [19] or an external server [12]; etc.

However since the primary motivation of this section is just to provide a general idea of the different approaches to the sign recognition problem, we are not going to explain each of them in detailed, but provide a general idea of the most common aspects studied in the literature. The three methods here described have been proposed as stand-alone algorithms, but more and more frequently we can find publications were the combination of different methods is explored.

2.2.1 Color-based segmentation

This technique is very popular due to the fact that the colors of a traffic signs are usually easy to distinguish from the environment. For that reason a high color resolution is not mandatory for a good system performance. The biggest disadvantage is that external factors like illumination and weather conditions can significantly reduce the ability to detect as it is studied by Benallal and Meunier in [23].

2.2.2 Shape-based detection

This technique has also been broadly studied in the literature. The basic idea is trying to find an arbitrary or pre-defined shape in an image. Systems that implement this method benefit from the fact that traffic signs are well-defined shapes within the picture; hence edges must be easily identified. Once the object is detached from the whole picture, an algorithm tries to recognize it. The definition method varies from one proposal to another. For example on [20] the algorithm defines the center of the object, on [22] it will try to define a regular polygon within the object and on [21] it searches a template from a database in the image. This technique performance is also highly influenced by illumination changes, shadows and weather conditions.

2.2.3 Based on machine learning

Unlike the previous two sub-categories, this method does not only rely on previous patterns, the expected color or shape, but is also able to discover new patterns using machine learning. Viola and Jons proposed a very robust solution against noise and low-quality images in [18]. It seems to be one of the most reliable solutions, and it has also been enhanced by later researches, however the computational requirements and complexity might be the highest compared with other techniques.

2.2.4 Camera approach evolution and limitations

Despite the fact that the system has been extensively developed and there is already a wide range of different products in the market, there are some obstacles still to be overcome. The first car company that offered this technology was *BMW* in 2008, followed by *Daimler* the following year. The system was able to recognize only speed limits signs that were compliant with the Vienna Convention. The information retrieved from the camera was always verified against a navigation system database.

Opel introduced the second generation of camera-based systems. The system was able to work using vision-only information and included overtaking ban signs recognition. OEMs like *Ford*, *Volvo*, *Volkswagen* or *Saab* offer today TSR systems, which are able to recognize and interpret a few more traffic signs, on mid-range vehicles with high detection accuracy.

In the literature [19], [20], [21] and [22], researches go further aiming detection of all other types of traffic signs like stop, dead end, railroad crossing, turn regulation etc. Their results are promising and we might soon get vehicles that are able to detect these types of signs. Nevertheless, in my humble opinion, they all share an ultimate goal, which is to detect and recognize traffic signs as a human being.

We should not fail to appreciate such ambitious target, but also cannot ignore that the human eyesight has its own limitations and it is just an element that we use to understand the road information and environment. Other elements like the driver instinct to foresee unsafe situations, or the ability to ignore ambiguous data are more difficult replicate into a computer based on the picture or a sequence of pictures.

At this stage we should reconsider whether exchanging traffic signs information in the traditional way is the best option in a scenario where technology is getting a more and more active role. In the case of camera-based systems we have to capture, store and process pictures that are a few Kbytes large, with the only intention of extracting data that can be codified using a couple of bytes. Furthermore this information is always the same; hence the image-decode algorithm is applied several times to the same image (every time a vehicle approaches to the traffic sign). Consequently, although a camera-based method is the instinctive solution to traffic sign recognition, it worth evaluating a system where the information exchange can be completed without image decoding. The next categories present two different pre-shared code systems where image decoding is not required.

2.3 Wireless communication applied to TSR

The name of this category might not seem an accurate delimiter if we consider that information transmitted through the visible spectrum does not require a wired connection either. However the kind of technology that we would like to group under this name is only related with sort of communications designed to transfer information altering some form of energy, mainly radio waves. Also both transmitter and receptor need to be designed upon the same protocol in order to be able to establish the communication link and exchange intelligible information.

A machine is not limited to the visible spectrum as the human is; hence it is coherent that a more efficient band is used to exchange information. The main idea is to provide the traffic sign with an asset that makes it able to broadcast the information that is visually available, through a wireless protocol.

Compared to the previous category, this subject does not usually appear in the literature as a case of study for traffic sign recognition by its own, but integrated in the vehicular communications matter. It is not easy to find technical details about the few real cases that have been implemented on real applications.

One of the reasons why this technology has not been as extensively studied as the camera-based systems is that wireless communications needs at least in two elements from the system to be developed: environment and vehicle receptor; while in the camera-based case only the receptor needs to be adapted. This fact has an important influence on real implementations because the vehicle owner covers the entire cost.

This is an critical factor that must underlie every decision while developing a wireless communication system. Unlike camera-based implementation, it is essential that the highest number of drivers is able to benefit from the system, thus the traffic sign adaptation worth the investment. Also the cost per traffic sign should not be unaffordable in order to cover the largest distance and therefore a higher number of drivers.

There have been different approaches to provide traffic signs with wireless technology assets. In the following paragraphs an overview about the different technologies proposed by different researches is presented.

2.3.1 Radio-frequency identification (RFID).

Among the different wireless protocols, RFID is probably the most extended in the literature. There are two main subcategories depending on the power source, passive and active. In either of them, the system is composed by two types of tags: transponders and beacons. The basic idea is that the beacon emits signals at a pre-set interval. The transponders "wake up" when they receive a radio signal from a reader and by transmitting a signal back. In the passive case transponders are usually called tags, and they do not require a dedicated power supply, using the transmitted signal to power on and reflect energy back to the reader.

One of the first examples in the literature is the field test carried out by Yoshimi Sato and Koji Makanae [8]. They took the opportunity that camera-based systems were not yet extended in Japan and proposed a different solution: equipping the road with general-purpose RFID tags that contains sign information. They overcame the extra cost problem of replacing existing traffic sign because passive RFID tags are inexpensive. In addition RFID is a proven technology in transportation systems. Many countries have already adapted their toll electronic collection using RFID technology, also adaptive traffic lights is controlled in some cities by the same kind of system.

The main issue observed by other researches about Sato's work is the position of the RFID tags and the antenna. The communication range of the type of RFID devices used by Sato is approximately 40 cm. Therefore the antenna was placed in the rear part of the car and close to the floor. Considering that a mid-size sedan is roughly 1.8 meters and the width of the traffic lane is 3.5 meters in Spain [14], the driver could easily ride the car out of the range of the antenna.

In [9], the research group proposed a RFID system that allows readers to be mounted at better location in vehicle and the tags attached directly to the traffic sign. The enhancement is critical, since the range achieved is approximately 30 meters. The battery life as per the supplier specification is three to five years. Although this proposal is a bit more expensive still a reasonable cost at the infrastructure side.

2.3.2 Bluetooth.

Another short-wavelength proposal is the one presented by Bohonos [10]. Although it is not exactly a vehicle application, an important concept is exposed in this paper: data beaconing using smartphones. He proposed Bluetooth as an ideal solution for location-aware information broadcasting. He implements a model that is able to assist the blind collecting information from the urban environment, for example from the traffic light real-time information.

However Bluetooth has a major restriction in the vehicle context: its maximum communication range, which is approximately 10 meters. This fact leaves it behind other approaches like the use of active RFID tags, which provides a range of 30 meters [9]. We cannot overlook the benefit of using a smartphone instead of a dedicated transceiver. Nowadays almost every citizen owns a portable device with Bluetooth-enabled, most of them smartphones. In Spain, at the end of 2013, 95,6% of the households had a contract with a mobile phone operator. It demonstrates that Bohonos' proposal can reach a higher number of potential users than previous RFID systems like [8] and [9]. Although a Bluetooth-based solution can reach a higher number of potential users, the technology cannot be adapted to the vehicle/road context due to its short communication range.

On the other hand, RFID is becoming more popular on the latest smartphone generations, so some could think about using this technology on the TSR context. However smartphones are only equipped, so far, with Near-Field Communication (NFC) technology, a short range variant of the Radio-frequency identification, which unlike the active RFID devices proposed in [9], it does transmit on the ISM band of 13.56 MHz, with a maximum communication range of 3 meters.

2.3.3 Wireless Access in Vehicular Environments (WAVE)

WAVE, also known as IEEE P1609, is a relatively new standard designed for IEE 802.11 devices to operate in the Dedicated Short Range Communications (DSRC) band. The idea is to provide a wireless communication protocol for automotive use. WAVE is based on an amended data link and physical standards that offers a high performance multi-channel communication for multiple application types.

The 802.11p standard draws on most of the IEEE 802.11a, IEEE 802.11e and IEEE 802.11q characteristics. The amendment motivation is supporting quicker data exchange among fixed and mobile nodes within a range of one kilometer. To achieve a robust connection under high vehicle speeds, the physical layer relies on the band of 5.9 GHz (5.85 – 5.925 GHz), divided in 10MHz channels, half clocked mode compared to the

802.11a standard. As a result, parameters in the time domain are doubled and data rates are halved. The signal is then more robust; effects of Doppler spread are reduced because of reduced bandwidth.

The seven channels belong to two categories, control and service. Devices should continuously switch from service to monitor, and only stay in monitor channel if there is no on-going communication. Also two channels are used solely for public safety applications, which means that they can only be used for messages with a certain priority or higher. Priority messages are also allowed to use the maximum allowable Effective Isotropic Radiated Power (EIRP) is raised to 33 dBM, which makes the maximum distance to be achieved.

Unlike 802.11a/b/g/n, the MAC services: SCAN, JOIN, ASSOCIATE and AUTHENTICATE are not applicable on the 802.11p standard, dedicated control channels for beaconing have replaced all such operations.

The 802.11p protocol is intended to be the standard in vehicular communication systems. 802.11p has also been adopted by the Communications Access for Land Mobile (CALM) M5 specification, which ensures that European and North American vehicular communication systems will be compatible at a data link layer.

The U.S. Department of Transportation (DOT) along with the University of Michigan Transportation Research Institute (UMTRI) has launched and ambitious pilot program that will involve 2850 vehicles during a 30-month period [7]. Traffic signs and vehicles have been equipped with specific technology that is able to manage communication on the 802.11p standard. Drivers can volunteer to participate in the program and all necessary equipment gets installed at the university and maintained for one year.

This solution overcomes the distance problem and also introduces the benefit of vehicle-to-vehicle communications (V2V). Nevertheless to the knowledge of the author there is still not overcome the cost issue of road/city adaptation, neither it has been studied whether an aftermarket product for the vehicles worth the market. Following Vandenberghe estimation [5], the new on-board transceiver units will cost about the same as a built-in car navigation system, between 1000 and 3000 Euros.

As it has been already mentioned, one pitfall of wireless communication systems applied to traffic sign detection and recognition is that the investment only worth if an important number of users can benefit from it. The problem is not easier if we consider V2V as part of the equation because even if the final user gets not only the possibility of receiving information from the road/city environment, but also from other vehicles, many vehicles are required to obtain an immediate positive impact as part of the driver experience.

2.4 Data mapped systems

Unlike previous underlined technologies, in data mapping the information associated with environment is not exchanged or captured while driving on a specific road or street. The entire set of information should be previously collected and stored on a location, that should be accessible by the client.

Similar to a cache, the environment data can be partially or fully copied to a convenient location, which can be used as primary source during the drive time. The information should be available during the drive time exchanging the vehicle data (position, direction, etc.) with the database.

This solution does not require a sophisticate method to acquire information from the environment. Nevertheless, there should be a system that efficiently collects accurate location data. The applications could be classified based on the technology used to collect this information.

2.4.1 Global Positioning System (GPS)

GPS is a space-based satellite navigation system owned by the United States government and opened for commercial and scientific applications in 1983. It provides location and time of a GPS receiver in the Earth.

The vehicle speed limit as part of the information provided along with the map and directions to destination is now common among most of navigation system suppliers. It provides the driver with more or less reliable indications of speed limit signs and although it is not real-time information, companies like *Garmin* provides updates every three months.

Although GPS could be the most extended option for outdoor location technology, the build-in GPS receiver from an average smartphone has an accuracy of about 15 meters for a moving object. Also there are some environments where tall buildings can obscure one or more satellites and impact the performance of this sort of technology. GPS only provides static location-based information, and not dynamic information such as the status of a traffic light.

GPS is not the only satellite navigation system, but it is the most popular one. Most of smartphones in the market are equipped with GPS receiver, and due to open systems like *Google Maps*, there are plenty of applications available for different smartphone operating systems that use the information. Another satellite system that has recently become popular is GLObal Navigation Satellite System or GLONASS. The system got restored on 2010 by the Russian Federal Space Agency and surprisingly it is becoming the best complementary to GPS. Those two systems working simultaneously provide better coverage and precision rather than just one of them. The number of smartphones that include both technologies has increased in the last years.

2.4.2 Radio-frequency identification (RFID)

Although this technology has been already outlined under the wireless communication category, in this section is presented as a solution to the position and direction data. The basic idea is that RFID tags broadcast an identifier with specific information for one place, for example the begging of a tunnel. The system uses this information to query a database and prompt the driver with the associated data to that particular tunnel. A new RFID tag will notify the system that the information is not longer valid.

Since the information from a RFID tag is not usually updated very frequently, the system relies on the database to provide up-to-date information based on the vehicle location. This technology has extensively used at indoors applications like airports or museums, however it has not so far been exploited in the vehicle navigation systems.

2.5 Hybrid traffic sign recognition

Although most of the examples in the literature study a stand-alone category from the ones underlined categories, it is very easy to find hybrid implementations in the market. Developers usually combine different technologies to overcome intrinsic disadvantages of a system.

As an example the method that García-Garrido proposed in [13] to overcome a common problem on camera-based systems. Their prototype had some difficulties to disregard traffic signs that belong to a freeway exit. This technology along with the GPS to provide primary source for vehicle trajectory creates a system that combines all three categories for a better result performance.

Nevertheless combining different technologies is not the most efficient solution because it increases the complexity degree of the system as well as the final product cost.

There are some other cases where a hybrid system can be implemented on an inexpensive hardware setup. As part of this work the free-of-charge application *aCoDriver* has been tested [26]. This application implements a hybrid system based on GPS data mapping and camera-based recognition. The only hardware infrastructure required is a smartphone equipped with an in-build camera, a GPS receiver and Internet connection.

Chapter 3.
Analysis & Foundation work

Most of the road sign detection and recognition systems underlined in the previous chapter require specific hardware to be installed in the vehicle to provide this functionality to the driver. We have also introduced the idea that camera-based systems have intrinsic restrictions and that wireless communication technologies can improve or enhance the TSR user experience. However camera-based systems are already present on the market and the introduction additional wireless technology will require a change in infrastructure.

In this chapter we analyze the benefits of wireless communications applied to TSR and some of the proposals about how to bring them into the world of active safety. We also introduce an original project that tries to close the gap between technology prerequisites on one side and the requirement of potential users to easy the infrastructure change on the other.

3.1 Maintainability, integrity and reliability

Previously we have defined different categories for the current applications and proposals on traffic sign detection and recognition subject. As we have already advanced, data mapping and camera-based are the two most extended solutions. In this section we will analyze these two systems from the maintainability and integrity point of view.

Maintainability applied to traffic signs information can be understood as the ability to capture live data from road or urban environments. The information from a traffic sign is assumed to be static and with rare updates, but nothing could be further from truth. Traffic direction, speed limit, etc. can frequently change without previous notifications. This is not a problem for camera-based systems, but it is for applications that use data maps. The navigation data provider usually updates road information every three-four months. It typically results in inaccurate information prompted to the driver.

Furthermore the information coming from a vertical road traffic sign does not represent the complete driver view. This sort of traffic information is just the fourth one in the priority list according to the circulation code [29]; above it we can find traffic lights, temporary signs and police indications. None of these dynamic sources of information could be understood by a data map system, as the information changes dynamically and most of the times does not match with the data recorded in a database. On camera-based systems the situation is not much better. Traffic lights are not usually covered by camera-based systems, temporary signs do not share colors with the regular ones and they are usually placed at ground level, which makes the detection much more difficult because the camera is not calibrated to detect objects at that level from the ground. Finally police indications will also be hardly understood by a vision system.

On the other hand, if the data is transmitted by a wireless device, the system is much more flexible in regards to the sort of information that can be exchanged. Wireless transceivers could be installed to temporary traffic signs and a policeman can hold it while he controls the traffic flow giving indications to the drivers through wireless media. This solution is much more beneficial in the case of traffic lights, as the final user will be able to receive not only the current illuminated color, but also the reminding time before the indication changes.

Regarding integrity applied to TSR we can analyze it by the amount of misleading information events registered by the system, like in the previous example where a navigation application warns the user based on out-of-date speed limit data. The degree of integrity on data mapped systems is based on two fundamentals: frequency on database updates and position information precision. The current situation of mobile Internet could make a smartphone-based system or any other portable device equipped with this technology able to be always connected to the Internet. However this fact does not completely overcome the integrity issue because to the knowledge of the author there is not an official and frequently updated record of traffic signs information that could be query at any time.

Camera-based systems also have a weakness in regards to integrity: detection of traffic signs from all different tracks available on the visual range. A frequent example is the freeway service road; the application can prompt the driver with gradually lower speed limits while driving on a real 120kph freeway. This sort of behavior has been proved to be an important factor on chapter 5 when some deployed solutions, like [25] and [26], are empirically tested.

Finally, another important issue is the total latency, which should always remain below the limit established by the vehicle speed and the information availability. The information should be always prompted before the driver overtakes the traffic sign. Latter prompted information could impact the integrity degree of the system. It might not represent a problem for data mapped systems that get continuously updates about vehicle location or real-time camera systems, however can have a major influence on the reliability of wireless systems.

3.2 Wireless technology adoption

Despite the fact that the development lifecycle for passenger and commercial vehicles has been reduced up to the current five years, it is still far from other products like smartphones, which are fully functional after five months from the first development steps.

Furthermore the average vehicle replacement time in Spain is 9.5 years, which translates that although OEM starts developing an affordable built-in technology for wireless communication applied to traffic sign recognition today, it will not be available on half of the vehicles in circulation until 2030. So, regardless any other motivation, if we assume that 50% of the fleet is an authoritative figure for the infrastructure investment, we need 15 years more to benefit from the advantages of the wireless technology applied to traffic sign recognition.

The need of a substantial embracement of this technology for it to become realistic target might be the main barrier to start benefiting from its potential. In order to break this “chicken-and-egg” situation, we propose an alternative platform that implements Human-machine interface and wireless communications.

The idea proposed in this document is to modify traffic infrastructure for a small and critical area of the city at low cost (further in this chapter we will present a rough calculation), in order to attract some potential users. Furthermore, we will offer the final user a solution that can be adopted at a low cost. The objective is to quickly reach a significant number of users that can justify a more important investment to cover a bigger area.

An aftermarket product normally follows a built-in feature that has already become popular among drivers. Moreover the initial market price for new aftermarket technology usually exceeds what an average driver is willing to pay for an add-on. On the other hand, smartphones have become a commodity of widespread adoption, and most of current models offer a complete set of wireless interfaces: NTC, Bluetooth, Wi-Fi, LTE (4G), etc. Hence we propose the smartphone as an excellent substitute platform that can easily bring the user into the wireless communications applied to TSR.

There are some examples in the literature that have addressed the use of smartphones on TSR from different perspectives. One example is [12], which proposes a camera-based system using the build-in camera from the smartphone. The smartphone captures video and then extracts the images at a certain frame-rate, and then these pictures are transmitted to the car computer, which will process them and interact with the user. Wireless assets are here used for in-vehicle computer and smartphone data exchange.

In the previous chapter we have also cited some other examples where UHF radio waves, like Bluetooth [10] or RFID [9], were used to get location-aware information. A mid-range smartphone is also equipped with the technology to exploit this sort of wireless communication. However its short-range capability makes this technology not the most appropriate one for a vehicle environment, where high degree of mobility and short time responses are critical targets. For this reason the initiative of using this technology on smartphones has been generally discarded. Moreover since the 802.11p standard was adopted by most of entities working on the connected vehicle concept.

In [5] and [6], the idea of smartphone on Ad-hoc networks applied to the connected car concept is addressed. They both extend the idea of the 802.11p standard to the smartphone domain. The main issue is that current network adapters on smartphones do not implement the physical and medium access layers standards. Choi studied in [6] the possibility of attaching a RF front-end module to the device that could provide 802.11p capabilities. The results show that the standard could be implemented within the stringent power and area budgets of a smartphone.

On the other hand, Vandenberghe analyzed the idea of implementing an Ad-hoc network using 802.11a or 802.11g protocols instead [5]. The intention is making the smartphone a "mid-way solution" between today's technology and the major updates that the vehicle fleet requires to provide an appropriate scenario for the connected car. Although the conclusions show that either protocol is not delivering as good results as

802.11p does, it is an important step forward, and could pave the way to the adoption of wireless vehicular communication on road and safety applications.

Nevertheless there is still a major constrain on smartphones, which these two papers do not reflect. Ad-hoc networks, like the ones based on the 802.11p standard, are the best approach for situations where high degree of mobility should be supported. However, *Google* official *Android* distributions are not compatible with Ad-hoc networks where there is no infrastructure set up or network administrator. Neither the other major operation system in the market, *iOS* [30], is an option, as its official distribution does not fully support this sort of decentralized network topologies.

This situation does not seem to be different in the near future. Each company has already developed its own customized solution to offer peer-to-peer wireless communications based on Wi-Fi standards. For example *Android* implements the Wi-Fi P2P specification under the name of "*Wifi Direct*", which basically is a traditional infrastructure network, where the Access Point role is negotiated among nodes before communication starts.

The P2P standard is supported by most of new smartphones and might be an interesting case of study on vehicle-to-vehicle (V2V) communications. Although this is not the main topic of the present document, the code included in this project has been prepared to be compatible with *Wifi-Direct* just implementing a minor modification.

3.3 The low cost approach

Previous sections have revealed the benefits of wireless technologies adapted to traffic sign detection and recognition. We have also uncovered the requirement of a large number of users to get the system implementation feasible in a real environment. These two circumstances make the smartphone a perfect platform to attract potential users due to high availability with no additional cost.

Most of the smartphones in the market come with a complete asset of wireless technologies, however we will need to use a standard that fulfills requirements from a road environment: long-range communications and high degree of mobility. Although active RFID or 802.11p are the most appropriate standards that we have mentioned in this document, they are not implemented on mid-range smartphones. Also we consider that the use of add-on peripherals deviates the implementation from its original purpose and turn away potential users.

Despite its limitations, we propose a traditional Wi-Fi infrastructure topology as a substitute for other specific protocols. The objective of this project is studding TSR performance of a real smartphone implementation in order to evaluate its feasibility in road environments. It is based on the idea that equipping traffic signs and vehicles with wireless technology at a low cost will pave the way to more efficient solutions that involve a more important investment.

3.4 The most extended Human Machine Interface

Most of solutions proposed on the previous chapter require specific technology and customized protocol standards. This approach achieves normally better final results than trying to implement a similar idea using general-purpose devices. However they are also more expensive solutions because general-purpose products are generally cheaper and can a reach a larger public.

In our particular case, we will take the advantage of a widespread platform that offers the general public the opportunity to get into the system without spending money on additional hardware. That brings another significant benefit: the final user is already familiar with the Human Machine Interface, which makes possible an earlier adoption.

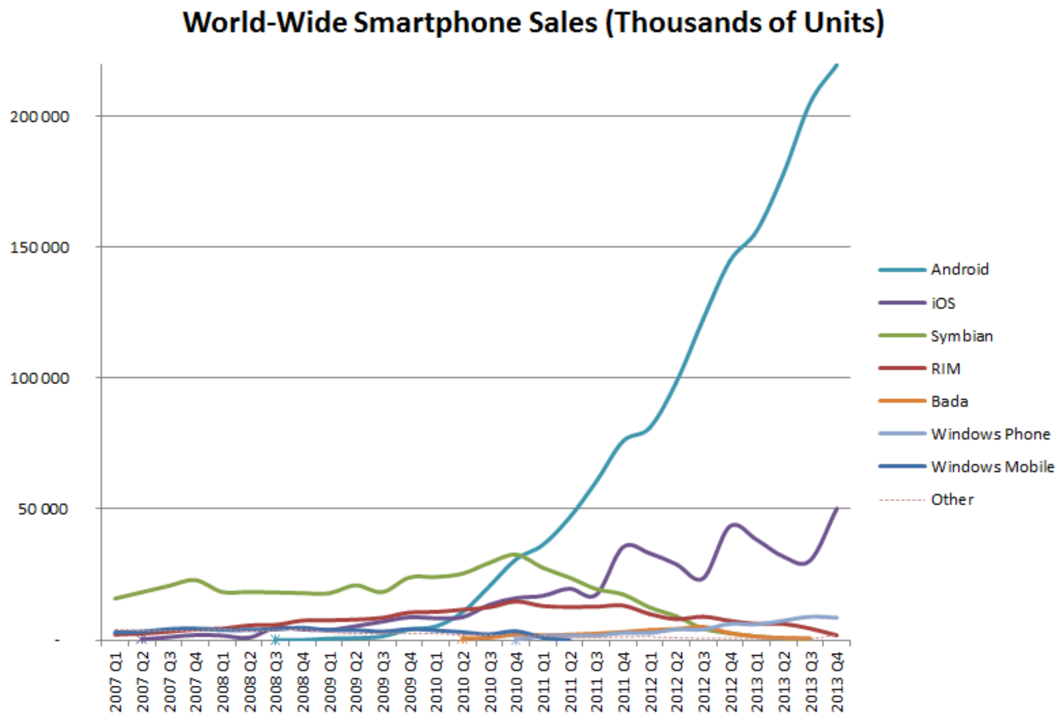


Figure 2: World-wide smartphone sales - 4Q13 update.

Because of the intention of this project is to be able to reach as many potential users as possible, we have based our decision about which mobile operating system to use on the number of users. In 2014, more than a billion smartphones were sold and global market share was 80.7% for *Android* [30]. Hence *Android* is the best mobile operating system option for an application that has the objective of reaching the highest number of users in a short period of time.

Android is a mobile operating system based on the *Linux* kernel and currently developed by *Google*. Its design is lead by the aim of providing a user-friendly interface for primarily touchscreen devices.

Google distributes *Android* under open source licenses, although most of devices ultimately ship with a combination of open source and proprietary software. On top of the *Linux* kernel, we can find middleware, libraries and finally the *Android* APIs.

3.5 A general-purpose transceiver.

Following the same reasons we want to provide a low cost equipment that allow us a certain flexibility for implementing wireless communications on the traffic sign side. The objective is to develop a network interface with the following characteristics:

- High compatibility with other standards. We should bear in mind that the proposal from this document is a temporary solution to attract potential users. Assuming the proposal become popular and easy the access to an infrastructure extension and an aftermarket cost-effective product, we will then revisit the complete design to improve the user experience. That might require other standards or technology to enroll the game.
- Quick reaction to data updates. In order to provide a good maintainability and integrity, digital traffic signs should quickly react to changes on the environment information. We propose an optional secondary network interface that will allow full control over the information sent to the driver at any time. Furthermore each sign is labeled with a priority class. That allows temporary traffic signs or police to broadcast indications to the driver that will override any other information.
- Live updates. Following the machine learning approach on camera-based systems, we want to allow the client not only recognize previously

downloaded traffic sign models, but also others that are not stored in the local database. For this purpose, we should be able to support different kinds of information exchange upon the client request.

- Low cost solution. One more time we remark the importance of staying at a low cost implementation to cover the biggest area and therefore the highest number of potential users.

Based on these four fundamentals, we have opted for a general-purpose board and an external wireless interface, which connects to the board by a USB port. Among the products that offer these capabilities in the market, we have chosen the *Raspberry Pi*. Its wide range of connectors along with the fact that it uses Linux-kernel-based operating systems makes it a versatile and robust solution for our objective.

The price was also a key-factor on this election; the basic pack (board + wireless USB adapter) does not exceed the 30 Euros per traffic sign. The price increase slightly when we think about solutions where the board interacts with the environment beyond the wireless communications, for example traffic lights control, alarms sounding for pedestrians, etc.

Although this sort of general-purpose embedded system offers a wide range of possibilities, we have focused this document on the use of network communications. Hence apart from the wireless USB adapter, we will only exploit the on-board Ethernet adapter, which will be used for software and data updates.

3.6 Benefits of using widespread standards and technology

As it has been previously exposed on this chapter, the selection criteria followed on each step of this project is based on applying well-proven technologies, highly developed standards and popular platforms. The main reason is that the system can reach as many potential users as possible. However the high availability is not the only benefit.

Although the first *Android* device was developed seven years ago, *Google* has already released six major versions of operation system and it is an on-going project. Our client application not only has a public of 80% of the mobile device users, but also will benefit from potential improvements on the operating system and hardware enhancements coming with future devices.

On the other hand although our initial digital sign implementation is presented on a low cost hardware platform, it is deployed on top of a *Linux* distribution (*Raspbian*) and therefore compatible with most of *Linux*-based platforms.

The transport layer has also been chosen upon this principle. TCP/IP standard is implemented using well-proven libraries on each side of the system. At the application layer we have used popular programming languages with a very helpful community behind. Although *Android* systems are more and more supporting different programming languages, the most efficient one is still Java. On the other side, we have used a Python script for the digital sign implementation. Python is cross operating system developed, which means that our application only requires the Python interpreter to run on a different platform like a *Windows* machine.

All these decisions and others that will be described on the next chapter make our system able to adapt to potential enhancements automatically or with a very small effort, which reinforces the idea of a transition solution towards a standard vehicle to infrastructure technology.

Chapter 4.
Design & Development methodology

The previous chapters have emphasized the importance of a general public solution for the reliability of wireless digital signs. We have not been able to find a product in the market that fulfils the requirements to allow users start benefiting today from wireless technologies applied to traffic sign recognition. The motivation of this project is the one of understanding the reliability and limitations of using a widespread platform, such as the smartphone, to implement a low-cost and efficient traffic sign recognition device. For this purpose we have developed an *Android* application that allows the driver exchanging information from a customized digital traffic post.

In this chapter we will first explain the server solution that we propose to commit with the low-cost approach. The model that we have designed is platform and operating system independent, however for the implementation we have mainly used open source software and inexpensive hardware.

Later it is described how we achieved an efficient *Android*-based solution at the application level without using additional hardware. The main idea we followed was to provide the potential user with a ready-to-use application, which does not disturb other processes running on the smartphone, e.g. in-coming calls, instant message services, etc.

4.1 A wireless digital traffic sign model.

The communication system we have designed is based on a server-client model, where the road information is normally owned and distributed by the server. The server main purpose is providing a particular service to more than one client at the same time. On the other hand, the client shall explicitly request a connection to establish the communication and make use of the service.

In this document we propose to apply the server-client model at a traffic sign level, in other words, each traffic sign or post acts as the server that supplies a service to its clients, the vehicles. The idea is that the information displayed on the physical traffic sign should be always available for the TSR clients over a wireless connection.

With this intention each traffic sign is equipped with a piece of hardware, which main component is a Wireless LAN (WLAN) network adapter. For the prototype we are using a single-chip USB network interface, which is included in most of *Raspberry* developer kits. The operating systems that the *Raspberry Foundation* distributes integrate drivers to support this USB wireless adapter. However we need additional software packages to implement the connection manager, authenticators, etc. *Hostapd* and *dnsmasq Linux* libraries will be installed to accomplish this function.

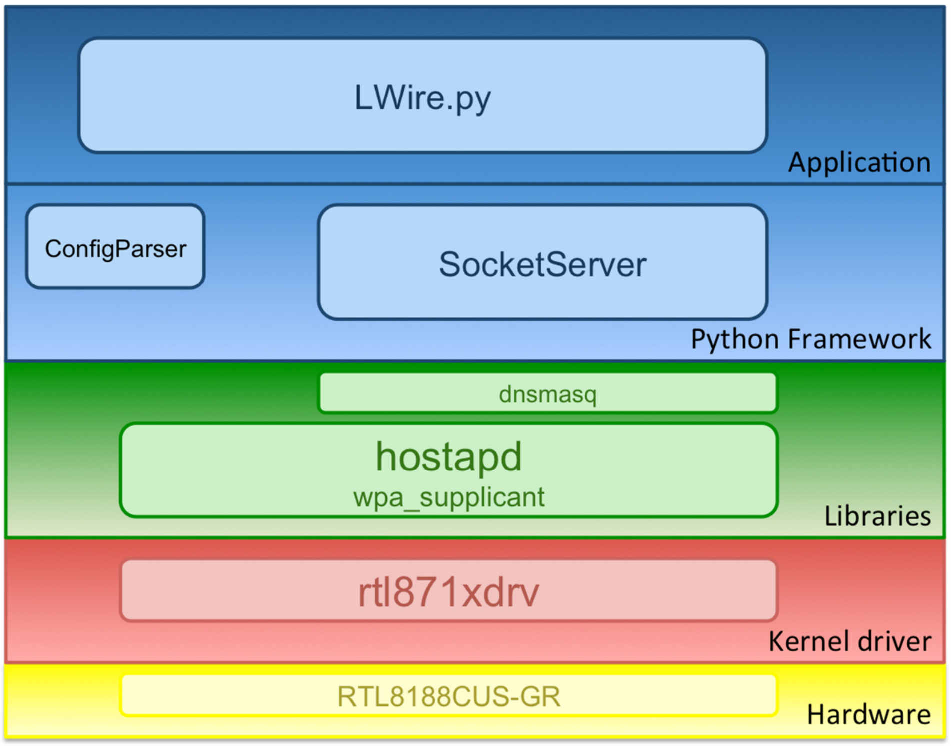


Figure 3: Simplified Server Communication Stack Proposal.

The fundamental idea is to setup each *Raspberry Pi* as an IEEE 802.11 access point and run a daemon application that offers different sort of road information depending on what the client requests. The communication between server and client applications is established based on the TCP/IP standard, which is implemented by the *SocketServer* package at the server side. Figure 3 shows a simplified communication stack of the server proposed.

LWire is the *Python* application that we use as a daemon to serve client information requests. Apart from the *SorcketServer* Package, it imports the *ConfigParser* class that reads the traffic sign specific information from the *station.ini* file stored on each *Raspberry Pi*.

4.1.1 The Raspberry Pi

On the previous chapter we advanced that a general-purpose hardware solution will be used to equipped traffic signs with wireless interface. We chose *Raspberry Pi* because of its price and versatility. It provides the benefits of a well-proved operating system on a single board prepared with plenty of interface possibilities, including analog and digital input/output pins.

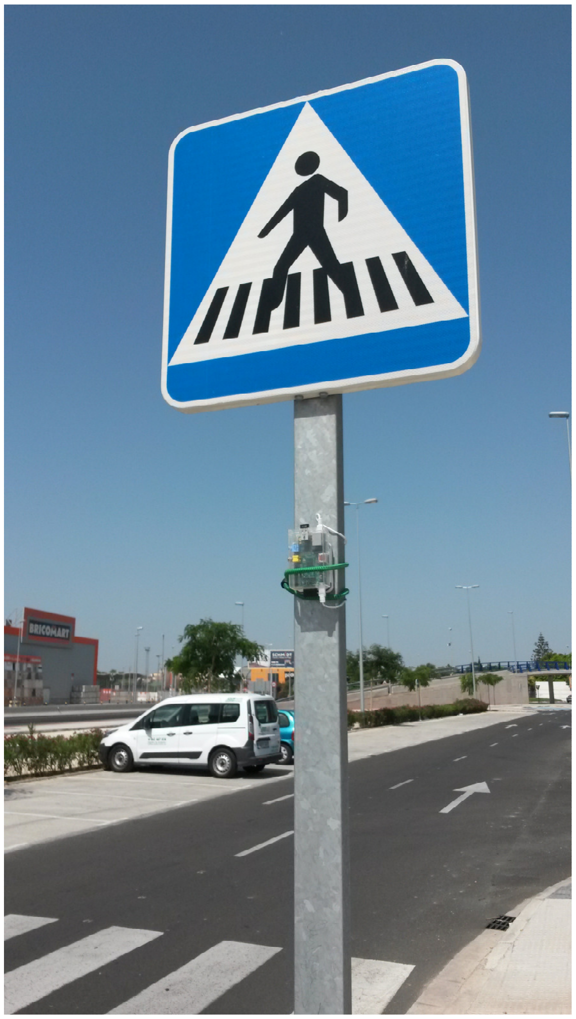


Figure 4: Traffic sign equipped with 802.11 wireless interface

Raspberry Pi has been manufactured in several board configurations. In our case, we will use the model B, which is based on the *Broadcom BCM 2835* system on a chip

(SoC). It includes an *ARM1176JZF-S* 700 MHz processor and 512 megabyte of RAM. The board is shipped with a Secure Digital (SD) socket for boot media and persistent storage. This is a benefit for the distribution of our application since we do not need the actual board to load the software.

This model of SoC from *Broadcom* does not support native Wi-Fi. However as the drawing on figure 5 shows, there are two USB 2.0 ports available that we will use to equip the board with a Wi-Fi interface. There is an additional USB port attached to the hub that is used by an Ethernet adapter. We propose to use this LAN interface to update software and calibration files, leaving the WLAN fully dedicated to TSR.

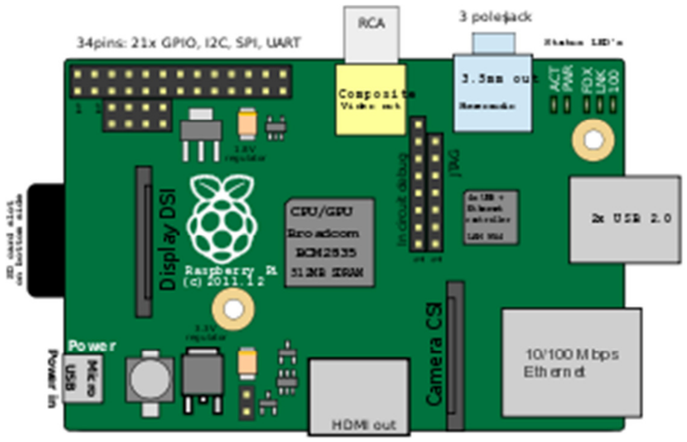


Figure 5: Drawing of *Raspberry Pi* model B rev2 [31]

Although *Raspberry* has not been designed for industrial or urban projects, it fits our requirements for a hardware platform prototype: mobile, versatile and inexpensive. However Raspberry Pi is not the only product that can provide these characteristics, there are alternatives in the market that we will not analyze in this document; some examples are *Galileo 2*, *Beaglebone*, *Banana Pi* or *Odroid-U3.* All these microcomputers run *Linux* distributions among other operating systems, which makes them fully compatible with the solution proposed in this section.

4.1.2 *Hostapd*

Hostapd is a daemon program distributed under the BSD license for access points and authentication servers. It is designed to run in the background and acts as the backend component controlling access and authentication. The package distribution that we can get from the official *Linux* repository, implements IEEE 802.11 access point management, IEEE 802.1X/WPA/WPA2/EAP Authenticators, RADIUS client, EAP server, and RADIUS authentication server.

The design goal for *hostapd* was to provide a hardware, driver, and OS independent, solution for all WPA functionality. The basic idea is to implement a driver wrapper that parses the operating system calls to the driver for controlling WLAN devices and the other way around. All hardware and driver dependent functionalities are implemented in separate C files that can be replaced or updated with upcoming device drivers.

Although this package is the most widespread solution for IEEE 802.11 wireless access point implementations on *Linux*, the official distribution is not updated with some of the wireless dongles available in the market, and the one chosen for this project, *RTL8188CUS-GR*, is not an exception. Fortunately the device manufacturer, *Realtek*, maintains a customized version of *Hostapd* on its website which is compatible with this specific chipset. Therefore we just need to download the driver package and compile the file to be able to execute the program on the *Raspberry Pi*.

Hostapd uses a configuration file that offers the user different setup options regarding authentication, encryption, signal strength, etc. We have chosen a rather common configuration, as we understand the reliability of wireless data exchange relaying on the 802.11 standard.

The Appendix I explains how to compile and setup the downloaded file on a *Raspberry* in order to use it as traffic sign wireless interface.

4.1.3 *Dnsmasq*

Hostapd provides the system with a full 802.11 access point management, however we will need an additional packet to identify the clients within the network. Since the identity of the clients is unknown at this stage we cannot rely on static Internet Protocol (IP) addresses. A mechanism should be implemented to handle dynamic addresses for each client that aims to exchange information with the access point.

Dnsmasq is a free software package that provides Domain Name System (DNS) and Dynamic Host Configuration Protocol (DHCP) services for small networks. Although we are only interested on the second service, DHCP, this software is designed to be lightweight and have a small footprint; hence it is a good solution for resource-constrained devices.

Regarding the size of the network, we can expect a reduced number of clients connected at the same time because the connection range, as we will see on the following chapter, is not greater than 30/50 meters. The mid-size sedan is roughly 4.5 meters long and less than 1.8 meters width, hence the density of nodes cannot be very high considering one device per vehicle. Nevertheless, as any other part of the model, there is always the possibility of replacing the piece of software in the future if required.

Like *Hostapd*, this package is not included in the *Linux* distributions offered by the *Raspberry Pi* foundation. The installation and setup process is also described in the appendix I of this document.

4.1.4 *LWire*

The highest communication layer of the server is basically a daemon that attends any request from the client, if it is compliant with the protocol we propose. The communication is established over the TCP/IP standard, which is implemented by the *SocketServer Python* library.

As we have previously mentioned, the *Raspberry* foundation provides different *Linux* distributions adapted to get the best performance on a *Raspberry* board. These distributions come with tools for *Python* as the main programming language. *Python* is operating system independent and there is an extensive collection of high quality libraries, like the *SocketServer*, for helping to solve almost any programming task on *Raspberry*. These are the reason for choosing *Python* for the high-level application of our model.

The task manager, as part of the *rc.local* file description, launches the *LWire* right after the operating system has completed the boot up process. The application will then load the *station.ini* file, which contains all specific information about the traffic sign. The appendix II describes the content of this file.

Once the configuration file is parsed, the application starts serving requests until the *Raspberry* shuts down or an administrator user terminates the process. The current application version supports three different requests; nevertheless the objective is to extend this functionality upon different needs of the system.

The client first requests the data related with the visual information of the traffic sign. This is a four characters symbol that in our example is based on the codification that is proposed by the circulation code [29]. More than one symbol could be expressed by a unique response if the '\1' character joins them. The string should always end with a '\0' character.

Additionally the client has the possibility of requesting extra information by sending a second request. The information received may or may not be related with the traffic sign, but it is always with its location. For example, if a certain street is scheduled to be inaccessible the following day because of a social event, the driver will have a notification with the information available on his smartphone once he arrives to the destination. Thus, the driver is getting additional location-aware information without increasing the cost of the system.

Finally, applying a similar concept as the one proposed by Viola on [18], our system should be able to react to unknown traffic symbols by downloading the complete set of data from the station.ini file. This feature is handled by a special request, which requires some more time for the transmission because of the amount of data to be transmitted. The objective is providing the system with a method to "learn" unknown traffic signs that do not belong to the database until this request is completed. This method is not intended to be the usual process for updating the smartphone database; the client will receive periodic software updates when the phone is connected to the Internet.

The last two requests have a lower priority than the first one and are only served in a good network state. At any signs of data traffic congestion, low battery level, etc. the second and third request should be ignored.

The script receives its name in honor to the inventor of the electric traffic light. The original traffic lights, based on a semaphore system, gas lit, had been invented in London in 1868 with a very little success. Forty-four years later, Lester Wire, an American policeman, installed the first 4-way electrical traffic light in Detroit, Michigan, based on the railroad sign concept. He had observed that railroad traffic was automatically controlled and thought about adapting it for street use. Within a year, Detroit had installed a total of fifteen of the new automatic lights [34].

4.2 *Android* client implementation.

In this section we will describe the client implementation, which is based on the *Android* operating system. We propose in this chapter a minimal installation that should be compatible with any *Android* device equipped with a WLAN network interface.

Figure 6 shows the communication stack of the client solution we propose. One of the requirements we set at the beginning of the project was to avoid deviations from the typical installation process on the client side. The reason is that a complicated installation can dissuade potential users from adopting the system. Therefore we are limited to work only at the application layer. The rest of stack is expected to come preinstalled on the device to facilitate a WLAN interface. Only kernel driver and hardware may change from one device to another.

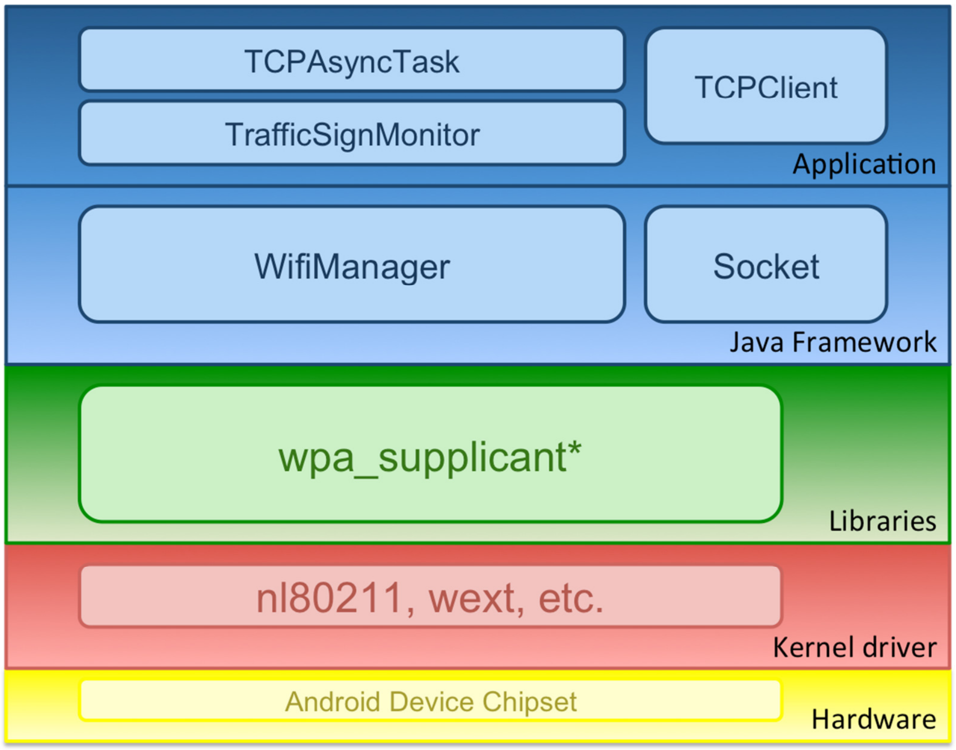


Figure 6: Simplified *Android* Client Communication Stack Proposal.

One of the reasons for the popularity of *Android* is its versatility to run on different hardware architectures and chipsets. In some devices we can find a dedicated chip for the radio stack, for example in one of the best sellers ever, the *HTC Dream*; while in others, a single chip is used containing both the processor for the radio stack and the processor(s) for the application layer. These differences are overcome on higher layers making the operating system easy to be installed on different types of devices.

First the kernel network drivers used on *Android* phones are typically common for several chipsets. For example *wext*, or its modern replacement *nl80211*, are used on most of *Linux*-based operating systems as a standard driver. However there are still chipsets not supported by these drivers. Again, taking the *HTC Dream* as an example, we can find a custom driver designed specifically for the *Texas Instruments* chipset.

At the user space we have the *wpa_supplicant*, which acts as a wrapper for the previously mentioned drivers. This supplicant is the standard *Linux* process responsible for the discovery of, and the authentication with access points. The only way for the *Android* applications to interact with the supplicant is through the *WifiManager* class from the *Android* Application Programming Interface (API).

At the application layer the Java class *TrafficSignMonitor* requests a sequence of services through the *WifiManager* to establish communication with the access points. In the interim it also monitors different events through a *broadcastReceiver*. This way, the application is always aware of the communication state. Once the connection is ready, a second thread is created to set up a TCP/IP connection that demands the information stored on the traffic sign. The *TCPClient* and *Socket* classes handle this process at the communication layer.

Beside the message, the client has also available the received signal strength (RSSI) of the signal, which can indicate the distance between the smartphone and the antenna. The RSSI value decreases proportionally when the vehicle approaches the traffic sign.

4.2.1 The *Android* application

Android applications aim to extend the functionality of the mobile devices. The devices are shipped with the operating system pre-installed as well as a whole set of API's, which allow developers to exploit the hardware capabilities of the device. The first decision we had to make before starting to develope our application was the programming language. We chose Java because *Android* API's have been developed in this language and it is the only language that provides complete access to them. With this purpose we have downloaded and installed the *Android* Software Development kit (SDK) in the laptop that we will use to develop and evaluate the system.

The *Android* operating system is also distributed with a process virtual machine (VM). In our case, the experiments will be performed on the *Android* version 4.4.2, also known as *KitKat*. This version and earlier ones come with *Dalvik*, a VM designed for *Android* under the conditions of battery constrains, low availability of machine resources, etc. Our application, as the others installed on the device, runs on its own process, with a dedicated instance of the *Dalvik* VM. The virtual machine will process the requests originated by our application and processed by middleware. We should also consider its performance when analyzing the CPU and memory usage data from the tests that will be conducted in the next chapter.

4.2.2 WifiManager

The *WifiManager* class is a java package that provides the primary API to handle Wi-Fi connectivity. As the only available option in *Android* for performing Wi-Fi specific operations, it will be our interface with the wireless adapter.

The *TrafficSignMonitor* service creates an instance of the class at an early stage of the runtime. The objective is interacting with the supplicant during the application lifecycle. One of the most important methods for our objective is *startScan*. This method requests a scan for access points and returns a Boolean result immediately; according to the to the *Android* reference website [35], the value represents whether the operation has been successfully initiated or not. The completion of the scan operation is made known later by means of an asynchronous event.

In order to receive this sort of notifications, we need to register a *BroadcastReceiver*, which will alert the application with any change on the network adapter. Other strategies to get information about the network state, for example a direct request to the API, do not appear to be efficient for several reasons: the main thread needs to be stalled while monitoring the application state, the results are not linked to any particular request, the request could delay the completion of the operation, etc.

As we will see on the next chapter, the *Android* API is perhaps the main limitation we have found when putting our proposal into practice. Probably one of the main disadvantages that might affect the performance and definitely increase the latency of the traffic sign detection is the 802.11 scan phase. *WifiManager* does only offer this one method, which is not overloaded and does not accept any argument. The *Android* Framework Layer receives the call from this method and executes the respective commands in the *Java Native Interface* (JNI) layer, which in turns transmits this to the *Hardware Abstraction layer* (HAL). An event is fired upon completion of the scan; the application then is able to retrieve the scan results form the *WifiManager* class.

From the *wpa_supplicant* documentation [38], we have learnt that every time the application calls the *startScan* method, the driver performs a full scan and tries first to find the known access points. Empirically we have concluded that the event is registered after the supplicant has not been able to find any known BSSID along all available channels. However if a known one is detected during the scanning phase, the connection process starts before the scan is completed. This observation made us reconsider the original implementation, where the connection attempt was initialized only after receiving the scan results event.

Another limitation we have observed while comparing the solution with other proposals is the wide range of channels. The *startScan* method requests a passive scan, where the client listens to the wireless medium for beacon. These frames contain the relevant information that can be used in the association phase with the Access Point (AP). The 802.11b/g standards define 13 channels in the 2.4GHz frequency range, spaced 5 MHz apart. The client must listen to the medium in all channels, for an amount of time greater or equal to the interval, which is typically 100 ms [37], but can be different. The number of channels and the maximum beacon interval on the AP determines the scan latency.

We should be able to modify each of these variables on the server solution based on our requirements. However, on the one hand the time that the client is listening on each channel is not setup at the application layer, but at the kernel driver. On the other hand chipset, firmware, kernel drivers, supplicant drivers and the supplicant itself all have support for scanning a specific channel or set of channels [36], but, as we mentioned before, the *startScan* method from the *WifiManager* class takes no arguments, and there is currently no way to submit a reduced list of channels from the application layer.

This is, indeed, an important inefficiency of our system compared to other solutions. For example the 802.11p standard offers two dedicated channels to discover other devices in the network [3], like traffic signs. While there is no on-going communication, the network device in the vehicle does only listen to those channels, which reduces significantly the discovery latency. Browers proposed something similar on [36] for the traditional infrastructure network based on the 802.11a/b/g/n standard. He suggested a change at the user-space level to overcome the *WifiManager* limitations.

This solution is not adopted in our system because the implementation goes beyond the application layer, however it is later mentioned in this document for further analysis. We consider that it is interesting to understand the side effects in regards to the device usability and performance of Browers' proposal if it is be applied to the vehicular environment and specifically to TSR.

4.2.3 Multi-thread application

We have mentioned that *Android* has been designed as an operating system for touchscreen devices; hence the importance of the user interfaces. The *activity* is the main class and the entry point for most of the Android applications. In our case the key piece of code is not the user interface, but the network monitor and interaction.

This fact would be a potential problem on our implementation if we intend to run the TSR routine in the same thread as the user interface (UI). The reason is that operations, like a TCP request, hold the thread and makes the user interaction unavailable until their completion. In the practice, the result is very often a system exception; hence we will avoid running any network interaction in the same thread as the user interface.

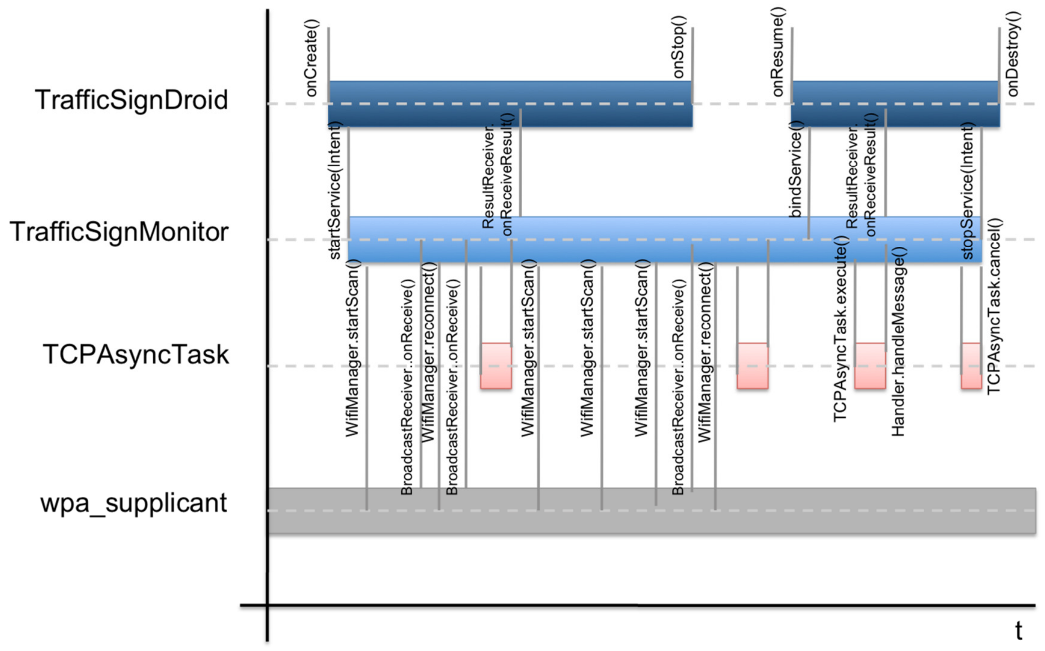


Figure 7: Example of TSR client application lifecycle.

We have chosen the *Android service* as an efficient solution to this issue. The interaction with the network API will not only run in a separate thread, but also will be independent from the user interface. The first time the activity is launched it will create the *service* and even if the user chooses to close de activity the *service* will keep running in the background until the user or the operating system explicitly requests to stop it.

Apart from that, we have also decided to manage the TCP/IP protocol on a separate thread. In this case the thread is created once the 802.11 connection has been completed, and destroyed when the client has received all the information requested to the server. The benefit is that the *Android service* can keep monitoring the network state while a separate thread exchanges information with the server. If an unexpected event occurs during the server interaction, the application might be able to decide the best strategy to close de connection based on the information registered by the event.

On figure 7 we can see a time diagram that illustrate the lifecycle of each thread and how they interact between each other.

4.2.4 The *TrafficSignMonitor* State Machine

As we have seen on the previous section, the implementation of our client proposal requires several processes and threads running in parallel. Thus, the events that could potentially trigger some routines on our application cannot only occur internally on the application itself. For instance the application needs to abort the scanning routine if the Wi-Fi adapter has been disabled. This event occurs at the kernel space and it is registered by a *broadcastReceiver*. Another example is the user interface, which is implemented in a different java class and runs in a separate thread.

It is clear that in order to reach a certain degree of efficiency in the code, we need to centralize all the asynchronous requests in a single process, which should able to control the flow of the sequence and dismissed unrelated events. Following this determination we have chosen the implementation of a finite-state machine (FSM).

The conception of the runtime sequence as a model made of states and transitions greatly simplifies the multithread challenge. In order to get the information from a wireless traffic sign the client should discover the sign, establish a connection, make a TCP/IP request and after the data is received it should handle the disconnection. Each of these steps should be taken in a sequence and one at a time. The different events are either asynchronously registered by the *broadcastReceiver*, a listener or even created by the main process itself, however all will ultimately be linked to a FSM transition. Hence, they are all processed by the same method, regardless the source of the request.

Figure 8 shows the FSM diagram implemented in the *Android* application, which covers the main workload of the client. After the service is started (*srv_start*) the state machine runs continuously unless an external request stops the service. The service were the FSM implemented is independent from the user interface, therefore the application will keep running although the graphic activity is destroyed or is sent to the background.

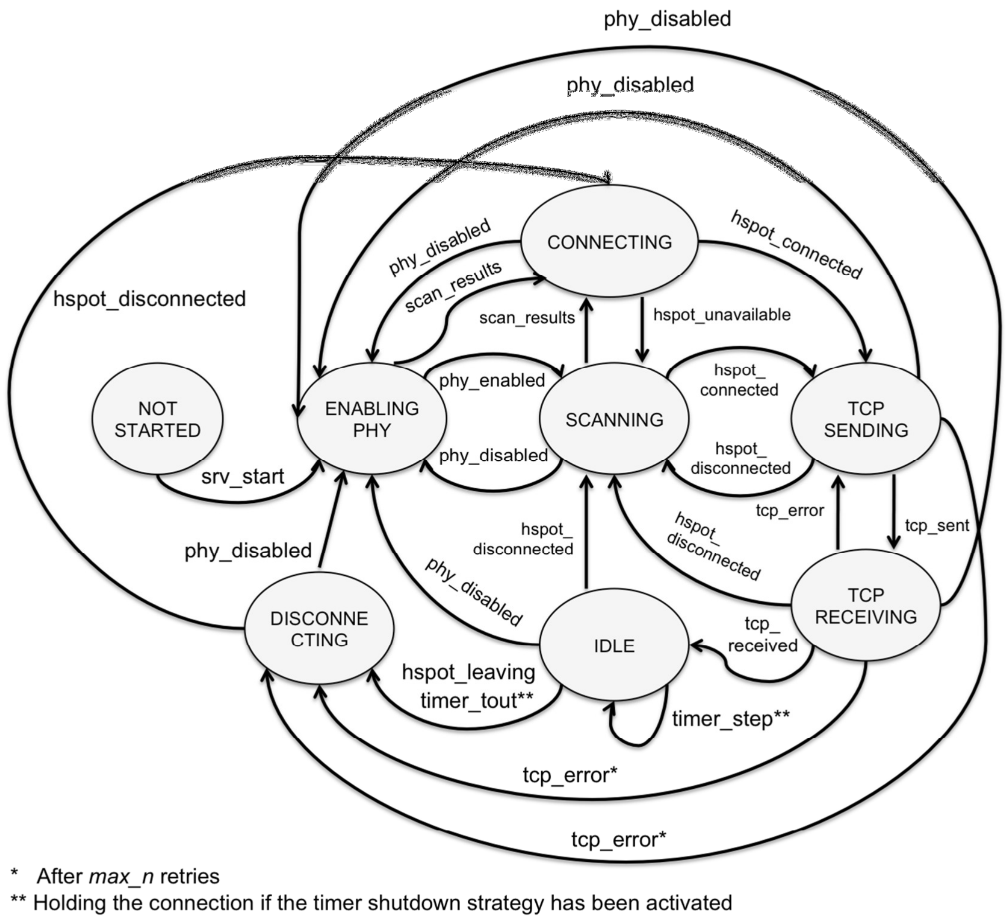


Figure 8: *TrafficSignMonitor* state machine diagram.

Another important benefit about the state machine is that we are adding one more abstract layer. This makes the application easier to implement to a different OS or API. Porting the code will be just a matter of linking the current transitions to different events and keeping the current states.

Chapter 5.
Experimental Analysis

The design and the implementation of the solution were explained in the previous chapters. We tested the proposal in a real environment to analyze the results and assess whether the system is a reliable option. This chapter presents the experimental analysis at three stages of the project. Each set of results obtained from the experiments has motivated changes on the implementation, which leaded us to the final product: an efficient and inexpensive solution for wireless technologies applied to traffic sign recognition.

In this chapter we will first describe the different setups we have used during the experiments. After that, it explains the evolution of the implementation based on the experience from the trials; and finally presents the results obtained with the latest version of the system implementation. In addition we have added a section where we perform an empirical analysis on other TSR solutions also implemented in smartphone platforms.

5.1 Experimental set-up

In order to test the solution we got hold of a *Raspberry Pi* model B, which operates at 700MHz. No modifications to the processor configuration have been required, so we assume that the operating frequency is constant and the chip is not overclocked while our application is running. The board is shipped with 512 megabyte of RAM, which is dynamically assigned to the CPU and GPU, depending on the load on each of them [32]. Although, our application will not make an extensive usage of the GPU, we have not disabled it, so we can use the display output for debugging purposes. The server application is actually writing one line per request/response on the screen, however such a little load can be disregarded. We will assume that the CPU can make entirely use of the memory. The operating system installed on the machine is *Raspbian*, a *Debian* distribution, which has been optimized for the *Raspberry Pi* hardware (*ARMv6*) and is now distributed by the *Raspberry Foundation*.

The WLAN adapter we have chosen is the single-chip USB2.0 network interface controller *RTL8188CUS*. This USB dongle is compatible with 802.11a/b/g/n specification and is included on most of *Raspberry Pi* development kits. The device is an inexpensive and complete solution for 2.4GHz band and according to the datasheet [33] its maximum communication range outdoors is 300 meters. Although for the experiment results we have determined a lower figure to establish a safe and durable connection.

To provide the *Raspberry* with mobility we attached an external battery to the hardware solution. In the decision of what sort of power supply is the most appropriate one, we must consider not only the board, but also the WLAN adapter. Per the *RTL8188CUS* specification the device needs 600 milliamps [33], while the recommendation for the *Raspberry* board is 1 to 1.2 amps [32]. We have acquired an *Sveon SAC 330* power supply, which offers a 6600 milliamp-hour cell. Although this battery has two outputs 1A/2.1A available, we have only used the USB port that supplies 1 ampere for the *Raspberry* because it should be enough for our current implementation. This solution provides us with autonomy of approximately 7 hours.

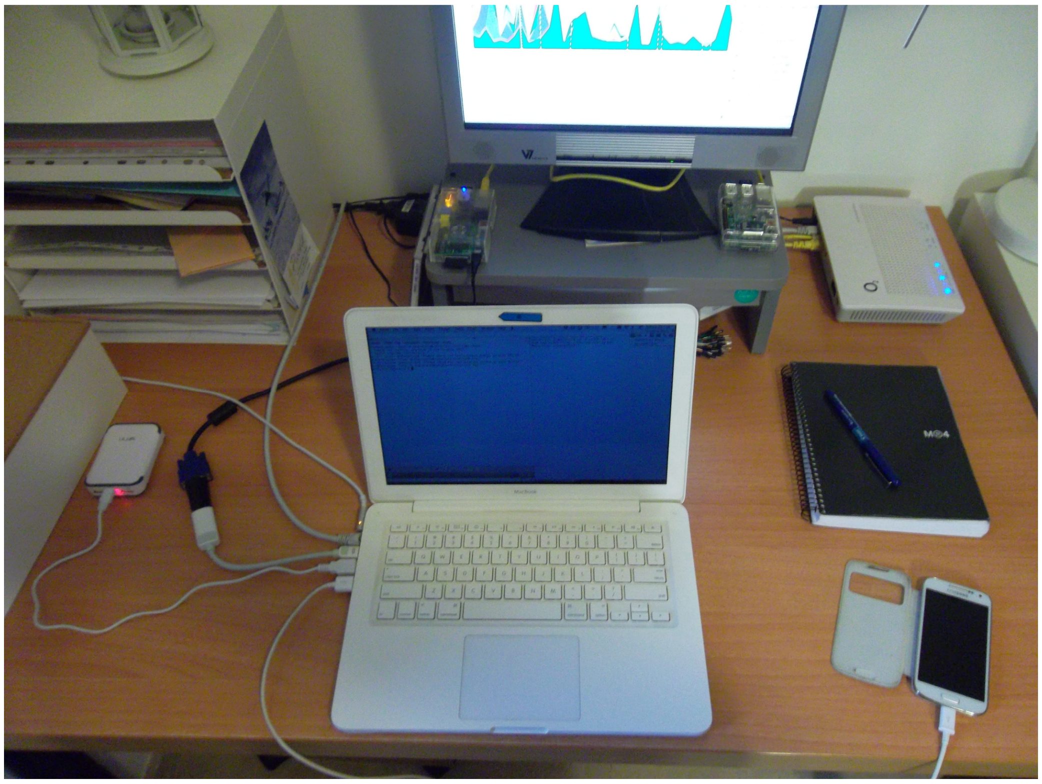


Figure 9: Development setup used during the simulation sessions

On the vehicle side, we have installed our *Android* application on a *GT-I9195*, also known as *Samsung Galaxy S4 mini*. This smartphone with a dual core at 1.73 GHz and 1331 megabyte of RAM belongs to the *Samsung* lower cost lineup that reached a certain degree of popularity because of its price. The device features a *Qualcomm snapdragon S4*, which is based on the *ARMv7* instruction set, but with a high performance for multimedia-related operations. This smartphone is shipped with a Li-Ion battery, which offers autonomy of 1900 milliamp-hour.

The operating system running on the device is *Android 4.4.2*, labeled as *KitKat*. This version of the system is the last one that *Google* released with the virtual machine *Dalvik*, however it is still the most extended one among the *Android* users. According to the data published by *Google*, 87.6% of the devices are still running *KitKat* or previous versions of the operating system [35].

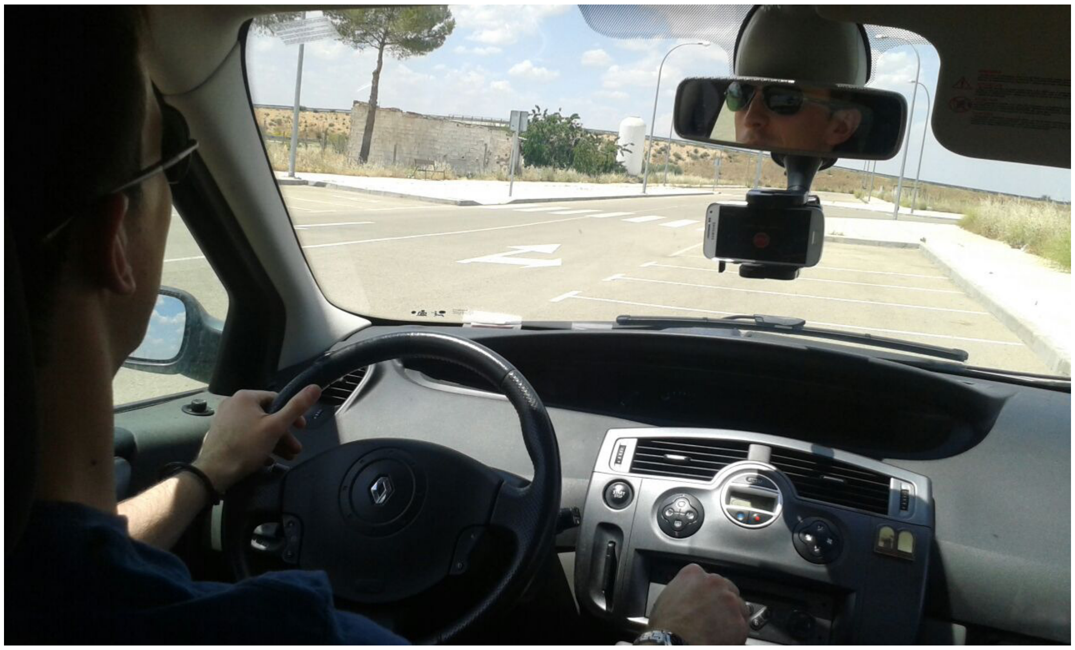


Figure 10: Smartphone car mount in a *Renault Scenic*

During the design time, the application layer was developed on a *MacBook* where we installed some tools from the *Android* Software Development Kit (SDK) and the *Oracle VirtualBox* application. When the software packages were ready, we used this laptop to retrieve the first results as figure 9 shows.

On the one hand we exploit the mini USB interface from the smartphone to retrieve information with the laptop. The *Android Debug Bridge* (*ADB*) tool was running on both phone and laptop to monitor the events of the application. On the other hand we created a small local network connecting laptop and *Raspberry* to a router. We run a *secure shell* client on the laptop and installed the server on the *Raspberry* to monitor any event during the runtime. This setup allows us to debug server and client applications from real devices in parallel and live time on single machine. The local network can also be used for higher number of devices either traffic signs or smartphones; the only limitation is the number of ports.

For the second experiment we went further and took the device to a real environment. We installed a smartphone car mount into the vehicle as figure 10 shows. The vehicle driven was a *Renault Scenic* from the second half of 2006 model year, which included no other type of connectivity, but the FM/AM antenna from the audio equipment.

The *Raspberry* equipped with the wireless network adapter and the external battery was placed approximately 1.5 meters from the floor at the end of a road acting as a stop sign.

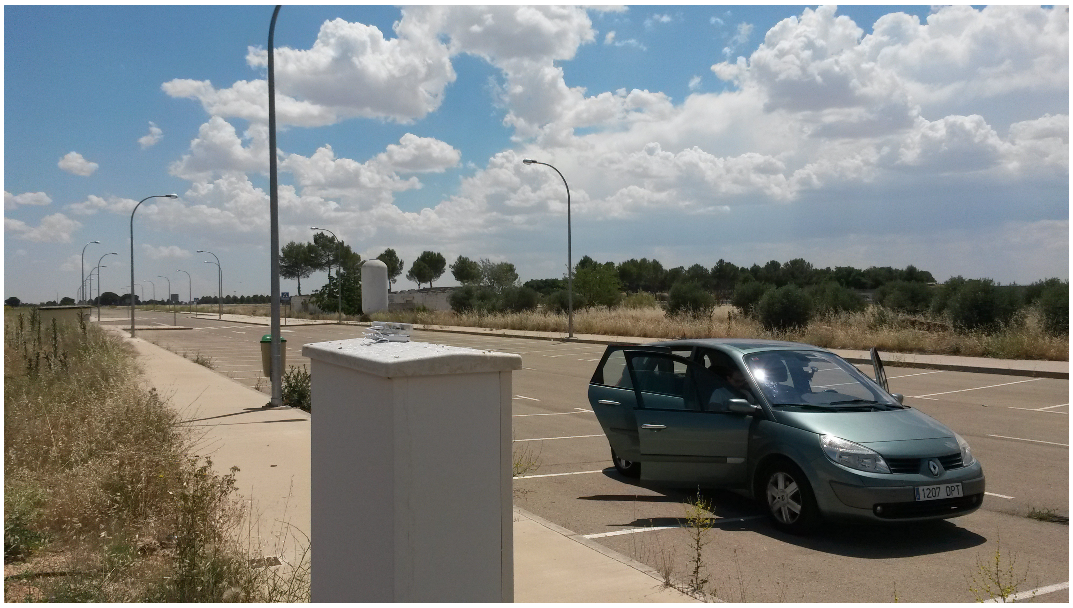


Figure 11: *Raspberry Pi* equipped with external battery during the first driving trial

The third experiment, again in a real environment, was performed driving a different car at a different location. In this case the vehicle was a *Mazda3* from the 2012 model year, equipped with GPS navigation system and audio over Bluetooth, both active during the trials. The *Raspberry* was this time attached to a traffic sign approximately 2.5 meters from the floor. On figure 12 we can see a vehicle approaching the traffic sign.

5.2 Empirical analysis of the implementation

In this section we will describe the evolution of our application based on the results obtained on each experiment. As it has been explained on previous chapters the *Android* client is responsible for discovering the traffic sign, requesting the connection and acquiring the consequent information. Our starting point is an application that does these three steps in a sequence. One phase does not start until the next one has completely finished.

The traffic sign implementation should be always available to the client requests and should reply in a short period of time. We started with a standard setup of the *hostapd*, based on one of the configuration file that comes within the installation package as it is explained on the Appendix I. Basically it builds up a secure WPA2 wireless network, which is encrypted based on a preshared-key and follows the Counter Mode Cipher Block Chaining Message Authentication Code Protocol (CCMP).

5.2.1 Methodology

The first empirical results have been retrieved using the *Android* SDK installed on a computer. We did just slightly modify the code from the *Raspberry* to simulate that the smartphone was leaving the range where traffic sign signal is available. As we can see on figure 9, server and client devices are placed a few centimeters apart. The simulation method basically turns off the *Raspberry* radio for some seconds; hence the *Android* application enters in the scan phase and stays there until the *Raspberry* starts broadcasting beacons again.

The timestamp is retrieved from each event that is considered relevant to the lifecycle of the application like: start scanning, connected, response received, etc. In addition we are running the *CPU Monitor* application [28] that collects information related to the CPU usage, the allocated memory and the battery status.

While smartphone and *Raspberry* are connected to a computer we can use debugging tools like the *ADB shell* to monitor the execution of the application and get results from different experiments very quickly. However a critical factor in the road environment that we are missing in this setup is the vehicle speed. Also, we should consider one of the disadvantages of the smartphone solution: the antenna will be inside the vehicle, which does not help on the quality of wireless communications. These are just a couple of the reasons for the importance of getting results from a real environment.

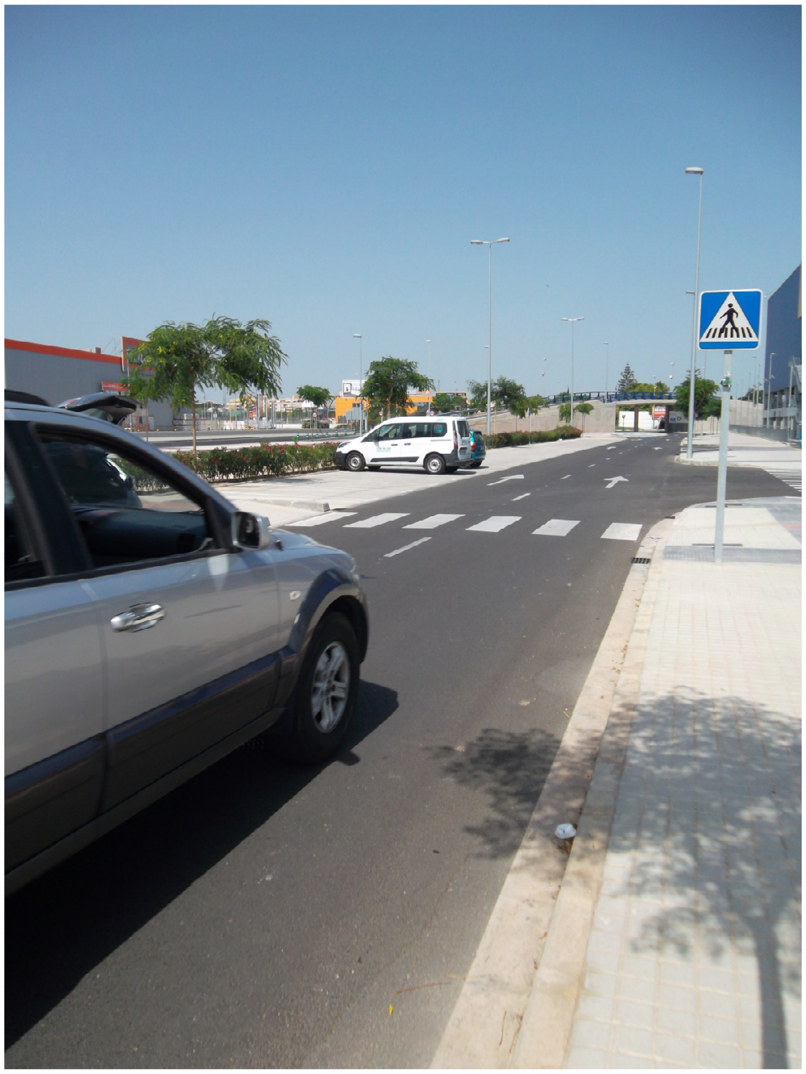


Figure 12: *Raspberry Pi* attached to a zebra crossing sign during the second trial

With this purpose we have modified the *Android* application to register all these events that we were collecting through the *ADB shell*. The user interface implemented on the *AndroidTrafficSign* java class, allows us view this information on the smartphone screen and export the data to a text file. Moreover, this class will also query the *LocationManager* from the *Android* API to retrieve GPS coordinates on each registered event.

All the information from the driving experiments has been retrieved from the logs stored on the smartphone. As mentioned on the previous section, the *Raspberry* gets installed on a fixed location at a certain distance from the floor, while the smartphone is attached to the vehicle windscreen, right below the rear mirror. Each set of results corresponds to the experience of driving the vehicle towards the *Raspberry Pi* at a constant speed, until the car overtakes the sing. After that, the vehicle returns to the initial position to repeat the operation at the same or different speed.

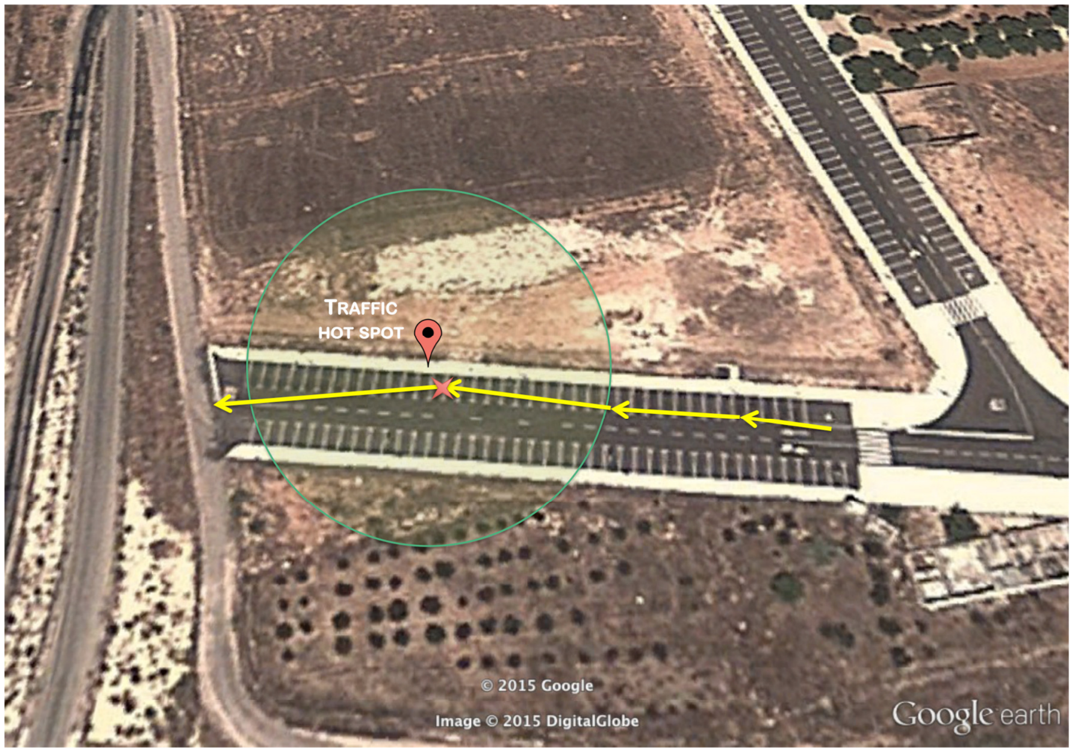


Figure 13: Satellite image from *Google earth* obtained after introducing data results from one of the trials during the first driving experiment.

5.2.2 Mesurable factors

The implementation that we are analyzing in this chapter is based on a complex system, which defines plenty of parameters and can be studied from different perspectives. We have decided to take the advantage of the smartphone, which is fitted all different sort of sensors and data ready for analysis. Therefore all data studied on this section has been collected by the smartphone while running different tests.

Among the parameters that define our system we have selected six factors that might compromise the performance of the whole implementation. In this section we are giving a brief explanation of each of these factors.

- **Detection time**: period required to complete one cycle of the operation since the device starts a scan for traffic signs until the information retrieved from the

RX22 command is displayed to the user. The maximum vehicle speed to obtain the information from a traffic sign depends directly on this parameter. Long detection times lead us to a non-reliable system at high vehicle speeds.

- **CPU usage**: amount of time for which the smartphone central processing unit (CPU) is used to process the instructions of a computer program or the operating system. A high usage can affect the general OS performance and slow down the response of some applications.
- **Memory allocation**: The amount of computer memory used at the system level is essential on the smartphones where physical memory is often constrained. *Dalvik* implements a garbage collection routine that avoids situations where the system runs at a full memory condition. This routine allows *Dalvik* to kill processes that are no longer in used or consuming many resources.
- **Battery power consumption**: The battery life is also an important factor on the final solution. The general public often rejects applications that make an extensive use of the battery resources. Hence we have to escape from using too much smartphone resources unless it is really necessary.
- **Vehicle speed**: In a real environment this parameter is not controlled by the system internally, hence we should be ready to support low and high values. Our objective is to provide the user with a system able to detect signs at least the highest speed allowed by the circulation code at the traffic sign location.
- **Distance to the traffic sign**: This is probably the most important factor that we face on this project. The traffic sign should be detected always before the vehicle overtakes it. On the analysis we have studied this parameter indirectly as a result of vehicle speed divided by the detection time.

5.2.3 Results: cycle time

For the first experiment we used a simulation setup as it is shown on figure 9. The application was running for one hour without interruption. During this time the smartphone has detected 74 times the traffic sign, approximately one every 48 seconds. The simulation code we run on the *Raspberry* turns the radio off for more or less 40 seconds, however we should consider a couple of seconds more for the kernel driver to perform the operation after the code is executed. Hence the first number of the analysis is a rough detection time of 5 seconds.

If we assume that the vehicle is moving at the maximum speed allowed in urban areas, 50 kph [29], the smartphone would cover approximately 69 meters in 5 seconds before the information is prompted to the user. This initial calculation showed a huge limitation on the system; moreover after we realized that when the smartphone is inside the car, it is only able to keep a safe connection if it is within a range of 40 meters from the traffic sign.

We analyzed the data results logged by the computer, which is attached to the smartphone, and we prepared a breakdown of the cycle time. The figures below show the average times that we calculated separately for each stage of the cycle:

SCAN PHASE	2859 ms.
AUTHENTICATION PHASE	134 ms.
ASSOCIATION PHASE	830 ms.
TCP DATA EXCHANGE	483 ms.
DISSASOCIATION PHASE	18 ms.

The scan, authentication and association phases represent the 88% of the cycle time, 52 out of the 69 meters. These results made clear that we had to think on a different strategy for detecting the traffic sign. The problem is that the *Android* API is limited in this area, and poorly documented.

Once the application calls the *startScan* method, it loses entirely the control until the results of the scan are available and the *wpa_supplicant* generates an event. When the supplicant receives the scan request, it executes the respective command to notify the device driver with the operation. The driver performs then a full scan, which in the 2.4 GHz band consists of 13 channels spaced 5 MHz apart. The driver is designed to stay on each channel a certain amount of time before switching to the next one. The reason is that access points send periodically a beacon with relevant information for the connection, without that beacon the communication cannot be established. This period is typically 100 milliseconds [37], however most of drivers use a more conservative value to ensure the beacon does not get missed if the antenna is within the communication range.

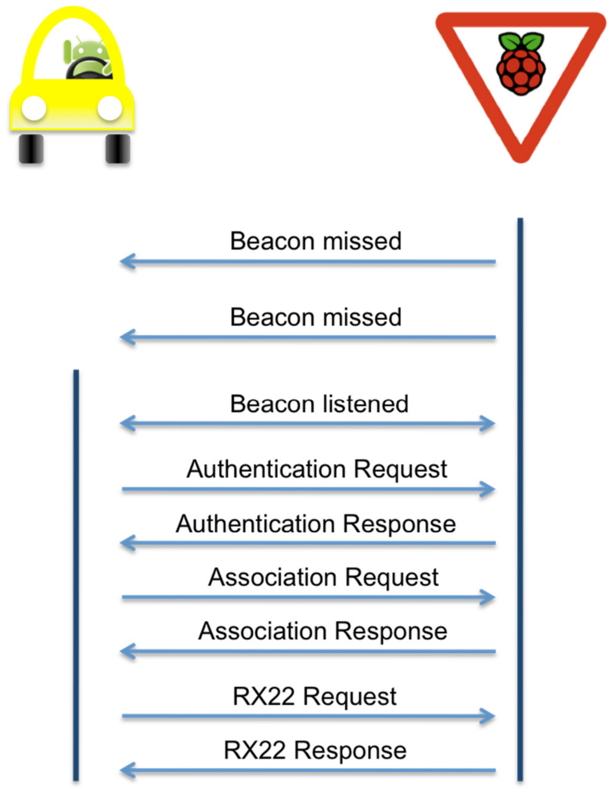


Figure 14: Client-server data exchange process.

For some smartphones, where the driver source code has been released to the public, it is possible to find the particular parameter that defines temporal constants to control the scanning phase. This is the case of the wireless extension (*WEXT*), where we have found a constant named *WEXT_CSCAN_PASV_DWELL_TIME* with a value set to 250. In our implementation, we have not been able to find a similar parameter, however trying not to fall in the idea that every *Android* smartphone will perform at least as well as the one used during the experiments, we will use the value found in *WEXT* on the next calculation.

Theoretically the scan phase should then follow the next equation:

$$Scan\_latency \ = Number\ of\ channels\ x\ Becan\_interval$$

, therefore the scan phase will take at least 3250 milliseconds.

Another fact we have learnt reviewing the open source code for the different components involved on the wireless communication is that the list of known Basic Service Set Identifications (BSSIDs) is handled at a user space level by the *wpa_supplicant*. This fact provides us with an important advantage: the association with a known BSSID is attempted right after it has been detected by the scan service.

In order to benefit from this feature, we provide the *wpa_supplicant* with all the connection details of our digital traffic sign before the scan phase starts. If the device is out the range of traffic sign, the full scan is completed like on the previous implementation. However if the device is within the range, the supplicant will not wait for the results and establish the connection as soon as the beacon is received.

After modifying the code of the application to support this sort of parallel execution, the results from the simulation show two benefits: The first one is that the maximum cycle time was reduced almost by one second. The reason is that now the association time has been absorbed by the scan phase. There has been no case registered where the association phase took longer than a full scan, hence we can simplify the equation of the detection time:

$Detection\_time = Scan\_latency \ + TCP\_data\_exchange$

The second benefit is based on the probability of finding the beacon before the full scan has been completed. If we assume that the listener has exactly the same chances of finding the beacon on any channel, a simple statistical analysis can determine that the scan average time should be 7/13 less than a sequential execution of the phases. Considering the conservative value that we have adopted, 250 milliseconds, the final result will be an average scan latency of 1750 milliseconds.

We decided to take this setup to a real environment and run six tests approaching the traffic sign at a certain speed. We placed the *Raspberry pi* at a fix location, and attached the smartphone to the vehicle windscreen. The results are displayed on table 1.

Vehicle speed(kph)	Experiment	Latitud	Longitud	Event	Distance to traffic sign (m)	Elapsed time (ms)
20	1	39,2704264	-2,7845067	Found	3,74	0
20	1	39,2704264	-2,7845067	RX:22	3,74	205
20	1	39,2704264	-2,7845067	TX:P-1	3,74	308
20	2	39,2704116	-2,784275	Found	20,94	0
20	2	39,2704177	-2,7843358	RX:22	15,72	1063
20	2	39,2704177	-2,7843358	TX:P-1	15,72	1253
30	3	39,2704029	-2,7842433	Found	23,82	0
30	3	39,2704029	-2,7842433	RX:22	23,82	95
30	3	39,2704399	-2,7843048	TX:P-1	17,81	240
40	4	39,2704463	-2,7846856	Found	15,19	0
40	4	39,2704463	-2,7846856	RX:22	15,19	102
40	4	39,2704463	-2,7846856	TX:P-1	15,19	215
40	5	39,2704149	-2,7844909	Found	5,28	0
40	5	39,2704149	-2,7844909	RX:22	5,28	70
40	5	39,2704149	-2,7844909	TX:P-1	5,28	158
50	6	39,2704456	-2,7846453	Found	-11,76	0
50	6	39,2704518	-2,7847721	RX:22	-22,58	159
50	6	39,2704518	-2,7847721	TX:P-1	-22,58	244

Table 1: Position and timestamp registered during the first driving trial.

The communication appears to be established just a few milliseconds after the traffic sign is discovered, between 95 and 1063 milliseconds. The smartphone receives the response after approximately an average of 120 milliseconds. The traffic sign was in none of the cases overtaken after it got discovered. In other words, although the TCP implementation of sign-vehicle communication could be improved, it is good enough for the application goal.

However we can observe that on some cases the sign is discovered just a few meters before the vehicle reaches the traffic sign location, or even when the sign has been already overtaken as it is the case of the last test. The reason is that we still have a maximum detection time of 3250 milliseconds, where the vehicle can cover up to 45 meters at 50 kph.

We were relying on the maximum range value from the device specification [33], where the traffic sign would be detected up to 300 meters away in an outdoors context. From the first experiment in a real environment we realized that this figure was much less reaching a maximum of 40 meters when the receiver device is inside the vehicle. This value effectively limits the maximum speed to 38.4 kph for our system to be reliable, and therefore explains the bad results of the sixth test.

We have studied different possibilities with the intention of reducing the scan latency of the application. Some of them are: using active instead passive scan, applying a selective scan per channels and the *Android* scan only lock. In all cases we have always ended up finding limitations within the *Android* API. For example the idea of listening only on a reduced number of channels to discover other nodes in the surroundings is typically used on ad hoc networks to reduce the discovery overhead. The firmware, kernel driver and supplicant of our implementation support this functionality; however the API reference does not offer any possibility to enter the channel information from the application layer.

There have been some attempts to overcome this limitation by modifying the *Android* software at a lower layer, for example the one proposed by Browers on [36] where the *wpa_supplicant* is set up to scan on a particular channel based on a coded SSID, which is passed through the *WifiManager* as a known network. This sort of solutions requires specific privileges that only a *root* user receives. One of the requirements of the present project was that the smartphone solution should be offered to the user as an application, which only requires a typical installation. Otherwise we will lose potential users that are reluctant to root their phones, hence we cannot adopt Browers proposal on our implementation.

The restrictions imposed by the *Android* API limit the performance of the final solution, however before starting with the next experiment we wanted to increase the maximum speed where the system is still reliable. With this purpose the implementation at the traffic sign was reviewed. The idea was to increase the communication range in order to allow the vehicle more distance to complete the scan phase with successful results.

$$V_{max} = \frac{Communication\_range}{Scan\_latency\_\max \; + TCP\_data\_exchange}$$

After trying different options we decided to upgrade the *hostapd* configuration to support 802.11n capabilities. The measurements showed an important improvement: the mobile device was now able to establish a reliable connection in an open space 100 meters away even if it is attached to the windscreen inside the vehicle. This range increases the reliable maximum vehicle speed to approximately 96 kph.

However an open space is probably one of the best possible scenarios, therefore similar data was collected at the city in a couple of narrow streets were vehicles were parked at both sides of it. The results show a 40% shorter communication range, which still allows a maximum speed of 58 kph, a valid range considering that the limit inside the city is 50 kph.

Another interesting fact we observed is that although we are using an isotropic antenna for the experiments the communication range is not homogeneous when the hardware solution is attached to a real sign. The signal strength appears to be much higher on the traffic sign side where our hardware is installed. The reason seems to be that the physical sign acts like a major obstacle for the AP signal; hence the antenna should be always installed at the front side of the post or at higher point than the sign if we want to avoid short communication ranges.

We took this setup to a real context and record GPS data as well as the events timestamp. The test was conducted twenty five times at different vehicle speeds. The results were not far from previous conclusions, however there is a major difference. Although the cycle time is similar to the previous tests, now the vehicle has longer distance available to discover the traffic sign, therefore the communication gets established earlier, which improves the user experience.

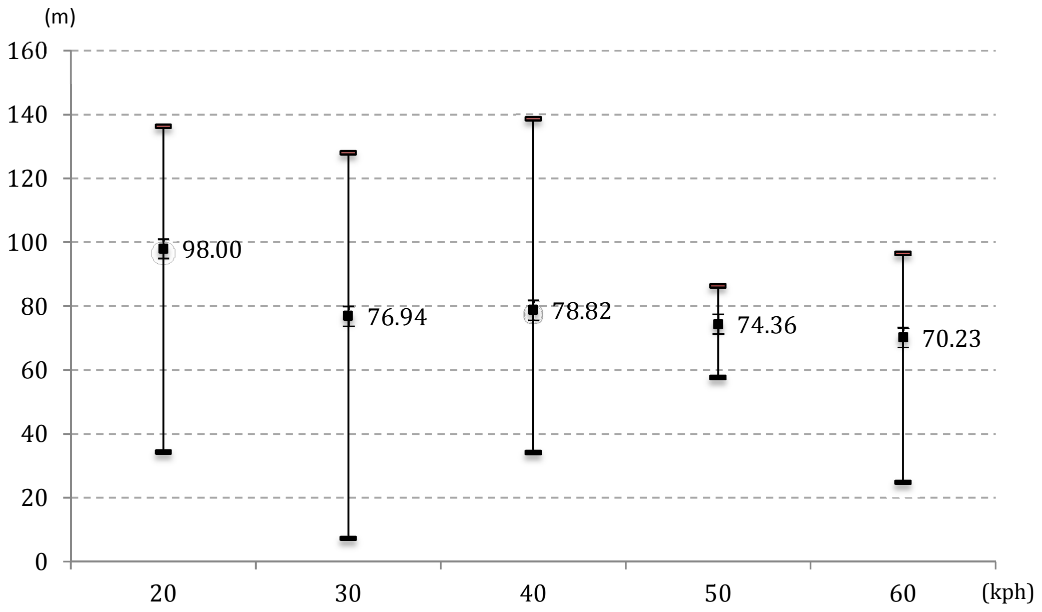
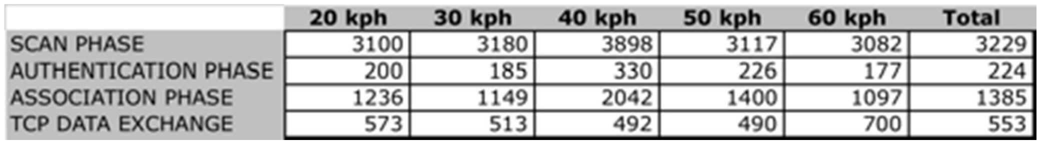


Figure 15: Distance to the traffic sign where the information is received.

Compared with the data collected at the simulation environment, the results show a slight increase on the detection times. We expected this difference because the node is now trying to communicate from inside a metal structure, which can act as a Faraday cage. Also a moving node will suffer the so-called Doppler effect, missing in some cases part of the information sent by the sign.

One more thing we studied is the influence that the vehicle speed causes on the detection time. The idea was to determine whether a higher speed would increase the time that the smartphone needs to complete any of the communication phases. On table 2 we cannot observe a clear trend as we increase the vehicle speed. In fact, if there is any impact, it seems to be shadowed by the variability introduced by the scan phase of the device, which could in some cases perform much better at 60 kph than statically. Therefore we can conclude that the speed is not a major factor on the whole system performance at this stage.



	20 kph	30 kph	40 kph	50 kph	60 kph	Total
SCAN PHASE	3100	3180	3898	3117	3082	3229
AUTHENTICATION PHASE	200	185	330	226	177	224
ASSOCIATION PHASE	1236	1149	2042	1400	1097	1385
TCP DATA EXCHANGE	573	513	492	490	700	553

Table 2: Detection time brake down at different vehicle speeds.

5.2.4 Results: CPU usage and memory allocation

The constrained resources of a smartphone device make critical the usage that applications do of the CPU and memory of the device. The operating system has its own mechanisms to prevent that an extensive usage of these resources causes any effect on the user experience. These mechanisms are a key factor to achieve a good operating system for mobile devices, however they can affect the performance of specific applications. Therefore it is essential to have at least an overview of our application performance.

During some of the trials we have collected data from the system with the help of a third-party application that register CPU usage and memory allocation every 20 milliseconds. This data has been validated with the values registered by the tools that come preinstalled on the device, showing both very similar results. These tools are useful to have an idea of the whole system performance, but they are not sufficient to study a specific application.

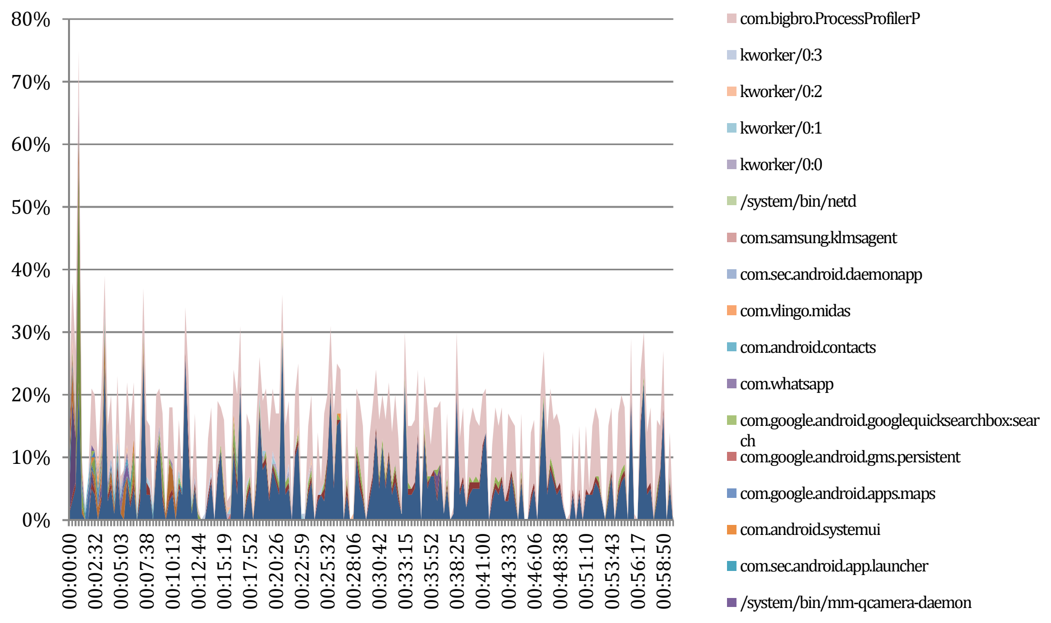


Figure 16: CPU usage breakdown from one-hour simulation sample

After one hour simulation the maximum CPU usage registered by both systems is 99%. The breakdown shows a slightly different figure, we have assumed that this difference is used by the operating system itself and it is not included in the process list. The *trafficsign* process does not register more than a 2% during the whole simulation, however we cannot fall into the error of assuming this is the only usage that our application is adding to the system during the execution. As we explained in previous chapters, *Dalvik* creates a virtual machine instance each time an application gets launched. The process that covers the virtual machine is called *system_server*. We can observe on figure 16 how the *system_server* process is loading the CPU in cycles, each peak coincides with a *startScan* request, and hence we understand that this process is responsible to transmit the application requests to the user space, where the *wpa_supplicant* belongs.

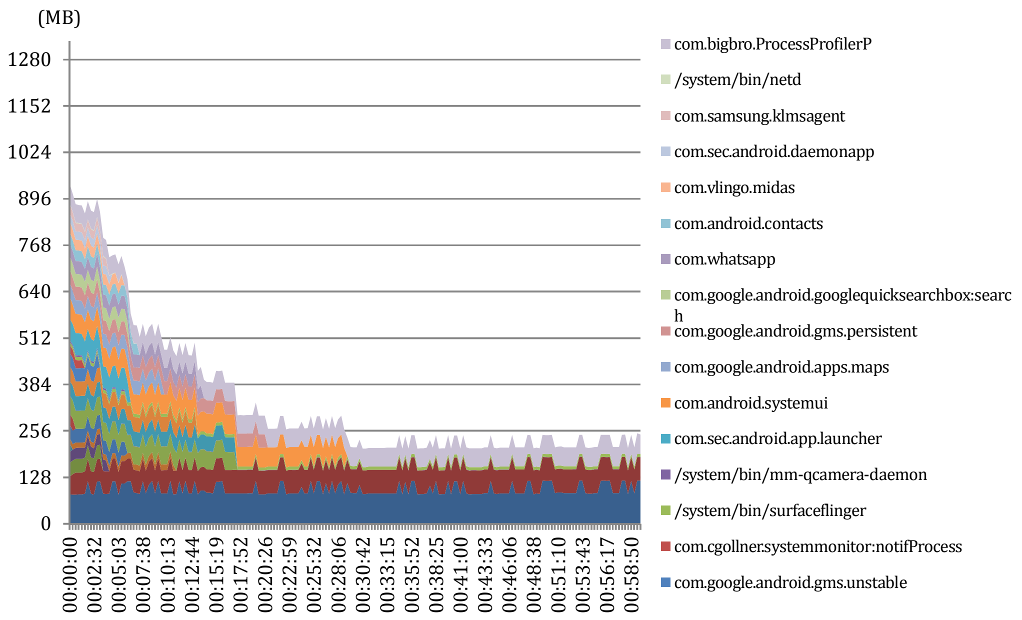


Figure 17: Memory allocation breakdown from one-hour simulation sample

Regarding the memory usage, we have not observed a direct relation between the events registered by our application and the memory that is allocated. As Figure 16 shows, *trafficsign* process allocates an average of 64 megabytes, less than a 5% of the memory available on our case. We can determine that the application is not prone to cause issues related with memory shortage.

5.2.5 Results: battery power consumption

Since our application is meant to run in the background for long periods of time, the battery power consumption is an important factor that will influence on the user satisfaction. Previous chapters expose a strong emphasis on the importance of a good reception of the client solution. We cannot fail in providing a battery-efficient application; otherwise we might loose potential users.

We have performed a test recording the battery state of charge every 20 milliseconds on two different situations. During the first test the smartphone was running our application at the simulated scenario, which has been described on the previous section. The screen was on all the time as per a lock implemented at the application user interface. Also the Wi-Fi radio was running all the time because we force it by the service in charge of discovering traffic signs.

During the second test, the smartphone was running just a few applications also executed on the first test, which are very popular and might be found in most of smartphones. Some of them are *Twitter*, *RSS feed*, *whatsapp*, *tripadvisor*, and others that come preinstalled on the device.

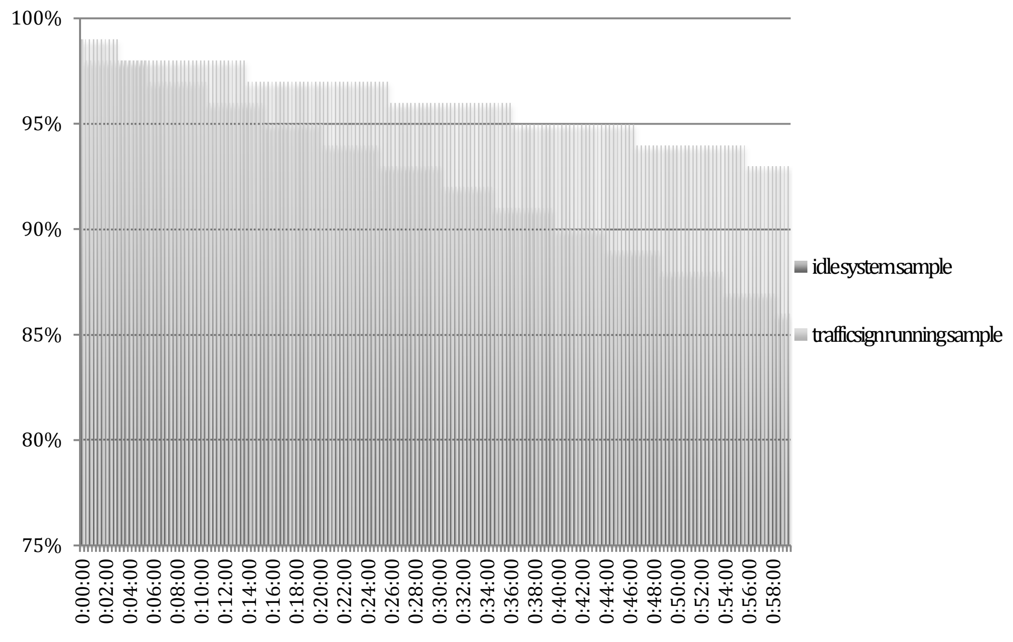


Figure 18: Battery state of charge from one-hour simulation sample

The results are shown on figure 18. Our application causes the battery to discharge 46% faster, not much more than leaving the Wi-Fi enabled on the smartphone by accident. Hence the *trafficSign* process is able to run on a smartphone with a Li-Ion battery of 1900 milliamps-hour for more than seven hours.

5.3 Brief comparison with other TSR solutions for smartphones

In this chapter we have analyzed the behavior of our application in different scenarios. The results seem promising despite the limitations found on the smartphone implementation. However we want to go a step further and compare our solution with other TSR applications that are already available in the market. This will help us to create a more complete outlook of the efficiency of our application.

Three applications have been chosen to carry out an experiment. The first one, *myDriveAssist*, which has been developed by *Robert Bosch GmbH* [25]. This application is a camera-based TSR solution, which uses the built-in camera from the smartphone to detect traffic signs and advice the driver with the speed limit for a certain section of the road. There is a limit of one sign prompted at a time, which is not an issue because only speed limit traffic signs are recognized by the system. Apart from the camera, the application also requires GPS data to retrieve additional information like vehicle speed and driving direction.

The second application is called *aCoDriver*, which implements a hybrid TSR system. Like *myDriveAssist* it uses the smartphone camera to detect vehicle speed signs and the GPS sensor for the vehicle speed and direction. However, unlike the previous application, it also uses the data to query a speed limit database [26], which is not stored locally. This functionality requires Internet connection; therefore our device should be always connected through a mobile network to benefit from the hybrid system advantage. Otherwise it would rely on the information captured by the camera only.

	Robert Bosch myDriveAssist	Evotegra's aCoDriver	Michelin Navigation
Detectability (Urban area)	43%	14%	71%
Detectability (Freeway)	96%	64%	48%
Accuracy (Urban area)	67%	100%	N/A
Accuracy (Freeway)	57%	50%	N/A

Table 3: Empirical detectability and accuracy rates from different TSR applications.

Finally, in order to have a general view of the present situation of TSR solutions based on smartphones, we have also evaluated a data mapped system, which relies completely on the database information. This application has been developed by *Michelin* and uses the reference of the GPS sensor to retrieve speed limit information from an external dataset [27]. Unlike *aCoDriver*, the Internet connection is mandatory to run the application.

All these applications are available at the *Google Play Store* and their official websites, which address can be found on the reference section of this document.

The test consisted of monitoring the applications during approximately one-hour trip. The itinerary was 30% urban and 70% freeway, covering approximately a distance of 50 kilometers. The vehicle we used is the same one as per the last experiments on the previous section. The three applications were installed on the *Galaxy S4 mini* from the last trial. In all three cases the smartphone was attached to the vehicle windscreen in the same way as we did for *TrafficSign* evaluation. We covered the urban area within a speed range of 30-40 kph, while on the freeway we reached 100-120 kph. Considering only the speed limits, in the itinerary we can find up to 7 signs to discover in the urban area, and 25 in the freeway.

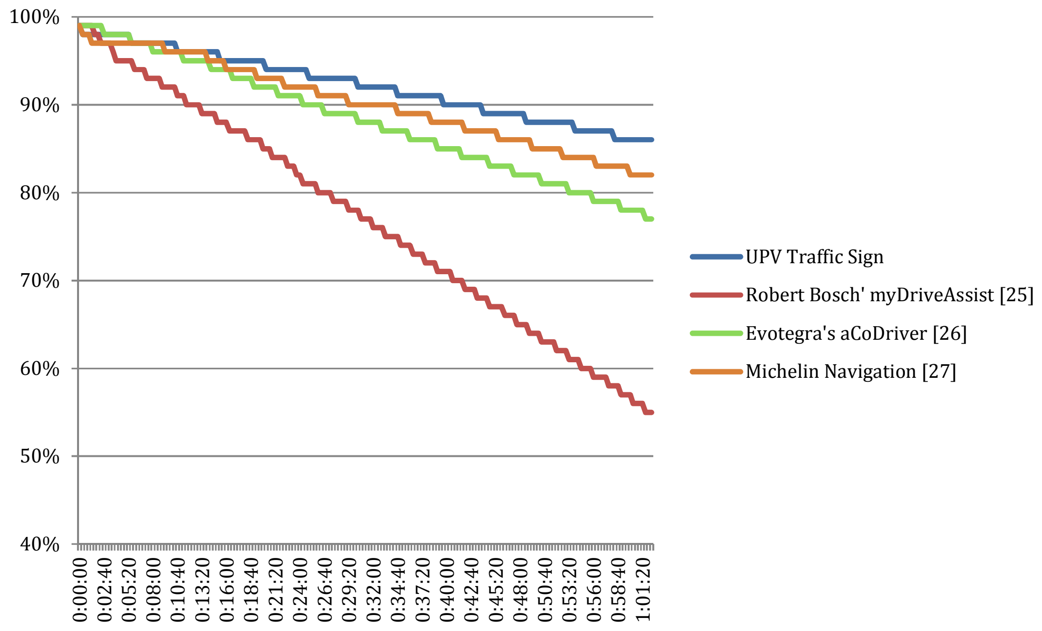


Figure 19: Battery power consumption comparison

Table 3 shows that the camera-based system is the most capable for detecting speed limits with an 84% of the traffic signs detected. However it seems to perform better on a freeway than in urban areas. These results are only surpassed in the urban context by the data-mapped system with a 71% of the speed-limited areas correctly prompted to the driver.

Regarding the accuracy, neither the hybrid nor the camera-based system has great performance. In the case of *aCoDriver*, the application prompted almost a half of the signs erroneously. In both cases most of them belonged to a different road that can be found in parallel to the itinerary. Other false speed limits were prompted while driving the vehicle close to freeway exit. The only invalid result in the urban area was registered by *myDriveAssist*, which detected the speed limit from the rear part of a truck, where its maximum speed is displayed. We are not able to calculate the accuracy for data-mapped system because the detection events are not registered by the application; hence all erroneous speed limits might have been included within the detectability figures.

While conducting these trials we have also run the *CPU Monitor* application [28], which records different values from the processes registered by the operating system. In this section, instead of preparing the data from these tests in one report per application, similar to the ones shown at figure 15 and 16, we have only taken the data from the processes that are relevant to the TSR application and prepared a single graph that we show in figure 20.

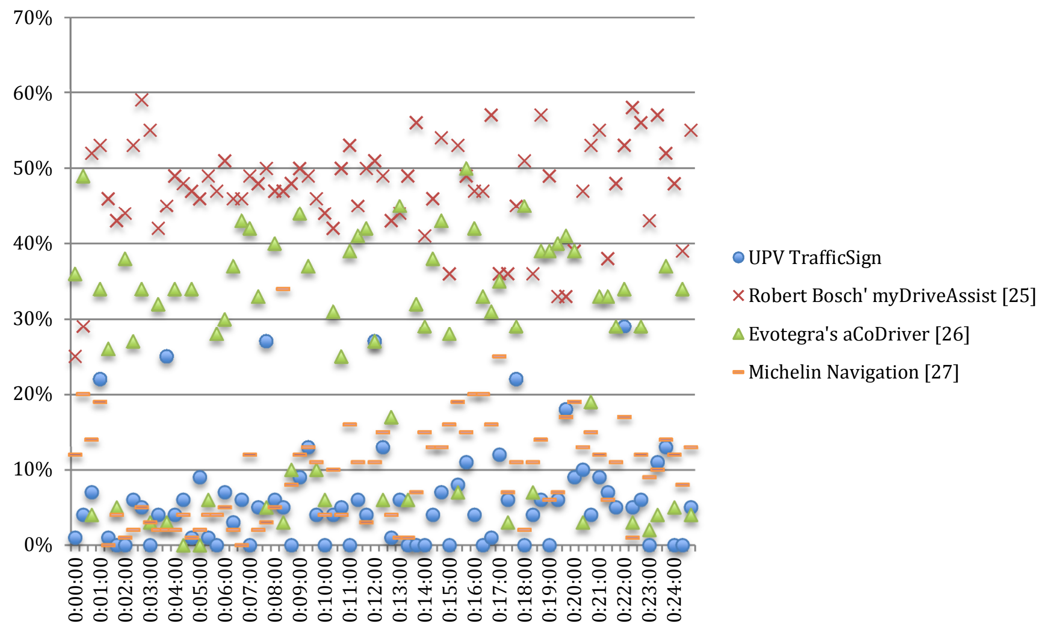


Figure 20: CPU usage by application

From the results the two applications that use the camera for sign detection are the ones making the most extensive use of the CPU. The other two, *Michelin Navigation* and *TrafficSign*, rely on the network interface as the main element in the smartphone. Unlike the others, the main component is used based on the application demand, and not continuously, hence the CPU load should be less uniform, but lower than the camera-based systems in general.

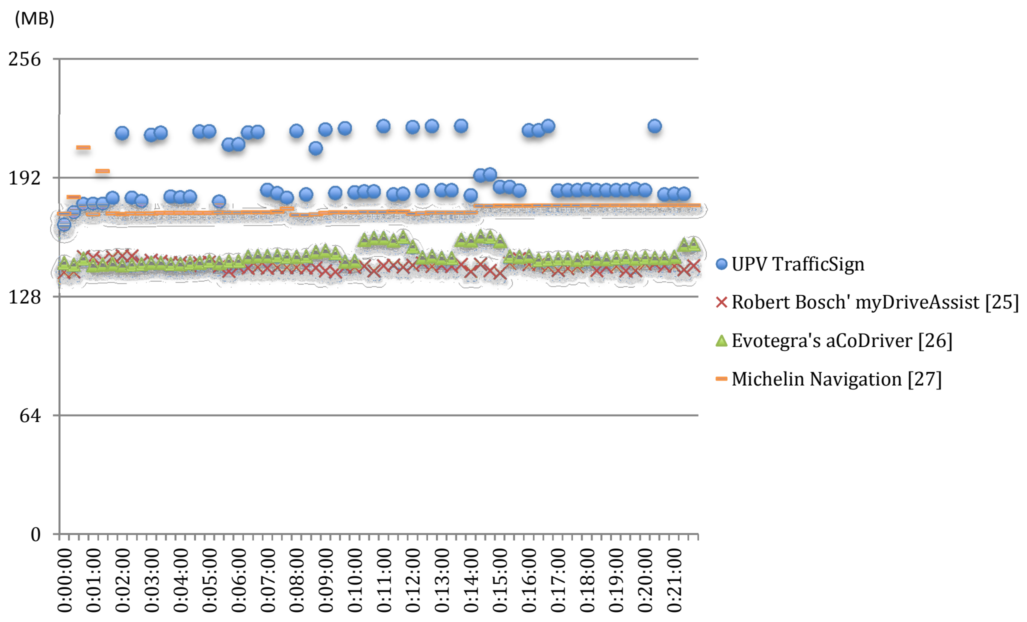


Figure 21: Memory allocated by application

Figure 21 shows the memory requirements for each of application. On none of the systems studied we have observed a major usage of the smartphone memory. The highest figures are registered by our application, however the maximum value recorded does only represent the 16% of the smartphone RAM.

Finally we have analyzed one of the most critical factors on a smartphone, the battery usage. Figure 19 shows the state of charge while the application is running. After one hour the data shows that our application is in the best position with an 86 percent of the battery load. On the other hand *myDriveAssist* has already consumed 44 percent of the capacity; hence the application will be running for another hour and a half before the device runs out of battery, unless it gets connected to a power supply.

5.4 Summary

In this chapter we have evaluated our proposal for wireless technology applied to traffic sign recognition in a real environment. Analysis of the data obtained shows that there are major reasons for further studies about the opportunity of using smartphones to expand the number of potential users. With a low budget we have implemented an application that is able run for a long time on any *Android* device without potential CPU or memory capacity issues and detect the traffic sign that are equipped with our wireless solution at any vehicle speed up to 96 kph.

The main limitations we have discovered during the implementation are due the APIs involved in wireless communication for *Android*. The *WifiManager* class, which handles most of the interaction between the application and the Wi-Fi controllers, is very limited compared with other operating systems. Moreover the scanning and association phases are not very well documented in the official *Android* reference, and most of premises we have followed during the implementation are based on descriptions of experiences made by investigation groups and developers.

Nevertheless we have compared our TSR solution with others *Android* applications based on different strategies, and it proves the benefits of using wireless technologies among the other exploit techniques underlined in this document. Our system does not only offer the possibility of detecting a wider range of traffic signs and have additional capabilities like providing live information from the area, but also we have verified that makes a more efficient use of the smartphone resources like battery or CPU.

Chapter 6.
Conclusions & Future work

The present document introduces a TSR system proposal based on wireless technology, which has been implemented and tested in an *Android* smartphone and a *Raspberry Pi*. The project makes use of existing initiatives from standards and researches in the scope of traffic sign recognition and digital signs. However it introduces the original idea of using the smartphone to offer a complete vehicle-to-infrastructure solution. This concept is led by the objective of closing the gap between technology prerequisites on one side and the requirement of potential users to easy an infrastructure change on the other.

Along with the benefit of using an extended operating system with a network stack that is available on more than the 80% of the mobile devices, the *Android* API also introduces the main constrain to the proposed system. The fact that the *WifiManager* class provides limited control over scan and association phases results on a lower-performance solution than what the hardware and communication standards could offer. An important part of the work presented in the document has been dedicated to overcome this limitation with a certain degree of success.

The data obtained from the different experiments showed promising results that authorize our initial premise: using the smartphone as the opportunity to expand wireless technology in the traffic sign recognition context. The maximum vehicle speed where we can ensure the reliability of the system is over ninety kilometer per hour. The application does moderate battery power consumption, covering long distances without connecting the device to a power supply.

The *Raspberry pi* has accomplished our requirements for the outdoor tests with a simple and robust implementation. Nevertheless for future extensions we do not need to stick to a unique hardware solution. For example the introduction of directional antennas or wireless sensor solutions, like *waspmote*, could be a future case of study.

Ultimately this document proposes a system that should be flexible at different layers of the designed stack. We pursued the intention of implementing a cost-effective solution that will be potentially upgraded in the future. Therefore flexibility and compatibility are important attributes to hold over the time.

6.1 Recommendation for future work

The solution presented on this document has fulfilled our necessities for the system evaluation. It could also be perfectly used for the initial implementations on real environments. Nevertheless the present work can be further developed in the future in a number of ways. In this section we comment some of them.

6.1.1 Peer-to-peer strategy

The present TSR system has been designed upon the fact that the two major operating systems for smartphones do not currently support ad-hoc network standards. During the development phase of the *Android* application we analyzed and tested different strategies to reduce the discovery and association latencies in an infrastructure network. We had not much success on this attempts due to some constrains found in the interaction with the wireless adapter. Essentially we focused our study on the possibilities offered by the *WifiManager* class, *Android* API for infrastructure networks. However during we also realized about a different java class, *WifiP2pManager*, which handles peer-to-peer connectivity over a similar communication stack.

Google offers this solution for wireless communication in a scenario where there is no network infrastructure. The *Wi-Fi Direct* or *Wi-Fi P2P* is not an IEEE standard, but a *Wi-Fi Alliance* specification. It allows devices to connect to each other without the use of a traditional Access Point. From the *WifiP2pManager* class, there is a particular method that has drawn our attention; its name is *discoverPeers*. This method would serve a similar functionality as the *WifiManager.startScan* in our application, however it accepts an argument to select a specific channel, which is used during the discovery phase. Instead of listening for beacons during a specific amount of time channel by channel, the device alternately listen and send probe requests on those specific channels, which are called "social channels".

Wi-Fi Direct is officially supported by *Android* since version 4.0, also known as *Ice Cream Sandwich*, and has been marketed as a replacement of Ad-hoc mode since then. Although Ad-hoc and *Wi-Fi Direct* belong to different types of communication, the message from *Google* could be interpreted as *Android* not supporting the Ad-hoc standards in the near future. A case of study could be how to apply *Wi-Fi Direct* on the traffic sign recognition context. The current TSR system implementation, which has already been presented in this document, will support peer-to-peer connection with a slight modification to the *TrafficSignMonitor* class. If we attach the *Wi-Fi Direct* events to the finite state machine and extend *WifiP2pManager* methods, we should be able to create a beta version of the code with just a little effort. On the other hand, *hostapd* does also support the *Wi-Fi Direct* specification; we will just need to modify our *Raspberry* network configuration to have a complete peer-to-peer solution for evaluation.

6.1.2 Multiple access points

All experiments presented in this document have been carried out with a single traffic sign and only one smartphone. It allowed us to reduce the system complexity and focus on the limitations that are implicit in the nature of the implementation. This way, we have set a clear performance outlook, which will be potentially used as a reference for future calculations in a scenario where the number of nodes has increased.

We have included in the code a mechanism that avoids the supplicant trying to exchange information to undesired Access Points. This is to prevent malicious attacks based on replication, but also to avoid an increase of the detection time by connecting to a different network. The mechanism uses basically two parameters: (1) Service Set Identifier (SSID) and (2) Basic Service Set Identifier (BSSID). The first one is passed to the *wpa_supplicant* through the *WifiConfiguration* class, which includes it on the list of known networks. The device will only attempt to connect to the network if the information broadcasted by the Access Point matches with an entry from the supplicant list. The second one has been implemented at the application layer, so that before sending any TCP request the MAC address is compared against a list, filtering the negative results.

For a good performance in a scenario with several traffic signs, we should determine a set of rules that specifies the number of known SSID entries on the supplicant list and when the application should add or remove them from the list. For example in a bidirectional street we can use two different SSID, one per direction. When the device realizes that the vehicle is moving on one direction it will remove the other SSID entry from the list, so the supplicant stops trying to connect to those Access Points that are only relevant to vehicles driving on the opposite direction. We could also add a wild card SSID at both ends of the street that determine when the vehicle enters and leaves the street as well as the direction of the vehicle.

Also, in order to get a more efficient system, in the future the BSSID filter should be moved to the OS user space in a similar way to the SSIDs supplicant list. Per the *Android* reference, the *WifiConfiguration* class includes a field called *BSSID*. Its purpose is to limit the supplicant entry of known networks to a single Access Point identified by the MAC address. The smartphone will then apply the filter before the connection has been established, saving an important amount of time when the traffic sign contains information that is not relevant to the vehicle itinerary. For example, if we think of a roundabout connected to four streets, each street might have its own digital traffic sign. The application is able to detect all four, however we are only interested on the one placed at the end of the street that we are driving on. This sign is typically the first one that the device will discover because of its proximity. If the sign is able to provide the BSSID information of the other three, we can disregard them when driving around the roundabout.

This mechanism intends to provide a system that can efficiently work without the use of location information. One of the benefits is that keeps the battery at low consumption. Nevertheless future works could study the possibility of using the GPS data to enhance the user experience.

6.1.3 Active safety

Most of built-in traffic sign recognition systems have evolved to a fundamental part of the active safety concept. The information recorded by the system is not only displayed to the driver, but also used for functionalities like adaptive cruise control or audible warnings when the vehicle exceeds the speed limitation.

The importance of collecting traffic sign information through a digital media will just grow in the future with concepts like the connected vehicle or the self-driving car. The idea of controlling critical car functions through an interface with our smartphone has not been studied in the document because we do not believe that such an interface is under the scope of OEMs. However earlier this year *Google* released *Android Auto*, a standard to allow mobile devices to be operated in automobiles through the dashboard. The system is already available on recent launched models like the *Hyundai Sonata* or the *Honda Accord*, and has been announced by other automakers.

A similar application to the one we have presented on the present document would be able to run on this extension of *Android*. This fact increases the potential of the smartphone as platform for traffic sign recognition. From the application layer we can now make use of some of the hardware that is installed on the vehicle, for example the GPS antenna. Also some vehicles are equipped with Wi-Fi facilities, which might be accessible by our application. The benefit of using an external antenna instead of the one embedded in the smartphone will definitely increase the communication range and therefore the system performance.

Nevertheless the major benefit introduced by this standard is the fact that it is not unreasonable to assume that certain applications will be able to interact with the vehicle in the future. In the end, this OS extension might open the doors to safety applications based on *Android* solutions.

References.

[1] Abdullah, Dhuhabasheer, and Mohammed M. Al-Hafidh. "Developing Parallel Application on Multi-core Mobile Phone."

[2] Jiang, Daniel, and Luca Delgrossi. "IEEE 802.11 p: Towards an international standard for wireless access in vehicular environments." Vehicular Technology Conference, 2008. VTC Spring 2008. IEEE. IEEE, 2008.

[3] Eichler, Stephan U. "Performance evaluation of the IEEE 802.11 p WAVE communication standard." Vehicular Technology Conference, 2007. VTC-2007 Fall. 2007 IEEE 66th. IEEE, 2007.

[4] ETSI, TCITS. "Intelligent Transport Systems (ITS); European profile standard on the physical and medium access layer of 5 GHz ITS." Draft ETSI ES 202.663 (2009): V0.

[5] Vandenberghe, Wim, Ingrid Moerman, and Piet Demeester. "On the feasibility of utilizing smartphones for vehicular ad hoc networking." ITS Telecommunications (ITST), 2011 11th International Conference on. IEEE, 2011.

[6] Choi, Pilsoon, et al. "A case for leveraging 802.11 p for direct phone-to-phone communications." Proceedings of the 2014 international symposium on Low power electronics and design. ACM, 2014.

[7] http://www.its.dot.gov/safety_pilot/safety_pilot_progress.htm

[8] Yoshimichi, S., and M. Koji. "Development and evaluation of in-vehicle signing system utilizing RFID tags as digital traffic signs." International Journal of ITS Research 4.1 (2006): 53-58.

[9] Pérez, Joshué, et al. "An RFID-based intelligent vehicle speed controller using active traffic signals." Sensors 10.6 (2010): 5872-5887.

[10] Bohonos, S., et al. "Universal real-time navigational assistance (URNA): an urban bluetooth beacon for the blind." Proceedings of the 1st ACM SIGMOBILE international workshop on Systems and networking support for healthcare and assisted living environments. ACM, 2007.

[12] Lai, Ching-Hao, and Chia-Chen Yu. "An efficient real-time traffic sign recognition system for intelligent vehicles with smart phones." Technologies and Applications of Artificial Intelligence (TAAI), 2010 International Conference on. IEEE, 2010.

[13] García-Garrido, Miguel A., et al. "Complete vision-based traffic sign recognition supported by an I2V communication system." Sensors 12.2 (2012): 1148-1169.

[14] World Road Association (PIARC) Road Tunnels Manual http://tunnels.piarc.org/tunnels/ressources/1/58,2001-05.11.B-Chap-5-EN.pdf

[15] Informe Económico de las Telecomunicaciones y del Sector Audiovisual 2014 http://www.cnmc.es/Portals/0/Ficheros/Telecomunicaciones/Informes/Informes%20Anuales/2014/Informe%20Telecomunicaciones%20CNMC%202014.pdf

[16] Brkic, Karla. "An overview of traffic sign detection methods." Department of Electronics, Microelectronics, Computer and Intelligent Systems Faculty of Electrical Engineering and Computing Unska 3 (2010): 10000.

[17] Zakir, Usman. Automatic road sign detection and recognition. Diss. © Usman Zakir, 2011.

[18] Viola, Paul, and Michael Jones. "Robust real-time object detection." International Journal of Computer Vision 4 (2001): 51-52.

[19] Par, Kerem, and Oğuz Tosun. "Real-time traffic sign recognition with map fusion on multicore/many-core architectures." Acta Polytechnica Hungarica 9.2 (2012): 231-250.

[20] Loy, Gareth, and Nick Barnes. "Fast shape-based road sign detection for a driver assistance system." Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on. Vol. 1. IEEE, 2004.

[21] Gavrila, Dariu M. "Traffic sign recognition revisited." Mustererkennung 1999. Springer Berlin Heidelberg, 1999. 86-93.

[22] Paulo, Carlos Filipe, and Paulo Lobato Correia. "Automatic detection and classification of traffic signs." Image Analysis for Multimedia Interactive Services, 2007. WIAMIS'07. Eighth International Workshop on. IEEE, 2007.

[23] Bénallal, Mohamed, and Jean Meunier. "Real-time color segmentation of road signs." Electrical and Computer Engineering, 2003. IEEE CCECE 2003. Canadian Conference on. Vol. 3. IEEE, 2003.

[24] Bahlmann, Claw, et al. "A system for traffic sign detection, tracking, and recognition using color, shape, and motion information." Intelligent Vehicles Symposium, 2005. Proceedings. IEEE. IEEE, 2005.

[25] Robert Bosch GmbH myDriveAssist http://mydriveassist.bosch.com/mydriveassist/

[26] Evotegra GmbH aCoDriver http://www.acodriver-shop.com/

[27] Michelin Navigation http://mn.viamichelin.com/fr/

[28] ECNApps CPU Monitor http://androidlookup.org/default.aspx

[29] Reglamento General de Circulación para la aplicación y desarrollo del texto articulado de la Ley sobre Tráfico, Circulación de Vehículos a Motor y Seguridad Vial

[30] Gartner, Inc. "Market Share: Devices, All Countries, 4Q14 Update." http://www.gartner.com/document/2985017

[31] "Drawing of Raspberry Pi model B rev2" by Efa - OpenOffice Draw and Inkscape. Licensed under CC BY-SA 3.0 via Wikipedia

[32] Raspberry Pi official website help section and forums. https://www.raspberrypi.org/forums/

[33] Realtek WLAN chipset drivers and utilities http://152.104.125.41/downloads/downloadsView.aspx?Langid=1&PNid=21&PFid=48&Level=5&Conn=4&ProdID=277&DownTypeID=3&GetDown=false&Downloads=true

[34] The great idea finder http://www.ideafinder.com/history/inventions/trafficlight.htm

[35] Java Library for Android Software Reference http://developer.android.com/reference/packages.html

[36] Brouwers, Niels, Marco Zuniga, and Koen Langendoen. "Incremental wi-fi scanning for energy-efficient localization." Pervasive Computing and Communications (PerCom), 2014 IEEE International Conference on. IEEE, 2014.

[37] Ishwar Ramani and Stefan Savage. Syncscan: practical fast handoff for 802.11 infrastructure networks. In INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE, volume 1, pages 675–684. IEEE, 2005.

[38] wpa_supplicant reference https://w1.fi/cgit/hostap/plain/wpa_supplicant/

Digital traffic sign setup

This document explains how to configure a Raspberry Pi as a digital traffic sign. The commands used on our example are for Raspbian 3.18, however most of them are fairly common on most of Linux distributions. The complete hardware solution includes an USB wireless adapter apart from the Raspberry. During the configuration process you will need Internet connection to download the driver and applications from the repository.

Installing

Before installing the packets we recommend to execute a system upgrade if it has not been recently performed.

```
sudo apt-get update
```

Once the system is up-to-date we can install the hostapd and dnsmasq packages from the official repository.

```
sudo apt-get install hostapd dnsmasq
```

Configuring dnsmasq

The dnsmasq configuration is defined in the /etc/dnsmasq.conf file. We open the file with a text editor, in our examples we have used nano, which is preinstalled on the Raspbian distributions.

```
sudo nano /etc/dnsmasq.conf
```

We just need the service to run as a DHCP server for the WLAN adapter, so the following lines should be appended to the file:

```
interface=wlan0
no-dhcp-interface=eth0
dhcp-range=10.10.10.150,255.255.255.0,12h
no-resolv
```

Configuring hostapd

Since our setup is rather common we will use one of the files that come with the package as starting point to configure the service. We copy the file to the hostapd configuration folder.

```
zcat /usr/share/doc/hostapd/examples/hostapd.conf.gz | sudo tee –a
/etc/hostapd/hostapd.conf
```

We now open the file and modify some of the parameters to customize the service for our solution.

```
sudo nano /etc/hostapd/hostapd.conf
```

With the following modification hostapd will build up a secure WPA2 network based on a preshared-key authentication and the CCMP encryption algorithm.

```
driver=rtl871xdrv
ssid=WardAP
country_code=es
hw_mode=g
channel=6
auth_algs=1
wpa=2
wpa_passphrase=WardTEST
wpa_key_mgmt=WPA_PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
```

Now we just need to let the service know the configuration file location.

```
echo ‘DAEMON_CONF=”/etc/hostapd/hostapd.conf” >>
/etc/default/hostapd
```

Compiling Realteck hostapd

At this stage, if we try to run hostapd, the service would return a driver not found error. It means that our hostapd version is not compatible with the chipset of the wireless adapter that we are using. We could have modified the code, however Realteck, the chipset manufacturer, distributes a version of the service, which is compatible with the adapter. The following steps explain how to download and compile the file to overcome the issue.

```
wget
http://12244.wpc.azureedge.net/8012244/drivers/rtdrivers/cn/wlan/0001-
RTL819xSU_usb_linux_v2.6.6.0.20120405.zip
unzip RTL819xSU_usb_linux_v2.6.6.0.20120405.zip
mv RTL8188C_8192C_USB_linux_v3.4.4_4749.20121105/ rtl
cd rtl/wpa_supplicant_hostapd
unzip wpa_supplicant_hostapd-0.8_rtw_20120803.zip
cd wpa_supplicant_hostapd-0.8/hostapd
make
```

The last step might take a while, however once we have the binary file, we do not need to go through these steps for each Raspberry Pi that we set up. The binary file as well as the configuration files should be the same for each traffic sign, hence we should be able to create a repository and save some time copying the files from there.

We will rename the current hostapd binary file and replace it with the Realteck version. The configuration files and rest of the installation remain the same.

```
sudo mv /usr/sbin/hostapd /usr/sbin/hostapd_archive
sudo mv hostapd /usr/sbin
sudo chmod 755 /usr/sbin/hostapd
```

Configuring the network interface

We have already downloaded and configure both services required to run a wireless interface on our digital sign. Now we will disable the adapter and make a slight modification to the network interfaces configuration file.

```
sudo ifdown wlan0
sudo nano /etc/network/interfaces
```

If there is already a configuration for wlan0, it should be overwritten. Otherwise we can just append the following lines to the file.

```
iface wlan0 inet static
        metric 0
        address 10.10.10.1
        netmask 255.255.255.0
```

Please notice that we are keeping the eth0 interface for future use. The parameter metric is set to give priority to the wireless interface over Ethernet. This solution is simple and very efficient if we are not making a frequent usage of the Ethernet connection, however if we plan to have all traffic signs connected to a server for maintenance and data control we recommend the use of IP tables and rules for a more efficient packet management.

Starting the daemons

At this stage the system is ready to restart the network and begging to use our new services.

```
sudo /etc/init.d/networking restart
sudo service hostapd start
sudo service dnsmasq start
```

The client application might be able now to find and connect to the traffic sign, however it will not receive any response to the traffic information requests. For that, we need to start also the TCP server daemon, which we have developed in Python.

```
python /home/pi/PythonProjects/TCPServer/LWire.py
```

And finally the last step on this configuration process is preparing the system to get automatically ready after the Raspberry Pi is rebooted. For services like hostapd and dnsmasq, they are have been automatically setup to be started with the operation system, however LWire belongs to the application layer, hence we need to perform a minor modification.

```
sudo nano /etc/rc.local
```

We propose to edit the rc.local file, which is automatically executed after the operating system has booted up. We just need to add the following lines to the file:

```
#!/bin/sh -e
python /home/pi/PythonProjects/TCPServer/LWire.py &
```

The configuration process explained on this document does not need to be completed on every Raspberry Pi. Once we get one of them successfully configured and running, we should be able to copy the following files to replicate the configuration on the rest of traffic signs:

/etc/dnsmasq.conf

/etc/hostapd/hostapd.conf

/default/hostapd

/usr/sbin/hostapd

/etc/network/interfaces

/etc/rc.local

UNIVERSIDAD POLITÉCNICA DE VALENCIA

Station.ini keyword reference

LWire description file

Latest document revision:

8/23/2015

The present document can be used as a reference to the station.ini configuration file. The file contains the specific data that defines each traffic sign, which is divided in two different sections.

Document Revision History			
Rev #	**Date**	**By**	**Revision Description**
1	24/08/2015	C. Fernández	Initial Revision of STATION INI Doc

Currently the document is divided in two sections STATION_DATA and TS. The first one contains information with the coordinates and SSID references of the traffic sign. This information is intended to be the footprint of the sign, so it can be used to avoid spoofing attacks. In this section we can find the following keywords:

SSID	
Description	It identifies the network name, which is defined by the hostapd.conf file. If there is a mismatch between these two files, we might be facing a spoofing case.
Section	STATION_DATA
Requirement	Mandatory
Example	WardAP

latitude	
Description	GPS coordinates where the traffic sign is located.
Section	STATION_DATA
Requirement	Optional
Example	39.27046

longitude	
Description	GPS coordinates where the traffic sign is located.
Section	STATION_DATA
Requirement	Optional
Example	-2.78451

altitude	
Description	Placeholder that identifies the third coordinate where traffic sign is located.
Section	STATION_DATA
Requirement	Not yet implemented
Example	234.0323

The TS section specifies the information sent to the client. The keywords under this section are described below:

msg	
Description	This keyword identifies the traffic sign (TS) by a four characters code follow by a null digit. For additional information to be sent along with the TS code, this should be added between the four first characters and the null digit, e.g a 50kph speed limitation would be represented by "R30150/0". For more than two TS to be sent at the same time the four-digit codes from each TS should be sent separated by a '/1' digit, e.g. a right turn prohibited sign follow by a traffic light indication should be coded as "R302/1P003/0".
Section	TS
Requirement	Mandatory
Example	P021/0

extra	
Description	The traffic sign is also able to provide extra information that is not necessarily related to the sign. This is a string parameter that will only be sent upon a "TX23" request after the information after msg has been delivered.
Section	TS
Requirement	Optional
Example	This is a message to the drive. Keep calm and carry on.

caption	
Description	This keyword is a brief indication to the driver, which will be included in the notification.
Section	TS
Requirement	Mandatory
Example	ESCUELA

notification	
Description	This keyword contains a more detailed explanation.
Section	TS
Requirement	Mandatory
Example	Peligro por la proximidad de un lugar frecuentado por niños, tales como escuelas, zona de juegos, etc.

severity	
Description	This parameter is used by the client to decide priorities on a scenario where more than one TS message needs to be prompted to the driver.
Section	TS
Requirement	Optional (default severity 0)
Example	4

sdownstrategy	
Description	Integer value that decides the strategy to cancel the notification to the user. The strategies supported at the moment are: # 0 - SSID is not visible by the client. # 1 - RSSI from the current hot spot has started to decrease. # 2 - New TS overrides. A new message should invalidate or update the current one. # 3 - Countdown timer. After the fix/variable number of seconds the notification is cancelled.
Section	TS
Requirement	Optional (default strategy 0)
Example	0

image	
Description	Bitmapped image that will be transmitted to the client if the database does not contain one for the code delivered after the “RX22” request.
Section	TS
Requirement	Optional
Example	… $EQRIT©¤ž–•uÛz-æalÌø-Œiþ >ÌÛ<qÞzÌjºLÕÕ--•J¦¢Hq————————————@¬ ÁÄ¾ç¾ß%"ÜÏ<xÜ%!Ö"eJ Aÿ™™…

TrafficSignDroid quick start guide

TrafficSignDroid is an application for Android 2.3.3+ that will allow you to detect and recognize any traffic sign equipped with wireless interface compatible with LWire. To install the application you just need to download the package from Google Play. The application will be ready on your smartphone or tablet in a few moments.

Please be aware that the present application requires the following permissions:

android.permission.INTERNET
Required to exchange data through the Wi-Fi interface.

android.permission.ACCESS_NETWORK_STATE
Required to determine the connection state.

android.permission.ACCESS_WIFI_STATE
Required to enable the adapter when it is disabled.

android.permission.CHANGE_WIFI_STATE
Required to determine the state of the adapter (enabled/disabled).

android.permission.WAKE_LOCK
Required to keep the screen on and unlocked.

android.permission.WRITE_EXTERNAL_STORAGE
Only for debugging purposes. To be removed on the final version.

android.permission.ACCESS_FINE_LOCATION
Only for debugging purposes. To be removed on the final version.

Once installed you can launch the application by touching the following icon:

The splash screen will be prompted while the system initializes the service. After that your device will be ready to receive data from traffic signs.

For better results place your device on a car mount at the vehicle windscreen.

August 5th 2015 Test results **20kph**

		Traffic Sign location		39.413515	-0.386901			
Vehicle speed(km/h)	Vehicle speed(m/s)	Number of TS detected	Time stamp	Latitud	Longitud	Event	Distance to hp (m)	Time Delta (ms)
N/A	N/A	0	11:53:48	39.41310145	-0.3853536	start Scan	140.66	N/A
0	0.05	0	11:53:51	39.41310187	-0.38535153	start Scan	140.82	2836
0	0.00	0	11:53:54	39.41310167	-0.38535152	start Scan	140.82	2793
0	0.00	0	11:53:57	39.41310158	-0.38535147	start Scan	140.83	2809
0	0.00	0	11:53:59	39.41310164	-0.38535146	start Scan	140.83	2808
0	0.00	0	11:54:03	39.41310167	-0.38535151	start Scan	140.82	3906
1	0.19	0	11:54:07	39.41312177	-0.3853521	start Scan	140.06	4030
11	3.17	0	11:54:11	39.41318651	-0.38547908	start Scan	127.50	3961
21	5.96	0	11:54:14	39.4132252	-0.38567353	start Scan	110.26	2891
21	5.96	0	11:54:17	39.41325931	-0.38587597	start Scan	92.53	2973
21	5.85	0	11:54:20	39.41329212	-0.38606572	start Scan	75.92	2840
19	5.23	0	11:54:21	39.41330455	-0.38613238	TS Found	70.05	1120
N/A	N/A	0	11:54:21	39.41330455	-0.38613238	TS connected	70.05	12
N/A	N/A	0	11:54:21	39.41330455	-0.38613238	TCP connected	70.05	46
N/A	N/A	0	11:54:21	39.41330455	-0.38613238	TCP request	70.05	118
N/A	N/A	1	11:54:21	39.41330455	-0.38613238	TCP response	70.05	76
N/A	N/A	1	11:54:21	39.41330455	-0.38613238	TCP disconnectec	70.05	83
20	5.64	1	11:54:44	39.41364279	-0.38756737	TS disassociated	-58.98	22898
N/A	N/A	1	11:55:17	39.41301886	-0.3853949	start Scan	140.66	N/A
0	0.09	1	11:55:20	39.41305605	-0.3853719	start Scan	140.93	2884
0	0.07	1	11:55:23	39.41306925	-0.38536795	start Scan	140.72	2794
1	0.16	1	11:55:25	39.41306383	-0.38537599	start Scan	140.29	2727
0	0.14	1	11:55:28	39.41306287	-0.38538141	start Scan	139.89	2867
1	0.29	1	11:55:31	39.41306289	-0.38537127	start Scan	140.70	2807
0	0.06	1	11:55:35	39.41306064	-0.38536926	start Scan	140.95	3897
4	1.21	1	11:55:39	39.41313018	-0.38539675	start Scan	136.13	3991
8	2.26	1	11:55:40	39.41314804	-0.38542507	TS Found	133.20	1297
N/A	N/A	1	11:55:40	39.41314804	-0.38542507	TS connected	133.20	32
N/A	N/A	1	11:55:40	39.41314804	-0.38542507	TCP connected	133.20	41
9	2.56	1	11:55:42	39.41316749	-0.38545266	TCP request	130.29	1136
N/A	N/A	2	11:55:42	39.41316749	-0.38545266	TCP response	130.29	327
N/A	N/A	2	11:55:42	39.41316749	-0.38545266	TCP disconnectec	130.29	19
19	5.19	2	11:56:19	39.41362732	-0.38763021	TS disassociated	-63.88	37378
N/A	N/A	2	11:59:03	39.41301274	-0.38541526	start Scan	139.32	N/A
3	0.74	2	11:59:06	39.41301944	-0.38538585	start Scan	141.35	2726
1	0.27	2	11:59:09	39.4130393	-0.38538464	start Scan	140.60	2765
0	0.01	2	11:59:12	39.41305158	-0.38537849	start Scan	140.58	2799
3	0.88	2	11:59:15	39.41311822	-0.38537903	start Scan	137.99	2927
12	3.44	2	11:59:18	39.41317847	-0.38547435	start Scan	128.15	2865
10	2.78	2	11:59:19	39.41318711	-0.38551866	TS Found	124.22	1410
N/A	N/A	2	11:59:19	39.41318711	-0.38551866	TS connected	124.22	44
128	35.43	2	11:59:19	39.41319852	-0.38557712	TCP connected	119.05	146
N/A	N/A	2	11:59:19	39.41319852	-0.38557712	TCP request	119.05	143
N/A	N/A	3	11:59:19	39.41319852	-0.38557712	TCP response	119.05	60
N/A	N/A	3	11:59:19	39.41319852	-0.38557712	TCP disconnectec	119.05	53
20	5.54	3	11:59:53	39.41366085	-0.38763635	TS disassociated	-65.22	33242
N/A	N/A	3	12:02:12	39.41298038	-0.38543418	start Scan	139.33	N/A
0	0.12	3	12:02:15	39.41302717	-0.3854028	start Scan	139.67	2747
0	0.14	3	12:02:18	39.41308171	-0.38536996	start Scan	140.07	2918
1	0.17	3	12:02:22	39.41310974	-0.38536536	start Scan	139.41	3863
1	0.40	3	12:02:24	39.41312762	-0.38536818	TS Found	138.55	2176
N/A	N/A	3	12:02:24	39.41312762	-0.38536818	TS connected	138.55	4
N/A	N/A	3	12:02:24	39.41312762	-0.38536818	TCP connected	138.55	10
36	9.96	3	12:02:24	39.41314756	-0.38538656	TCP request	136.37	219
N/A	N/A	4	12:02:24	39.41314756	-0.38538656	TCP response	136.37	112
N/A	N/A	4	12:02:25	39.41314756	-0.38538656	TCP disconnectec	136.37	210
20	5.58	4	12:03:00	39.41364285	-0.38757009	TS disassociated	-59.21	35036
N/A	N/A	4	12:04:01	39.41309219	-0.38536469	start Scan	140.10	N/A
4	1.05	4	12:04:05	39.41316728	-0.38539182	start Scan	135.29	4564
14	3.76	4	12:04:08	39.41319202	-0.38551191	start Scan	124.62	2840
18	5.03	4	12:04:11	39.41322659	-0.38567505	start Scan	110.09	2889
18	5.01	4	12:04:14	39.4132611	-0.38583779	start Scan	95.60	2891
22	6.13	4	12:04:17	39.41330533	-0.38603079	start Scan	78.31	2821
21	5.86	4	12:04:20	39.41334881	-0.38621662	start Scan	61.63	2846
20	5.63	4	12:04:23	39.4133869	-0.3864022	start Scan	45.16	2924
17	4.80	4	12:04:24	39.41339832	-0.38646653	TS Found	39.51	1176
N/A	N/A	4	12:04:24	39.41339832	-0.38646653	TS connected	39.51	15
N/A	N/A	4	12:04:24	39.41339832	-0.38646653	TCP connected	39.51	33
78	21.68	4	12:04:24	39.41340988	-0.3865263	TCP request	34.25	243
N/A	N/A	5	12:01:29	39.41340988	-0.3865263	TCP response	34.25	154
N/A	N/A	5	12:01:29	39.41340988	-0.3865263	TCP disconnectec	34.25	38
21	5.82	5	12:01:29	39.41367522	-0.38769785	TS disassociated	-70.74	18052

				August 5th 2015 Test results				30kph
		Traffic Sign location		39.413515	-0.386901			
Vehicle speed(km/h)	Vehicle speed(m/s)	Number of TS detected	Time stamp	Latitud	Longitud	Event	Distance to hp (m)	Time Delta (ms)
N/A	N/A	0	80073	39.41294651	-0.38540167	start Scan	143.48	N/A
0	0.06	0	82970	39.41301094	-0.38536569	start Scan	143.31	2897
3	0.76	0	87191	39.41315064	-0.38533963	start Scan	140.12	4221
4	1.19	0	89944	39.41316606	-0.38537344	start Scan	136.84	2753
2	0.48	0	92840	39.41315002	-0.38539686	start Scan	135.44	2896
4	1.25	0	95741	39.41313896	-0.38535716	start Scan	139.06	2901
5	1.37	0	98585	39.41313352	-0.38531186	start Scan	142.96	2844
6	1.61	0	101439	39.41314953	-0.38536153	start Scan	138.36	2854
28	7.75	0	104403	39.4132126	-0.38561601	start Scan	115.40	2964
31	8.57	0	106419	39.41325037	-0.38581128	TS Found	98.13	2016
N/A	N/A	0	106454	39.41325037	-0.38581128	TS connected	98.13	35
N/A	N/A	0	106528	39.41325037	-0.38581128	TCP connected	98.13	74
N/A	N/A	0	106813	39.41325037	-0.38581128	TCP request	98.13	285
N/A	N/A	1	107104	39.41325037	-0.38581128	TCP response	98.13	291
N/A	N/A	1	107162	39.41325037	-0.38581128	TCP disconnected	98.13	58
28	7.83	1	128015	39.41365177	-0.38763851	TS disassociated	-65.16	20853
N/A	N/A	1	165487	39.41307138	-0.38535842	start Scan	141.40	N/A
2	0.54	1	168328	39.41310826	-0.38536035	start Scan	139.87	2841
5	1.43	1	171135	39.41313929	-0.3853961	start Scan	135.86	2807
22	6.21	1	172397	39.41317028	-0.38547899	TS Found	128.03	1262
N/A	N/A	1	172466	39.41317028	-0.38547899	TS connected	128.03	69
N/A	N/A	1	172498	39.41317028	-0.38547899	TCP connected	128.03	32
N/A	N/A	1	172619	39.41317028	-0.38547899	TCP request	128.03	121
N/A	N/A	2	172768	39.41317028	-0.38547899	TCP response	128.03	149
N/A	N/A	2	172822	39.41317028	-0.38547899	TCP disconnected	128.03	54
28	7.79	2	197820	39.41366499	-0.38765191	TS disassociated	-66.63	24998
N/A	N/A	2	251016	39.4131348	-0.38525108	start Scan	147.91	N/A
13	3.57	2	253883	39.41315458	-0.38536784	start Scan	137.67	2867
42	11.63	2	256731	39.4132289	-0.38574165	start Scan	104.55	2848
31	8.68	2	259515	39.41329649	-0.38600913	start Scan	80.38	2784
17	4.60	2	260811	39.41331492	-0.38607435	TS Found	74.42	1296
N/A	N/A	2	260874	39.41331492	-0.38607435	TS connected	74.42	63
N/A	N/A	2	260896	39.41331492	-0.38607435	TCP connected	74.42	22
N/A	N/A	2	260984	39.41331492	-0.38607435	TCP request	74.42	88
166	46.08	3	261159	39.41334398	-0.38616101	TCP response	66.35	175
N/A	N/A	3	261267	39.41334398	-0.38616101	TCP disconnected	66.35	108
29	8.13	3	277767	39.41363771	-0.3876747	TS disassociated	-67.85	16500
N/A	N/A	3	3587	39.41307438	-0.38531653	start Scan	144.67	N/A
10	2.73	3	7953	39.41316117	-0.3854252	start Scan	132.75	4366
29	7.96	3	12868	39.41326685	-0.38585938	start Scan	93.64	4915
28	7.79	3	13995	39.41328708	-0.3859582	TS Found	84.87	1127
N/A	N/A	3	14015	39.41328708	-0.3859582	TS connected	84.87	20
N/A	N/A	3	14089	39.41328708	-0.3859582	TCP connected	84.87	74
N/A	N/A	3	14374	39.41328708	-0.3859582	TCP request	84.87	285
N/A	N/A	4	14537	39.41328708	-0.3859582	TCP response	84.87	163
N/A	N/A	4	14582	39.41328708	-0.3859582	TCP disconnected	84.87	45
30	8.26	4	35253	39.41369281	-0.38787288	TS disassociated	-85.80	20671
N/A	N/A	7	65596	39.41309179	-0.38532073	start Scan	143.68	N/A
10	2.68	8	70746	39.41316133	-0.38545993	start Scan	129.90	5150
27	7.55	9	73665	39.41322001	-0.38570481	start Scan	107.87	2919
29	7.95	10	76563	39.41327412	-0.38596416	start Scan	84.82	2898
28	7.80	11	79427	39.41332852	-0.38621474	start Scan	62.50	2864
26	7.28	12	82258	39.41337536	-0.38644821	start Scan	41.88	2831
26	7.26	13	85043	39.41343531	-0.38667077	start Scan	21.67	2785
26	7.11	14	87799	39.41349629	-0.38690092	start Scan	2.08	2756
N/A	N/A	15	88766	39.41349629	-0.38690092	TS Found	2.08	967
N/A	N/A	16	88829	39.41349629	-0.38690092	TS connected	2.08	63
339	94.15	17	88929	39.41351073	-0.3869862	TCP connected	-7.33	100
N/A	N/A	18	89053	39.41351073	-0.3869862	TCP request	-7.33	124
N/A	N/A	19	89634	39.41351073	-0.3869862	TCP response	-7.33	581
N/A	N/A	20	89674	39.41351073	-0.3869862	TCP disconnected	-7.33	40
28	7.90	21	99303	39.4136807	-0.38784739	TS disassociated	-83.36	9629

August 5th 2015 Test results **40kph**

		Traffic Sign location		39.413515	-0.386901			
Vehicle speed(km/h)	Vehicle speed(m/s)	Number of TS detected	Time stamp	Latitud	Longitud	Event	Distance to hp (m)	Time Delta (ms)
N/A	N/A	0	3333	39.41305383	-0.38505061	start Scan	167.03	N/A
0	0.03	0	6169	39.41305575	-0.38504888	start Scan	167.11	2836
0	0.12	0	7518	39.41305792	-0.38504596	TS Found	167.27	1349
N/A	N/A	0	7520	39.41305792	-0.38504596	TS connected	167.27	2
N/A	N/A	0	7574	39.41305792	-0.38504596	TCP connected	167.27	54
1	0.31	0	9091	39.41306081	-0.38505048	TCP request	166.80	1517
21	5.86	1	13877	39.41311904	-0.38536948	TCP response	138.74	4786
N/A	N/A	1	13919	39.41311904	-0.38536948	TCP disconnected	138.74	42
33	9.16	1	39187	39.41369508	-0.38795396	TS disassociated	-92.65	25268
N/A	N/A	1	79719	39.41310989	-0.38521278	start Scan	151.87	N/A
5	1.47	1	84561	39.41313998	-0.38528732	start Scan	144.76	4842
33	9.21	1	88697	39.41322537	-0.38571727	TS Found	106.67	4136
N/A	N/A	1	88713	39.41322537	-0.38571727	TS connected	106.67	16
N/A	N/A	1	88725	39.41322537	-0.38571727	TCP connected	106.67	12
N/A	N/A	1	88885	39.41322537	-0.38571727	TCP request	106.67	160
N/A	N/A	2	89127	39.41322537	-0.38571727	TCP response	106.67	242
N/A	N/A	2	89163	39.41322537	-0.38571727	TCP disconnected	106.67	36
39	10.84	2	106706	39.41368981	-0.38784714	TS disassociated	-83.57	17543
N/A	N/A	2	23137	39.41303222	-0.38477823	start Scan	190.10	N/A
0	0.02	2	25883	39.4130276	-0.38478571	start Scan	189.63	25883
0	0.03	2	28641	39.41300159	-0.38479815	start Scan	189.46	5504
1	0.39	2	31494	39.41301262	-0.38482045	start Scan	187.26	5611
13	3.72	2	35422	39.41307523	-0.38510258	start Scan	162.05	6781
16	4.56	2	38184	39.41314709	-0.38544607	start Scan	131.51	6690
21	5.82	2	41124	39.41323127	-0.38581661	start Scan	98.35	5702
19	5.37	2	44009	39.41330264	-0.38617029	start Scan	67.07	5825
9	2.63	2	45113	39.41332621	-0.38628929	TS Found	56.59	3989
N/A	N/A	2	45143	39.41332621	-0.38628929	TS connected	56.59	1134
N/A	N/A	2	45208	39.41332621	-0.38628929	TCP connected	56.59	95
N/A	N/A	2	45431	39.41332621	-0.38628929	TCP request	56.59	288
N/A	N/A	3	45598	39.41332621	-0.38628929	TCP response	56.59	390
N/A	N/A	3	45709	39.41332621	-0.38628929	TCP disconnected	56.59	278
38	10.56	3	57474	39.41364537	-0.38767	TS disassociated	-67.63	11765
N/A	N/A	3	94439	39.41304507	-0.38476427	start Scan	190.85	N/A
1	0.42	3	97198	39.41302042	-0.38478756	start Scan	189.71	2759
6	1.53	3	99980	39.41304099	-0.38483136	start Scan	185.45	2782
14	3.89	3	102698	39.41306712	-0.38494968	start Scan	174.87	2718
33	9.16	3	105504	39.41313348	-0.38523614	start Scan	149.18	2806
44	12.21	3	108324	39.41321629	-0.3856223	start Scan	114.76	2820
42	11.67	3	111280	39.41329315	-0.38601182	start Scan	80.27	2956
39	10.71	3	112331	39.41331786	-0.38613928	TS Found	69.01	1051
N/A	N/A	3	112342	39.41331786	-0.38613928	TS connected	69.01	11
N/A	N/A	3	112373	39.41331786	-0.38613928	TCP connected	69.01	31
N/A	N/A	3	112689	39.41331786	-0.38613928	TCP request	69.01	316
341	94.76	4	112806	39.41334407	-0.38626406	TCP response	57.92	117
N/A	N/A	4	112831	39.41334407	-0.38626406	TCP disconnected	57.92	25
39	10.83	4	125980	39.41369767	-0.38785626	TS disassociated	-84.54	13149
N/A	N/A	4	166219	39.41299812	-0.38476476	start Scan	192.31	N/A
2	0.46	4	168957	39.41300282	-0.38477806	start Scan	191.06	2738
20	5.59	4	171738	39.41306431	-0.38494284	start Scan	175.53	2781
25	6.96	4	174621	39.41309178	-0.3851761	start Scan	155.48	2883
46	12.91	4	177495	39.41319174	-0.38558823	start Scan	118.37	2874
44	12.19	4	180404	39.41327133	-0.38598891	start Scan	82.91	2909
38	10.61	4	183225	39.41333564	-0.38632974	start Scan	52.97	2821
25	6.89	4	184557	39.41335533	-0.38643501	TS Found	43.79	1332
N/A	N/A	4	184580	39.41335533	-0.38643501	TS connected	43.79	23
N/A	N/A	4	184621	39.41335533	-0.38643501	TCP connected	43.79	41
N/A	N/A	4	184691	39.41335533	-0.38643501	TCP request	43.79	70
220	61.23	5	184848	39.41337763	-0.38654508	TCP response	34.18	157
N/A	N/A	5	184936	39.41337763	-0.38654508	TCP disconnected	34.18	88
30	8.25	5	199270	39.41369243	-0.38785239	TS disassociated	-84.08	14334

				August 5th 2015 Test results				50kph
		Traffic Sign location		39.413515	-0.386901			
Vehicle speed(km/h)	Vehicle speed(m/s)	Number of TS detected	Time stamp	Latitud	Longitud	Event	Distance to hp (m)	Time Delta (ms)
N/A	N/A	0	12:38:34	39.41293722	-0.38438397	start Scan	225.57	N/A
2	0.52	0	12:38:37	39.41291674	-0.38440972	start Scan	224.12	2777
0	0.06	0	12:38:39	39.41291655	-0.38441169	start Scan	223.97	2735
13	3.67	0	12:38:42	39.41295304	-0.38452417	start Scan	213.54	2843
23	6.32	0	12:38:45	39.41299333	-0.38472466	start Scan	195.76	2811
42	11.69	0	12:38:48	39.41307146	-0.3850907	start Scan	163.15	2789
48	13.45	0	12:38:51	39.41316678	-0.38552022	start Scan	124.78	2852
41	11.49	0	12:38:53	39.41323128	-0.38581294	TS Found	98.65	2274
N/A	N/A	0	12:38:53	39.41323128	-0.38581294	TS connected	98.65	15
N/A	N/A	0	12:38:53	39.41323128	-0.38581294	TCP connected	98.65	4
N/A	N/A	0	12:38:53	39.41323128	-0.38581294	TCP request	98.65	98
190	52.68	1	12:38:53	39.41326737	-0.38594796	TCP response	86.38	233
N/A	N/A	1	12:38:53	39.41326737	-0.38594796	TCP disconnected	86.38	59
36	10.09	1	12:39:10	39.41365828	-0.38780731	TS disassociated	-79.47	16441
N/A	N/A	1	12:39:58	39.41294363	-0.38444203	start Scan	220.59	N/A
3	0.80	1	12:40:01	39.41295223	-0.38446588	start Scan	218.35	2807
12	3.31	1	12:40:04	39.41297843	-0.38456912	start Scan	209.02	2822
27	7.46	1	12:40:07	39.41301807	-0.38480621	start Scan	188.25	2786
42	11.55	1	12:40:09	39.41308016	-0.38516871	start Scan	156.47	2752
49	13.68	1	12:40:12	39.41316896	-0.38560588	start Scan	117.73	2833
45	12.38	1	12:40:15	39.41326949	-0.3860016	start Scan	81.95	2889
35	9.66	1	12:40:16	39.41330062	-0.38613665	TS Found	69.86	1252
N/A	N/A	1	12:40:16	39.41330062	-0.38613665	TS connected	69.86	8
N/A	N/A	1	12:40:17	39.41330062	-0.38613665	TCP connected	69.86	104
N/A	N/A	1	12:40:17	39.41330062	-0.38613665	TCP request	69.86	218
N/A	N/A	2	12:40:17	39.41330062	-0.38613665	TCP response	69.86	139
N/A	N/A	2	12:40:17	39.41330062	-0.38613665	TCP disconnected	69.86	35
40	11.09	2	12:40:32	39.41371472	-0.38796626	TS disassociated	-94.17	14794
N/A	N/A	2	12:42:27	39.41294292	-0.38426127	start Scan	235.53	N/A
0	0.00	2	12:42:30	39.41294156	-0.38426189	start Scan	235.52	2767
1	0.20	2	12:42:33	39.41291035	-0.38428069	start Scan	234.93	2872
1	0.33	2	12:42:36	39.4128996	-0.38429637	start Scan	233.99	2877
7	1.87	2	12:42:39	39.41291748	-0.38435398	start Scan	228.67	2842
27	7.38	2	12:42:42	39.41297387	-0.38459267	start Scan	207.23	2907
40	11.01	2	12:42:45	39.41305006	-0.38494229	start Scan	176.03	2835
51	14.06	2	12:42:47	39.41314446	-0.38538367	start Scan	136.71	2796
49	13.59	2	12:42:50	39.41324063	-0.38581529	start Scan	98.13	2839
35	9.70	2	12:42:52	39.41327022	-0.38596358	TS Found	85.01	1353
N/A	N/A	2	12:42:52	39.41327022	-0.38596358	TS connected	85.01	24
N/A	N/A	2	12:42:52	39.41327022	-0.38596358	TCP connected	85.01	52
N/A	N/A	2	12:42:52	39.41327022	-0.38596358	TCP request	85.01	104
N/A	N/A	3	12:42:52	39.41327022	-0.38596358	TCP response	85.01	199
N/A	N/A	3	12:42:52	39.41327022	-0.38596358	TCP disconnected	85.01	50
47	12.97	3	12:43:06	39.41372327	-0.38794884	TS disassociated	-92.95	13716
N/A	N/A	3	12:43:51	39.4129157	-0.38439567	start Scan	225.31	N/A
4	0.98	3	12:43:54	39.41292716	-0.38442498	start Scan	222.53	2834
21	5.73	3	12:43:57	39.41297557	-0.38460714	start Scan	205.99	2884
25	7.05	3	12:44:00	39.4130197	-0.38482666	start Scan	186.52	2763
45	12.58	3	12:44:04	39.41314491	-0.38537954	start Scan	137.03	3933
47	13.13	3	12:44:07	39.41323318	-0.38580167	start Scan	99.50	2859
46	12.90	3	12:44:09	39.41332694	-0.38621696	start Scan	62.37	2879
49	13.59	3	12:44:12	39.41341639	-0.38665681	start Scan	23.67	2848
50	13.92	3	12:44:15	39.41351098	-0.38707455	start Scan	-14.92	2773
47	13.06	3	12:44:18	39.41361324	-0.38749717	start Scan	-52.37	2868
30	8.43	3	12:44:21	39.41367444	-0.38775821	start Scan	-75.74	2774
N/A	N/A	3	12:45:56	39.41291144	-0.38438529	start Scan	226.30	N/A
1	0.33	3	12:45:58	39.4129451	-0.38438312	start Scan	225.40	2737
1	0.24	3	12:46:01	39.41291424	-0.38440309	start Scan	224.75	2752
1	0.23	3	12:46:04	39.41291073	-0.38441238	start Scan	224.10	2785
13	3.54	3	12:46:08	39.41295859	-0.3845875	start Scan	208.15	4500
37	10.31	3	12:46:13	39.4130676	-0.38506163	start Scan	165.66	4121
55	15.15	3	12:46:17	39.41324439	-0.3858204	start Scan	97.59	4494
51	14.16	3	12:46:20	39.41333482	-0.38627125	start Scan	57.69	2818
32	8.94	3	12:46:23	39.41339814	-0.38656354	start Scan	31.77	2901
49	13.66	3	12:46:26	39.41350887	-0.38698162	start Scan	-6.96	2835
32	9.02	3	12:46:35	39.41366923	-0.38793368	start Scan	-90.36	9247
5	1.31	3	12:46:38	39.41357141	-0.38790805	start Scan	-86.74	2759

August 5th 2015 Test results **60kph**

		Traffic Sign location		39.413515	-0.386901			
Vehicle speed(km/h)	Vehicle speed(m/s)	Number of TS detected	Time stamp	Latitud	Longitud	Event	Distance to hp (m)	Time Delta (ms)
N/A	N/A	0	3723	39.41294181	-0.38430221	start Scan	232.18	N/A
0	0.00	0	6440	39.41294151	-0.38430244	start Scan	232.17	2717
0	0.09	0	9206	39.41293918	-0.38430624	start Scan	231.92	2766
2	0.45	0	11996	39.41292507	-0.38432678	start Scan	230.67	2790
1	0.18	0	14797	39.41292884	-0.3843192	start Scan	231.18	2801
10	2.68	0	17618	39.4129528	-0.3844018	start Scan	223.62	2821
29	8.12	0	20459	39.41300227	-0.38466274	start Scan	200.56	2841
23	6.38	0	23334	39.41304802	-0.38486798	start Scan	182.21	2875
51	14.04	0	31343	39.41331503	-0.38613146	start Scan	69.75	8009
57	15.84	0	33271	39.41338779	-0.38647524	TS Found	39.22	1928
N/A	N/A	0	33316	39.41338779	-0.38647524	TS connected	39.22	45
N/A	N/A	0	33353	39.41338779	-0.38647524	TCP connected	39.22	37
412	114.42	0	33479	39.4134242	-0.38663733	TCP request	24.80	126
N/A	N/A	1	33953	39.4134242	-0.38663733	TCP response	24.80	474
N/A	N/A	1	34115	39.4134242	-0.38663733	TCP disconnected	24.80	162
44	12.11	1	42728	39.41367731	-0.38780279	TS disassociated	-79.54	8613
N/A	N/A	1	91476	39.4129449	-0.38428818	start Scan	233.24	N/A
3	0.75	1	94239	39.41292084	-0.3843225	start Scan	231.16	2763
0	0.06	1	97094	39.41290973	-0.38432478	start Scan	231.32	2855
9	2.48	1	99933	39.41293721	-0.38439982	start Scan	224.27	2839
31	8.56	1	102897	39.41299428	-0.38468589	start Scan	198.91	2964
49	13.48	1	105654	39.41308815	-0.38510118	start Scan	161.74	2757
60	16.75	1	108567	39.41320693	-0.38564802	start Scan	112.96	2913
54	15.11	1	109651	39.41324363	-0.38583305	TS Found	96.58	1084
N/A	N/A	1	109659	39.41324363	-0.38583305	TS connected	96.58	8
N/A	N/A	1	109720	39.41324363	-0.38583305	TCP connected	96.58	61
N/A	N/A	1	110036	39.41324363	-0.38583305	TCP request	96.58	316
N/A	N/A	2	110268	39.41324363	-0.38583305	TCP response	96.58	232
N/A	N/A	2	110295	39.41324363	-0.38583305	TCP disconnected	96.58	27
43	11.81	2	126429	39.41357983	-0.38799136	TS disassociated	-93.95	16134
N/A	N/A	2	168951	39.41292501	-0.38431439	start Scan	231.69	N/A
1	0.19	2	171748	39.41289617	-0.38433215	start Scan	231.16	2797
4	1.24	2	174653	39.41291394	-0.38436879	start Scan	227.57	2905
22	5.98	2	177485	39.4129577	-0.38455751	start Scan	210.65	2832
24	6.78	2	180335	39.41300299	-0.38477492	start Scan	191.31	2850
49	13.55	2	183190	39.41310186	-0.38520674	start Scan	152.63	2855
59	16.32	2	186080	39.41321853	-0.38573506	start Scan	105.45	2890
50	13.84	2	187213	39.41325249	-0.3859129	TS Found	89.76	1133
N/A	N/A	2	187224	39.41325249	-0.3859129	TS connected	89.76	11
N/A	N/A	2	187261	39.41325249	-0.3859129	TCP connected	89.76	37
184	51.07	2	187557	39.41328711	-0.38608368	TCP request	74.65	296
N/A	N/A	3	187926	39.41328711	-0.38608368	TCP response	74.65	369
N/A	N/A	3	187972	39.41328711	-0.38608368	TCP disconnected	74.65	46
46	12.86	3	200807	39.41368685	-0.38792963	TS disassociated	-90.41	12835
N/A	N/A	3	247314	39.41292751	-0.38427249	start Scan	235.07	N/A
1	0.28	3	250195	39.41293933	-0.384278	start Scan	234.25	2881
0	0.07	3	253001	39.41291075	-0.38429125	start Scan	234.05	2806
5	1.38	3	255814	39.41291707	-0.38433602	start Scan	230.16	2813
18	5.11	3	258707	39.41295431	-0.38450128	start Scan	215.38	2893
34	9.58	3	261528	39.4130241	-0.38480245	start Scan	188.36	2821
31	8.63	3	264289	39.41308651	-0.38506763	start Scan	164.55	2761
56	15.58	3	267166	39.41319341	-0.38557085	start Scan	119.73	2877
52	14.33	3	268293	39.41323212	-0.3857522	TS Found	103.58	1127
N/A	N/A	3	268300	39.41323212	-0.3857522	TS connected	103.58	7
N/A	N/A	3	268313	39.41323212	-0.3857522	TCP connected	103.58	13
198	55.01	3	268611	39.41327349	-0.38593541	TCP request	87.19	298
N/A	N/A	4	268917	39.41327349	-0.38593541	TCP response	87.19	306
N/A	N/A	4	268945	39.41327349	-0.38593541	TCP disconnected	87.19	28
49	13.61	4	281971	39.41369773	-0.38792262	TS disassociated	-90.09	13026
N/A	N/A	4	33322	39.41293976	-0.38435603	start Scan	227.80	N/A
1	0.16	4	37149	39.41291735	-0.38437237	start Scan	227.17	3827
2	0.58	4	40994	39.41293344	-0.38439341	start Scan	224.92	3845
12	3.37	4	44886	39.41295907	-0.38454277	start Scan	211.81	3892
32	8.78	4	47736	39.41301772	-0.38482425	start Scan	186.78	2850
50	13.85	4	50538	39.41310036	-0.38526438	start Scan	147.96	2802
61	17.08	4	53372	39.4132237	-0.38580504	start Scan	99.57	2834
54	14.91	4	54468	39.41326792	-0.38598656	TS Found	83.22	1096
N/A	N/A	4	54499	39.41326792	-0.38598656	TS connected	83.22	31
N/A	N/A	4	54562	39.41326792	-0.38598656	TCP connected	83.22	63
N/A	N/A	4	54868	39.41326792	-0.38598656	TCP request	83.22	306
98	27.10	5	55433	39.41330611	-0.38615814	TCP response	67.91	565
N/A	N/A	5	55525	39.41330611	-0.38615814	TCP disconnected	67.91	92
44	12.19	5	67054	39.4136592	-0.38772505	TS disassociated	-72.59	11529