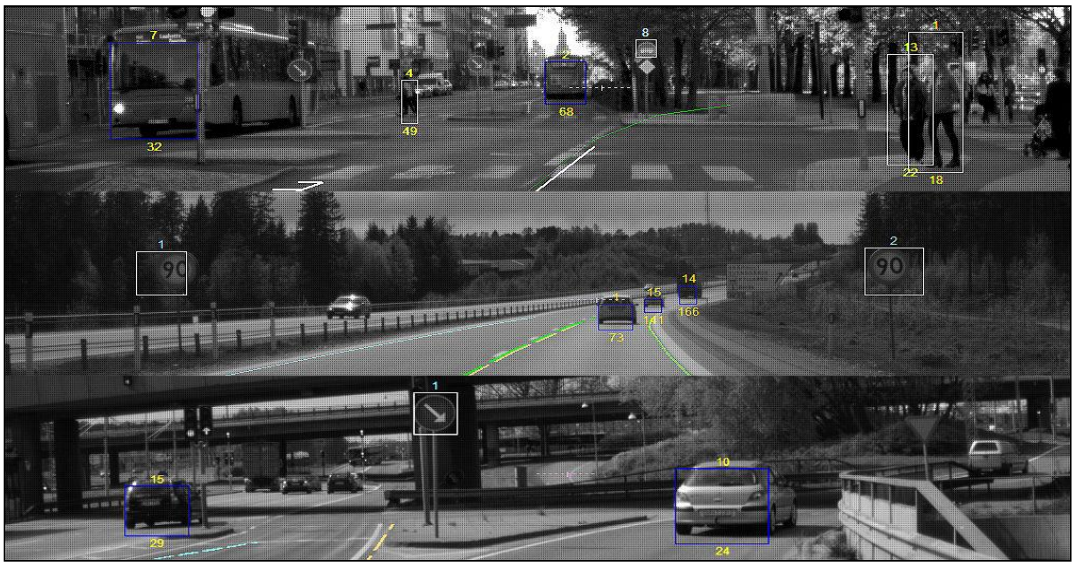
CHALMERS



Synthesising Data from Subsystems in an Active Safety System

Design of a Data Processing Tool

Master of Science Thesis in Computer Systems and Networks

Alvean Ekman
Daniel Axman

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, March 2015

Synthesising Data from Subsystems in an Active Safety System . Design of a Data Processing Tool

ALVEAN EKMAN
DANIEL AXMAN

Examiner: Aarne Ranta

Chalmers University of Technology
Department of Computer Science & Engineering
SE-412 96 Gothenburg, 2014
Sweden
Telephone +46 (0)31-772 1000

Cover: Courtesy of Delphi Automotive.

This thesis has been prepared using LaTeX.

Department of Computer Science & Engineering
Gothenburg, Sweden, March 18, 2015

Abstract

This project proposes a design of a data processing tool for a Traffic Sign Recognition (TSR) system, as a part of an Active Safety System that utilizes a front-mounted radar-camera set-up. Data from other subsystems that contribute to completing processed data from a utility perspective is aggregated as well. The tool is to be used for synthesising large amounts of data from log files generated during post-analysis of test drives, thereby increasing efficiency of testing and verification of the system.

A Python script functions as a connector between MATLAB structures storing information recorded during the test drives and a storage database. Data of interest is extracted from the MATLAB structures generated in post-analysis of test drives, then handled by a Python script that opens a connection to a PostgreSQL database in which the data is inserted. Conclusions concerning the accuracy of the TSR system can be drawn from evaluating the data in the database.

This thesis contributes to improvement of the current testing methods of the above mentioned systems, and hence utilises some existing procedures and functions at Delphi Automotive. The tool uses previously recorded and classified data for comparison, and is not intended for stand-alone use. Processing of data is not performed real-time in the vehicle, but during analysis of the data afterwards. The prototype tool is developed in connection to Swedish traffic. The test drives are performed under applied conditions, the input to the system is not simulated. The purpose of this thesis is to design a prototype tool to fulfil the above requirements, a complete tool is thus not implemented.

Furthermore, the systems handling extraction and analysis of data in the vehicle are treated as black boxes. There are protected material which cannot be accessed for the purpose of a thesis, and thus the information obtained from these systems cannot be dissected as to how they came to these conclusions.

The conclusion of this project is that it is advisable to continue development of such a tool as it can improve existing testing and verification procedures. There are a number of areas that need to be addressed, for which recommendations are given, none of which are deemed impossible to solve. This thesis suggests a design of a data processing a tool and a number of functionalities that can be implemented in a future project.

Acknowledgements

We would like to express our gratitude to our supervisor at Delphi Automotive subdivision Electronics & Safety in Gothenburg; Ted Henriksson for his help and support throughout the project. We would also like to thank Andreas Nilsson from the same team for his guidance and advice.

We feel very lucky to have been given the opportunity to do our master thesis at Delphi Automotive subdivision Electronics & Safety in Gothenburg, and would like to extend out thanks to Sarah Wills and Mats Björnerbäck for granting us the chance to work on this project as well as providing us with the facilities and tools necessary to complete this project.

We would like to thank our examiner Professor Aarne Ranta of the Computer Science department at the University of Gothenburg, for his time spent and input given regarding this project.

Finally we would like to express appreciation and give thanks to our supervisor at Chalmers University of Technology; Professor Graham Kemp, for his unwavering support and advice during this thesis and for his guidance in the bureaucratic jungle of thesis work.

Alvean Ekman and Daniel Axman, Gothenburg Wednesday 18th March, 2015

Contents

Abbreviations and Acronyms

ABS Anti-lock Braking System

ACC Adaptive Cruise Control

ADAS Advanced Driver Assistance Systems

AHS Automated Highway System

ASS Active Safety Systems

CAN Controller Area Network

DVtool Data Visulisation Tool

GM General Motors

GPS Global Positioning System

HDF5 Hierarchical Data Format 5

ORM Object-Relational Mapping

RCA Radio Corporation of America

ReSim Resimulation

ROI Region of Interest

SWC Software Component

TSR Traffic Sign Recognition

1

Introduction

In the automotive industry there is an ever present goal of a safer traffic environment for everyone, not just those protected inside their cars. For a long time *safety* meant how well a vehicle could withstand an impact with another vehicle at a particular speed, focusing on prevention of serious injury to the driver and passengers - this includes among other things a crash frame, seatbelts and airbags. Today these types of components are referred to as *Passive Safety Systems*, but a pedestrian crossing a street and getting hit by a car today will sustain the same injuries as he would have in the 1960's, hence it lies in the interest of the manufacturers to strive towards avoiding accidents, thereby creating a safer environment for everyone.

With the advancements in technology where circuits and chips are becoming ever smaller, modern cars carry an immense amount of computational power. The computers already control a wide range of functions; e.g. computing the estimated mileage out of the amount of petrol in the tank, managing the climate control, notifying the driver if there is something wrong with the engine, or even if a light bulb in a headlight is broken. These systems, while neither simple nor easy to design, are quite straightforward in their basic objectives in the sense that they are meant to assist the driver in various situations.

With the increasing amount of computational power available, vehicle systems have started to contribute to other areas as well including the safety area. Systems that take direct action in order to prevent or avoid accidents are referred to as *Active Safety Systems (ASS)*, which includes Anti-lock Braking System (ABS) and electronic stability control for example. Advanced Driver Assistance Systems (ADAS) is considered a part of ASS, but requires additional parts in the form of a radar and a visual input (see Section 2.1). These systems are connected to the car's mechanical systems (e.g. brakes and steering) and can take direct control of the vehicle if needed. The primary function of these systems are to react before the driver has had time to detect an immediate danger, e.g. pre-charging the brake system for faster response and thus shorter braking distance, or keeping the vehicle on the road in case of a driver falling asleep at the wheel.

Functioning *Passive Safety Systems* help limit the effects of an accident once it happens but a vehicle with *Active Safety Systems* could help avoid an accident altogether, therefore development of these systems are an important contribution to the environment on and around our roads.

1.1 Context

ADAS can potentially save lives when properly implemented and exhibiting correct behaviour but between the complexity of the systems, the vast number of parameters and the slim differences between a dangerous scenario and a safe one, mistakes can be made in several phases of development with potentially devastating consequences. If a climate control system sets the temperature wrong by a degree or two, it makes no significant difference to the driver, but if the system connected to the brakes suddenly interprets a safe environment as dangerous it could activate them at the wrong time, potentially causing accidents instead of preventing them. These scenarios are not taken lightly by companies developing these systems, nor the companies employing them, therefore testing these types of systems rigorously before deploying them in live circumstances is absolutely necessary.

Testing is usually performed by having drivers drive cars with testing equipment installed, referred to as "test cars" (see Section 2.2), for several hundreds of thousands of kilometres. The equipment installed is connected to the vehicle's Controller Area Network (CAN) and consists of sensors, camera, radar etc. In Figure 1.1 there are examples of what the vehicle systems can detect. For example in the top left corner a bus has been detected and boxed in in a blue square, which means the system has classified the object as a vehicle. It has also been assigned the id 7, which is an identifier to help developers and testers in keeping track of what data belong to which object. Also, in the top right corner a couple of pedestrians have been similarly indicated, but with white boxes instead of blue, indicating the system classifies these objects as 'other', as in them not being vehicles and finally in the middle picture two speed limit signs have been detected and similarly boxed in and given id numbers. The tool can also detect lanes as seen as lines along the lane markers.

The tool is not connected to the vehicle's mechanical systems, instead all information is logged and saved to a hard drive for post-analysis. Examples of information saved are all the system's parameters, conjectures regarding detected objects and outputs. Conjectures are conclusions based on incomplete information, typical conjectures the Traffic Sign Recognition (TSR) make are classifications of obstructed traffic signs, i.e. classifications based on incomplete data.

When the car returns, the hard drive is taken out of the car and the log files are subjected to post-analysis, which is where various problems are usually found. If a tester finds an error in the system by examining a log file, he can apply a fix and simulate the test drive with the same input data to verify that the error has been solved, this is referred to as performing a Resimulation (ReSim). Being able to ReSim entire test drives in this fashion almost entirely removes the need for actually repeating test drives

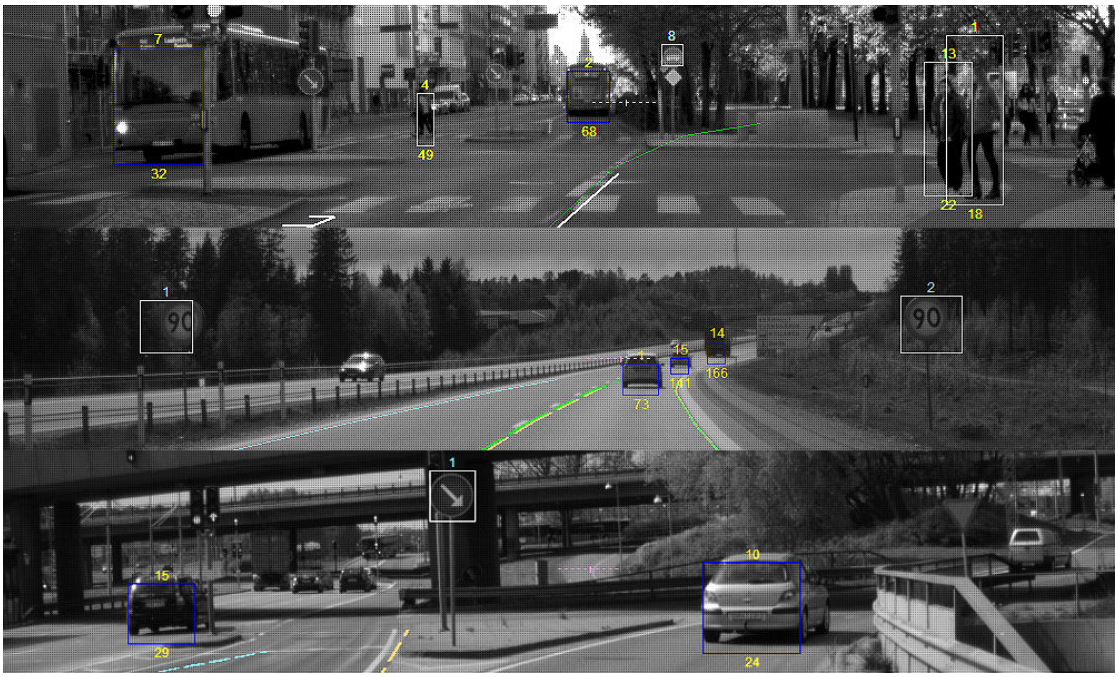


Figure 1.1: Three examples of what the systems at Delphi Automotive identifies. Picture courtesy of Delphi Automotive.

due to bug fixes or software updates. The exception to this is if hardware is changed in the cars, since this cannot be simulated.

Handling the tests in this way saves a lot of time and is better for the environment than driving test cars on the same stretches of road over and over again. Another reason for the concept of ReSim is the need to test the systems under real conditions, i.e. the systems need to be subjected to the real world to be properly verified. The systems could be tested in a simulated environment but there is a significant risk that a situation that could appear on the roads would be overlooked, hence the vehicles are sent out on long road trips to ensure that they function in real life circumstances.

The Electronic & Safety department at Delphi Automotive, where this project was carried out, develops components, systems and software for passive/active safety, security, comfort and infotainment. This requires a lot of test driving, yielding an abundance of data which there currently is no tool for interpretation of low level data in a comprehensible way, low level data refers to the data that is passed around in the vehicle's systems.

The clients that commission the ADAS quite often have verification and development teams themselves and require access to logs that concern them. It is common that they order a certain amount of mileage produced under specific circumstances, like rain or darkness, and wish to analyse the logs themselves.

Finding relevant data sets for verification of specific functions can be very hard due to the sheer amount of logs, e.g. driving on a highway, in darkness and heavy rain. This

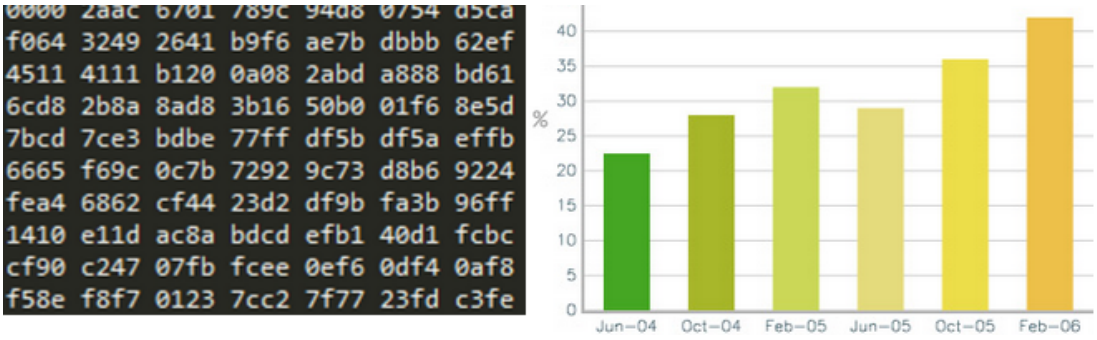


Figure 1.2: Left side: An example of what low level data looks like to humans. Right side: High level information - a diagram that is for humans a useful representation of the data in the vehicles, this could for example be the yield of correctly identified signs per month. The diagram shown here is an example and not based on actual data.

results in large amounts of man hours being spent on finding relevant logs for test suites of interest and is an immense waste of an engineer's time and a company's resources. Such waste can be prevented by simplifying the flow of information regarding the data gathered through the creation of a catalogue system.

To catalogue information regarding log files different characteristics of each individual log file's detections must be identified and presented in a comprehensible form. To accomplish this, data from the logs can be synthesised and converted into higher level information. Examples of information gathered in this way could include country, top speed, or weather conditions. Traversing this catalogue of information instead of the raw data can save both time and effort.

Low level data refers to the data that moves in the vehicle systems and is understood by computers. *High level information* is the result of fusing the data into a representation which is easily comprehended by humans. For example aggregating all the detections of traffic signs and how confident the system is of a correct classification over a long time period, and presenting these in a diagram. See Figure 1.2 for an example.

Storing this high level information regarding the logs in an efficient manner becomes more important as the amount of logs increase. For fast and easy access it could be facilitated in a database which has the added benefit of both sorting and fetching files when needed by the system or user. When this catalogue of information exists, one can start to implement software for automated checks.

An automated checking software could traverse the synthesised information; analysing and comparing detections and conjectures made by the vehicle's on-board systems. The comparison could be performed using either ground truthed data or previously analysed data, e.g. from a previous software version.

1.2 Problem Statement

The aim of this project is to design a tool and implement a prototype that can fuse synthesised low level data into high level information; specifically for enabling efficient evaluation of performance of a visually oriented TSR system and presenting it in a comprehensible way.

According to the project suggestion from Delphi Automotive, the following items should be investigated:

- The possibility to automate part of the testing process for the TSR, specifically yield a representation of correctly detected signs per log examined.
- The possibility to determine from existing data what type of road the recording is from, which country/area the recording is from, and what the meteorological conditions were like.
- The possibility to use a database for facilitating management of the synthesised high level data.
- The possible addition of further information types to the high level data.

With a design in place, a prototype should be implemented as far as possible as a proof of concept to demonstrate the features that have been deemed feasible to implement. For those features not implemented an evaluation and recommendation should be provided.

The challenges have been extracting low level data, synthesising it into a manageable structure, creating high level information by aggregating and integrating data from several subsystems. Hence transforming messages travelling in a vehicle system into a useful representation comprehensible to humans.

1.3 Scope of Work

Both the tool design and prototype will have the following technical limitations:

- The tool will use previously recorded data for verification, and is not intended for stand-alone use.
- Processing of data is not intended to be performed in real-time in the vehicle, but during post-analysis.

Furthermore, the prototype tool will only be tested using data recorded in Sweden and the systems handling extraction and analysis of data in the vehicle will be treated as black boxes (see Section 2.2).

1.4 Thesis Outline

This thesis report is structured as follows: In this chapter the problem is presented, together with a brief context, problem statement and scope of work. The following chapter explains the background, Chapter 2, the test system configuration, what was discerned from previous work and surrounding third party systems (such as Global Positioning System (GPS) and vision detection) as well as how the data was collected, how it is used and how it can be visualised. In the method chapter, Chapter 3, the work flow regarding the prototype tool and an evaluation of the available data is presented. In Chapter 4 there is the complete design of the prototype tool, how the data was extracted from the MATLAB files, in what way it was sorted into a database, and what statistical analysis was performed. The following chapter (Chapter 5) contains a discussion of the proposed design and a future work section; the latter containing the results of the investigations concerning additional features. The last chapter (Chapter 6) presents conclusions and provides recommendations.

2

Background

The work in the Electronic & Safety department at Delphi Automotive is mostly focused on Advanced Driver Assistance Systems (ADAS). These systems consist of several subsystems which are all required to work in harmony to aid the driver towards a safer driving experience.

To create a system that takes the environment into account in its behaviour, there are a lot of subsystems that need to be incorporated. Not all parts of the subsystems are developed in-house at Delphi Automotive, which means that some equipment used to interpret the surrounding environment of the vehicle is developed and manufactured by third parties. The particular system that this project has worked with is divided in two parts: a radar, which is a product of Delphi Automotive, and a vision detection system which is designed and manufactured by a third party.

To achieve a somewhat complete understanding of the system and learn how to interact with it, some background knowledge is required.

This chapter focuses on setting the project into a technical context, as what parts that make up the system. To understand the further work in this project some insight to each of the components is desired. The following sections explains what has been discerned from literature studies about the concept of Active Safety, the system set-up that this project has worked with, and the third party solutions that are employed at Delphi Automotive and how they contribute to the final system.

The first section of this chapter presents *active safety* as a concept, in Section 2.1, as this is the field this report takes its starting point. The following one presents the system set-up of the Active Safety Systems (ASS) in a test vehicle as it is used in the test cars used, Section 2.2. Two key components are the Global Positioning System (GPS) and the vision detection, as these are the basis of the work described further on in this report. Section 2.3 and 2.4 gives a short summary of GPS and vision detection respectively. Once the system has registered the surrounding environment the data need to be organised, an explanation of how the data used in this project was collected, both from real test

drives and simulated ones, is found in Section 2.5. When the data has been collected the flow in the projects at the Electronic & Safety department is relevant for later analysis, Section 2.6 gets acquainted with the current scripts and hardware in operation at Delphi Automotive. And finally, to find inspiration and guidance background studies in related fields is presented in Section 2.7.

2.1 Active Safety

Modern cars have several systems which contribute to the safety of those in and around them; some of which are passive, like seat belts, while others can take direct action to help the driver in handling the vehicle, such as Anti-lock Braking System (ABS). In this context *Active Safety* refers to systems that are capable of perceiving their surroundings and taking active actions to prevent unwanted effects, e.g. the ABS preventing loss of traction during braking. These ASS need a way to understand the state of the vehicle to help avoid an accident or in the worst case minimise the effects of a crash. There are many different types of highly sophisticated systems that belong to this group:

- Active Safety Systems (ASS) are the most common. Most vehicles on our roads today have at least one of these systems, e.g. ABS, traction control, or collision warning.
- Advanced Driver Assistance Systems (ADAS) refer to semi-automated vehicle functions. They often use visual processing and radar to interpret the surroundings and take evasive actions, e.g. Adaptive Cruise Control (ACC) and active steering. A set-up of such a front-mounted visual processing system is seen in Figure 2.1.
- Autonomous vehicles. Fully automated vehicle control, used for replacing human drivers, e.g. to form vehicle trains on highways.

An example of ADAS is the previously mentioned ACC subsystem which functions by measuring the distance to and velocity of the vehicle in front of the host and adapt the host's speed automatically to maintain a specified distance between the two vehicles, 'host' refers to the vehicle that the system is mounted in.

The ACC subsystem does not require any special hardware in other vehicles, it is completely self-sufficient in that way, but usually requires a radar, a camera and a computer to be present in the host vehicle. The camera and radar outputs are not exclusive to the ACC subsystem, they are used by several subsystems, referred to as Software Component (SWC)s, which all have different purposes [1].

The idea for the ACC is closely linked with that of the Automated Highway System (AHS) which was presented as early as 1940 at the New York World Fair. The idea behind the AHS was to build automatic motorways where vehicles would drive with little space between them, which would result in the benefit of increased capacity of existing roads. This would be possible through automated systems handling braking and acceleration faster and more accurately than humans.

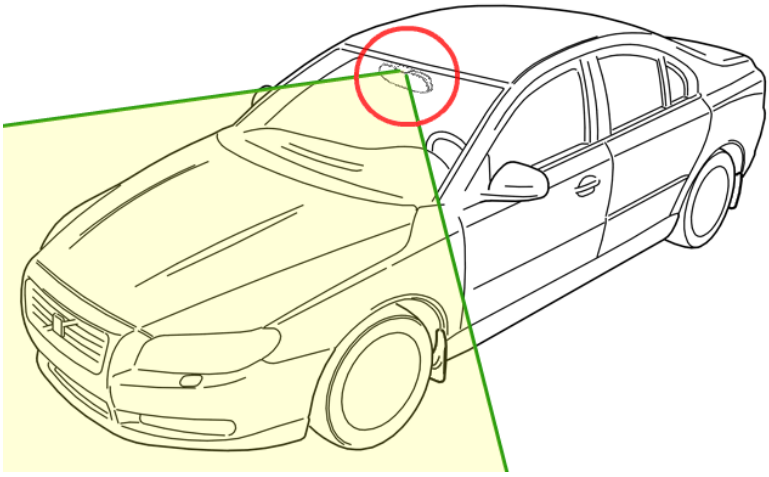


Figure 2.1: A test car. The circle shows the position of the camera: facing forward, mounted behind the rear view mirror. Picture adapted from Volvo's press material.

General Motors (GM) and Radio Corporation of America (RCA) started to develop the idea of AHS in the late 1950's and demonstrated (albeit on a test track) a functional system for control of vehicle steering and speed. The next big breakthrough came in the middle of the 1980's when the use of computers in vehicles, or more specifically microchips, became feasible. In the early 1990's, both Europe and the US started research programs on automatic highways [2].

2.2 System Set-Up

ADAS are dependent on perceiving their surroundings as previously stated, and in the system this project is based upon, perception is provided by a radar which detects solid objects and a video camera which "sees" what is in front of the vehicle, see Figure 2.1. Both of these systems are connected to the vehicle's Controller Area Network (CAN).

What the camera registers is in a sense what the systems "sees" and uses image recognition software to recognise and classify vehicles, pedestrians, traffic signs and other objects. It also gathers relevant data about these objects, such as heading angle or object type, which are obtained through analysis. Both the analysis and image recognition is run in real time on a specifically designed processor which is integrated with the camera. Both the camera and the processor used in the system is provided by a third party. How the system performs its visual detection is described in Section 2.4.

The radar detects objects in front of the host vehicle using radio waves to determine the object's direction, speed, distance to the host, among other things. Objects often yield several echoes depending on size,and each of these echoes is referred to as a 'tracklet'. The information from the radar is mostly utilised in SWCs that handle distances to buildings, traffic railings or other vehicles. The radar used in the

systems delivered by Delphi Automotive is designed and produced in-house to fit the specific needs of the company's products.

The system gathers the output from both camera and radar and proceeds by correlating objects detected by the camera with objects detected by the radar and subsequently combines the data points into one object. Once combined, relevant data can be computed for each object, e.g. heading, velocity, distance, projected path etc. This step is referred to as the *Fusion* stage and the process of combining these two trackings into one object will henceforth be referred to as fusion [1].

Combining these trackings together is difficult but necessary, since the camera alone cannot determine vehicle speed, heading etc, nor can the radar recognise different road signs or differentiate between cars and trucks. Each fused object should only be represented by one image object but can have several radar tracker objects connected to it since large objects may cause several echoes (or be hidden from view for a number of frames and then reappear). These problems are handled in the existing system implementation and are not something that needs to be considered during this project.

In a live system this fusion stream is created by a SWC which in turn acts as input for several other SWCs. These SWCs each handle a specific task, e.g. emergency breaking for immediate collisions or automatically turn the host vehicle if it is headed off lane into oncoming traffic.

The actual system that resides in the car is structured as seen in Figure 2.2 and consists of an external hard drive, an on-board testing computer and a screen mounted on the dashboard on the passenger side of the car, offering the test driver a possibility to monitor the system. A car carrying this type of test equipment is commonly referred to as a *test car*. When the systems are live they make decisions based on the detections made by the subsystems and approriate messages are sent out on the communication buses, thereby making the vehicle take action. An example of this can be applying the action of braking if the system detects a stationary object in front of the host vehicle while it is moving. But as the systems are not fully developed in the testing phase, they are not connected to the test vehicle's mechanical systems. Instead the test computer is connected to the communication bus and records all information that is transmitted on the bus and sorts it into a number of files stored on the hard drive, as described further in Section 2.5.

2.3 Global Positioning System

GPS-receivers are very common these days as most cell phones have one built in and they can be found in virtually every modern car with a navigation system installed. It all began when the first satellite (Sputnik 1) was launched and those studying it realised they could calculate the orbit using the Doppler Effect. If the position of the satellite was known, calculating the satellite positioning calculations backwards could determine the position of the people on the ground.

In the early 1960's independent parts of the US military started developing positioning systems and eventually, in 1973, a combined system was introduced to and

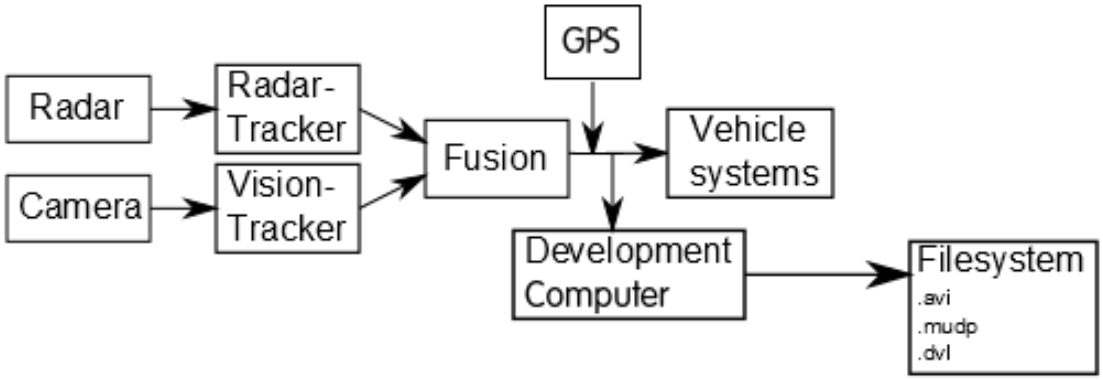


Figure 2.2: A block diagram showing an overview of the Advanced Driver Assistance Systems (ADAS) and the log files it produces. The boxes represent different parts of the system which in most cases are separate chips. The arrows represent communication buses.

approved by the US government which became known as *Navstar*. The first experimental satellite was launched in 1978 and 11 years later, in 1989, the first operational GPS satellite was launched. The system was completed in 1993 with the launch of the final satellite, totalling 24 satellites orbiting the earth [3].

The early systems utilised the Doppler Effect to calculate the latitude and longitude of the receiving device while modern day GPS uses a much more sophisticated method of determination. The GPS satellites continuously transmits a pseudorandom message which per default is 24 hours long, but can be shorter or longer depending on configuration. When the previous message ends, a new message immediately starts transmitting, and at the exact same time as the new message starts transmitting, all receivers starts reciting the same message internally at the exact same speed. When the receiver determines its position, it compares its own internal message recitation to how far the received message transmitted by a satellite has come in its recitation. Since both satellite and receiver recite the message at the same speed, the difference can be converted to a time difference, which is the propagation time of the signal. Multiplying the propagation time with the speed of radio waves in air yields the distance to the satellite.

The distance combined with the satellite's predetermined orbit, which is stored in all GPS devices from the factory, results in the device having a distance to a fixed point in space. For exact positioning a technique called trilateration is used, which means that the distance to each satellite yields a one axis determination, meaning three satellites are needed for a three dimensional determination. A fourth satellite is generally used to synchronise the internal clock of the receiver [4].

In this project, the source for the GPS coordinates (longitude and latitude) is the previously generated MATLAB files containing different types of data. The accuracy of these coordinates are around 15 meters according to the technical specification of the GPS receiver, but according to observations they generally are closer to 11 meters [5].

2.4 Vision Detection

The surroundings of a vehicle in traffic can often be chaotic with an abundance of fast moving objects. The recording systems cannot keep track of all the objects due to bandwidth problems, instead the recording equipment needs to filter what is of interest to the system and only save those data points. In the set-up of the system used in this project, a processor specifically designed for recognising and classifying objects of interest is mounted in the vehicle (together with the camera and radar). This section will explain the segmentation of images recorded by the video camera.

This system does not utilize any ground-breaking new methods. The idea of traffic sign recognition is so standard that MathWorks offers a basic MATLAB Traffic Sign Recognition bundle for free [6]. What is interesting is how it finds specific content in a complex image. There are several variants of doing this but usually they have three steps: segmentation of the image, detection of shapes, and visual classification [7–11].

In segmentation the image is divided into so called *blobs* using extrapolation of colours. The result of this is a coarse picture from which various shapes can be deduced through feature extraction. This means clustering parts of the image with similar colours into a Region of Interest (ROI) using pixel groups that are then analysed for shape. This is in many ways the most important step, since if there is a problem the following steps will have a incorrect picture to base their computations on.

Detection of relationships between ROIs are analysed using colour, which is error prone since colours are affected by a multitude of factors such as time of day or weather. There are techniques that uses a database of signs to train the software to recognise shapes and colours that enables it to recognise all signs know to said database [9]. In this particular system only round, rectangular and triangle shaped signs are regarded as interesting as the detection step is computationally expensive and there is limited memory on the chip holding the processor [7, 10].

The visual classification is done through comparing the interpretation of the recorded image with known relationships between different ROIs that occurs on the traffic signs that are of interest for the system. Furthermore the classification needs to be robust for things like occlusion and inconsistencies. This is handled through generating synthetic images with a variety of differences that the system could encounter, and yields an acceptable result of detected traffic signs [8]. The result of the different steps can be seen in Figure 2.3. In the system used at Delphi Automotive the image is never treated in colour but is always monochrome, this is due to improved performance without the extrapolation of more than one colour [7, 11].

The environment that traffic signs are located in is cluttered by nature, with many signs appearing in short distances, obstacles, moving cars etc. To make things worse, they are also heavily affected by sight limiting factors such as bad weather, bad lighting, wear etc. Because of these factors the system does not always produce a guaranteed outcome, in this case the classification and interpretation of something the system found in a recorded image. To handle this, the traffic sign recognition software gives a value representing the chance that the classification is correct, i.e. a probability [7, 11].

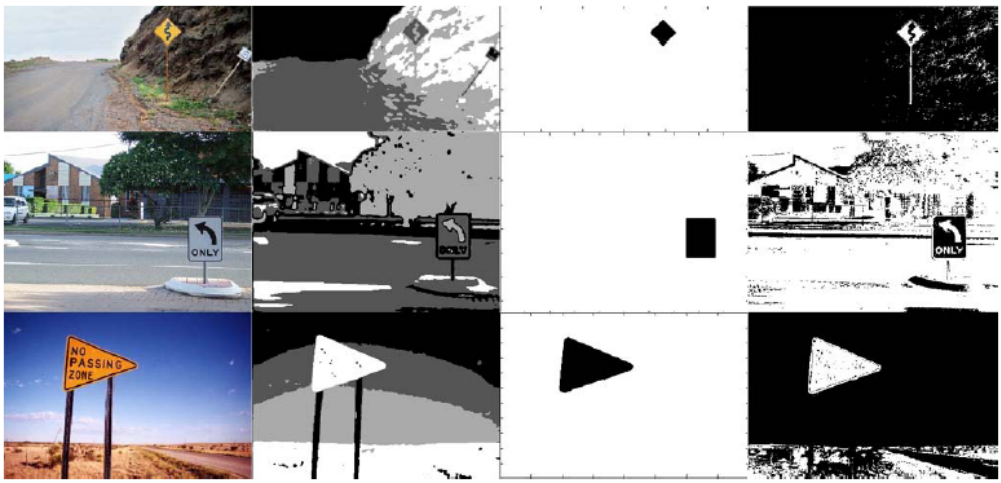


Figure 2.3: Segmentation result. First column: Original images. Second column: Segmented images. Third column: Candidate ROIs. Fourth column: Segmentation results of the three step method. Picture from [10]

Exactly how this value is calculated is not possible to conclude for the purpose of this project as the system used is covered by a non-disclosure agreement between Delphi Automotive and the third party, for the effects of this see Section 3.3.6.

2.5 Data Collection

During test drives all detections made by the system and all conjectures drawn from those detections are logged on the hard drive that is a part of the test equipment. These test drives are not performed in a controlled environment; the system is tested in real environments as the system is expected to manage a wide variety of situations. The on-board computer stores the data gathered from the system into the following files:

- .avi - A monochrome video file recorded by the camera.
- .mudp - Internal state logging of the system; records of the communications between the internal SWCs.
- .dvl - Communication log of what the system would send to the mechanical systems.

Due to the function of the systems developed by Delphi Automotive, the output is segmented into several pieces each about one to two minutes to keep the file size reasonable, which still is several gigabytes per segment. A simplified diagram of the system is shown in Figure 2.4.

A segment is referred to as a *log file* (despite actually consisting of the above three files). A full test drive generates a *log set*, which is 1-3 terabytes and 300-450 log files. The size varies depending on the length of the test drive.

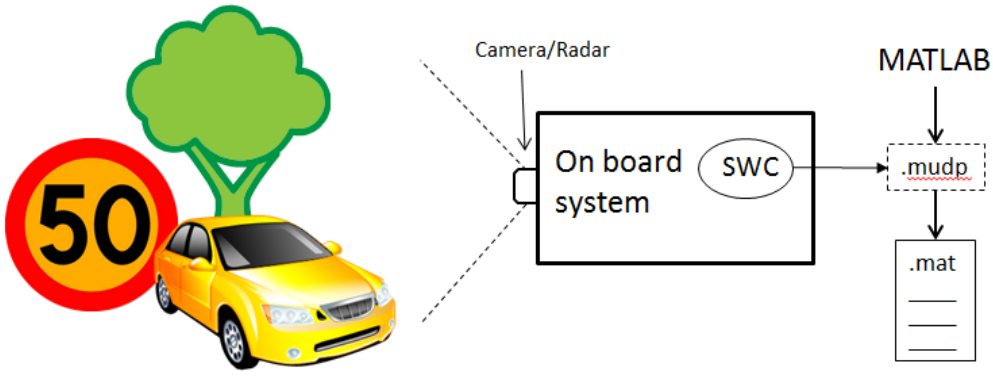


Figure 2.4: A block diagram showing an overview of the structure of the system: a Software Component (SWC) producing the .mudp file, where the MATLAB scripts are applied, and the result - a .mat file containing the structures with the metadata about the system.

2.5.1 Resimulation

When an update is applied to an SWC it is quite a waste of time and resources to repeat entire test drive suites. A test suite is at least 100 000 km, so repeating all test drives for each released software update is not feasible. Not only would it contribute to unnecessary pollution but the circumstances during the drive will most likely not be the same.

The exact combination of events which caused an erroneous response from the system is often complex and a nearly impossible to recreate as the test drives are not carried out in a controlled environment. Therefore a simulated test drive, referred to as doing a Resimulation (ReSim), is faster, cheaper, more environment friendly and all in all more sensible.

During a ReSim all detections and conjectures based on them are recalculated using the stored raw data from the test drive, i.e. the video feed and radar detections. With the raw data as input to the updated SWC, new results will be generated which are analysed to determine if problems have been corrected or if they still persist. The resulting output must also be analysed to verify both the absence of new errors and continued functionality of the SWC.

If the hardware in the vehicle needs to be changed, SWCs developed for the new hardware cannot use data collected with older hardware, rendering old data useless. The opposite is also true in practice as data could be translated between hardware generations, though this is not commonly done as the process is prone to error.

Running a ReSim usually takes two to three hours on the local cluster for an entire log set from a regular test drive, and can easily take up to eight or ten hours on a powerful laptop or stationary computer.

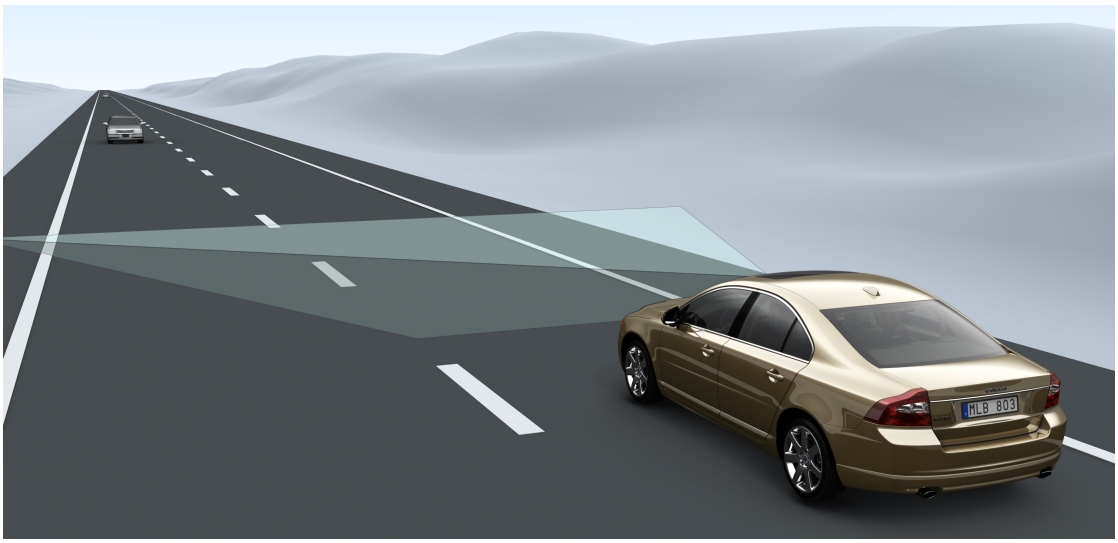


Figure 2.5: Field of view of the vision system, represented by the cone in front of the car. Picture courtesy of Delphi Automotive.

2.5.2 Data Visualisation

When a fault is found, either in the original logs or later during a ReSim, the developers need to know what type of scenario caused it. Therefore a tool for visualising recorded data has been constructed in-house at Delphi Automotive called Data Visulisation Tool (DVtool). It uses the .dvl files, previously described, as input and reads all files that is connected to that particular log file. The tool shows the video recorded by the test car's equipment with a selection of streams to choose from, such as just the radar tracking or the fusion mode. The video is about one to two minutes long for storage reasons, as previously discussed.

The classification part of the Traffic Sign Recognition (TSR) system relies solely on the vision stream originating from the forward facing camera, and is thus the one of interest to this project. The view that the system covers is seen in Figure 2.5.

Using DVtool in TSR mode gives access to some useful information: Coordinates of the sign in that specific frame of the vision stream, approximate distance to the sign from the car, a confidence value with each detection (denoting how correct the system consider the classification) etc. When playing the .avi file it also provides a targeting rectangle around the signs with an attached ID-number, enabling easy correlation between image object and matrix data. An example of how the visualisation in DVtool could look like is seen in Figure 2.6.

Figure 2.6: The view from the vision system. A screenshot from the DVtool showing detection of two speed limit signs and a car. Picture courtesy of Delphi Automotive.

2.6 System Research

The entire system was studied to be able to adequately traverse the documentation and code surrounding the test equipment to gain an understanding of what possibilities were open to this project.

The data flow in the testing system is as follows: The system interprets its environment during test drives, logging all detections and conjectures. When the test car has completed its test drive, the hard drive containing the log files are taken out of the vehicle for upload to the cluster storage at the Electronic & Safety department. MATLAB scripts are then used for extracting data from the .mudp and .dvl files coming from the test drives, creating .mat files containing the collated data.

These scripts were thoroughly studied, both to get familiar with the MATLAB code but more importantly to understand how the data extraction is performed and thus gain knowledge of the underlying structures. It is not possible (due to proprietary rights concerning the TSR system) for this report to recount in detail how the extraction of data from the log files is performed. What can be said is that the MATLAB scripts extract and translate the information from the internal logging to sizeable MATLAB structures containing everything the system recorded.

Running these MATLAB scripts on a log set typically takes around twenty minutes on the local computational cluster at the Electronics & Safety department. The size of such a set is around three terabytes which corresponds to around eight hours of test driving, a fixed megabyte per minute ratio is not applicable since the size rather depends on the number of objects encountered and the system communication.

These files are always generated in post-analysis for ease of access for other projects and enabling access of data without having to access the .mudp and .dvl files directly as it would take much time for ordinary desktop computers to produce, especially when weighing in that the data would need to be fetched remotely.

In the case of needing a ReSim (see Section 2.5.1) due to existing files being based

on an old software release, the detections from the original logs are given as input to the new software and the new output is recorded and stored. Once complete, the MATLAB scripts are run on the new results creating the same files as for regular test drives.

2.7 Literature Studies

In preparation for this project, various sources for journal articles, theses etc was searched through for relevant texts in the area of TSR. Most articles found were in the area of gathering data and image recognition functionality.

Three papers were read regarding traffic sign recognition systems in cars for understanding of the TSR system and the way it is designed. The systems described by the different papers are very similar to each other and only present slight differences to the set up of the TSR system used by Delphi Automotive. This shows that such a system requires a certain type of parts to be able to function properly and neither paper reveals any groundbreaking new technology [12–14].

In his thesis [15], Miguel Jorge Pereira Cova describes his construction of a system which shares many similarities with the system used for this project. It describes the development of the vision detection subsystem from scratch and in the final part of the paper adds a global localisation function. The calculation of the position of individual signs is based on the global position of the car and the distance from the car to the sign, much as discussed in Section 3.3.4.

What is lacking in that particular paper is a measurement or statement regarding the accuracy of the GPS coordinates obtained from the equipment used. Therefore, it is unknown if the localisation concept used in the paper would be able to reproduce somewhat similar output in a scenario where a vehicle passes the same sign several times, as is the case with the test driving.

A different approach is presented by Wai Yeung Yan, Ahmed Shaker, and Said Easa in in their paper [16], where instead of driving around with a camera, the signs and their locations are data mined from Google Street View. This sparked the idea of using the test vehicles' GPS coordinates in combination with Google Street View and comparing the results, verifying that the TSR system has made a correct classification against Google Street View.

Using such a system has potential, but requires more automatisation before becoming viable as there is no point in classifying traffic signs in another program when ground truthing of collected data directly works just as well. Additionally, it is unknown how often Google Street View updates their data and gathers new images of traffic signs. It was decided that such a system was outside the scope of this project, but it is kept in mind at Delphi Automotive for future projects.

Two papers [17, 18] were found which are both examples of projects where the gathering of data is not the goal, but rather a means to an end in the construction of a system, and was thus an inspiration for what could be achieved with the type of data collected by Delphi Automotive. These papers can be seen as the spark for the idea for this project's attempt at using Google Maps and OpenStreetMaps as a source for

information regarding roads, as discussed in Section 5.1.1.

Since the visual detection system comes from a third party it was challenging to find papers on how this particular image recognition works. They do supply related research and there are other articles written on TSR in relation to the visual detection system used in this project as well. In their article [11], the authors offer a good explanation of the internal computation of image recognition that is used for detection of vehicles. The method of parsing an image was described in greater detail in Section 2.4.

There are several articles [7–10] on alternative approaches of image recognition in TSR, but these do in general cover the algorithms parsing the image, though they definitely demonstrate the potential that exists in this field. For example the limitation that exist today that the system detect a limited number of signs that are deemed most relevant, in the future these types of systems may very well recognise any type of signs which comes in handy when the TSR is expected to have multi-regional support.

3

Method

To be able to tackle the design of the tool, a significant amount time was spent on background research, documentation studies, getting comfortable with the systems and tools, and constructing experimental set-ups. All these steps had to be taken before an assessment of the available data could be made which in turn provided a sufficient base for the final prototype design.

This chapter describes the different approaches taken to solving the problem of converting low level data into high level information, and what was learned in the process. The *low level data* is the raw data that is produced by the vehicle's on-board systems, the *high level information* refers to the data after it has been processed and synthesised for further analysis, i.e. made hundreds of terabytes of low level data understandable for humans.

The initial approach for a prototype tool was to write in MATLAB, the idea came from the existing MATLAB scripts for extracting data from the vehicle. This solution was found wanting, and the process is covered in Section 3.1. After some discussions and research, it was decided that Python was the way forward. It is a versatile programming language with easily imported libraries for MATLAB structures and database connections, the revised Python approach had better performance is described in detail in Section 3.2.

A functioning prototype allowed the low level data to be extracted and assessed, the data comes from several subsystems and each type is described in Section 3.3. Synthesising and correlating the data points to each other is the prerequisite to produce high level information as the different data points are fused into a consistent representation.

The final section describes why a database was needed, what type of database language was used and how the database connections were set up (Section 3.4).

3.1 Initial Prototype Using MATLAB

An initial attempt at constructing a prototype in MATLAB was made by using existing MATLAB scripts as a template, reading raw data from the .mudp and .dvl files directly into MATLAB. During these first trials output data was stored as a local file for convenience, focusing on the functionality of the tool rather than a potential database connection.

The prototype proved to be quite slow as for a single log file portion, around two gigabytes in size, it took at least five minutes to extract the Traffic Sign Recognition (TSR) data alone, meaning this solution required over sixteen hours to extract all data from a single log set. Comparing this solution to the MATLAB file generation described in Section 2.6 where an entire log set was completed in around 20 minutes, it was discerned that there was an enormous performance gap leading to the decision of scrapping the approach of raw data extraction.

Instead the .mat files, produced by the MATLAB scripts previously described, were to be utilised as they hold the data already extracted from the raw data files. They were also in many cases already generated as they are used in several projects at Delphi Automotive, and not exclusively for this project. It was decided that these .mat files were better suited as a source of data and therefore all subsequent imports of data use these .mat files instead.

By switching to the previously generated .mat files the resulting execution time was cut to around 15 minutes which compared to the previous version is a 4-8 times faster solution. These figures were measured when the scripts were run on an ordinary laptop with the log files stored locally on an external hard drive. The scripts have not been written with performance as a priority, but they still give a hint of what can be achieved in terms of time saved with the different approaches.

3.2 Revised Prototype Using Python

The later MATLAB script was significantly faster than the earlier, but was not deemed ideal for database connections, nor was it particularly fast in its data management. There is a Database Tool for MATLAB which enables data to be exported into a database and is available free of charge for students, but would require additional licences for Delphi Automotive, which was not feasible considering the limited resources granted to this project.

The project's supervisor at Delphi Automotive suggested an excursion into Python to investigate potential possibilities and benefits, especially regarding database connections but also to compare performance between MATLAB and Python since the latter is based on C and therefore should be able to provide a faster data management solution.

3.2.1 Reading MATLAB Files into Python

The problem, and what came to be the most time consuming part of the Python implementation, was to import MATLAB data into Python and storing it in a suitable

format. This was in essence a problem of typed structures.

MATLAB switched to a completely new type of structure in its .mat files with the release of version 7.3, also known as R2006b (the number in the middle denoting year of release). The previous file format was used between Version 7.0 and 7.3, and consisted of compressed Unicode characters, i.e. text files. The new structure is more advanced, being a Hierarchical Data Format 5 (HDF5) file capable of handling variables larger than 2GB.

It was initially thought that the existing MATLAB files were saved in HDF5 type since they were created using the later version R2011a. This was not the case as HDF5 files only supports numerical values and the files at Delphi Automotive also contain character strings. When MATLAB tries to save incompatible data to HDF5, it automatically reverts to its old Unicode structure [19].

These different formats naturally require different procedures for importing the data into Python, but there exists free Python packages for both types. For the HDF5 files, a package named h5py is required while the files of the older format require two packages called SciPy and NumPy.

SciPy is a open-source package for Python with functions that helps developers in the areas of mathematics, science and engineering. SciPy depends on the package NumPy which is the fundamental package for scientific computing with Python, especially its N-dimensional array object and ability to integrate Fortran code. The latter becomes important as the MATLAB engine is written in Fortran, and it is preferable to be able to utilise MATLAB functions without the need to start MATLAB as an external program [20].

Once it was discovered the MATLAB files were saved in the old format, it still required quite a bit of work before importing these files into Python yielded any usable data as they consist of nestled MATLAB structures.

SciPy's read function per default does not attempt to recursively traverse a MATLAB structure when asked to import a MATLAB file, it exclusively reads the highest layer. The solution is to check each variable that SciPy reads, and if the object is in fact a MATLAB structure, apply SciPy's read function again. This is repeated until all variables have been read. The solution is further described in code in Listing 3.1.

A constant factor in the extraction of data from the MATLAB files has been that the data the program tries to extract may not exist, and if it does it may be corrupt. This has led to special care being taken around statements extracting data from these files. The reason for data missing is in most cases due to the entire system in the test vehicle entering a bad state and not being able to recover properly, or subsystems not being able to meet their deadlines thus also entering a bad state.

3.2.2 Performance

When running the final Python script on the same laptop as was used for testing the MATLAB scripts, it completes its task in around 10 minutes. That is for en entire three terabyte log set and it should be noted that the Python solution fetches data remotely, and instead of storing the result in a local file, stores it in a database, albeit on localhost.

The only optimisation that has been implemented is based on how the MATLAB files are structured. MATLAB stores the data collected in a tree structure, where the leafs are matrices. The script produced reads the confidence value matrix to determine when detections were made, as all detections are assigned a confidence value.

The confidence value matrix for each log stores 0 in all locations where there are no detections, but all detections are grouped and placed on the first positions of each row. This means that as soon as a 0 is encountered, there will not be any more detections on that row and therefore the tool can safely stop traversing it, instead moving on to the next one and therefore not asserting that there were no more detections.

```
def loadmat(filename):
    # Replacement for SciPy's loadmat that reads more than one level of
    # nestled MATLAB structures and converts them to Python dictionaries

    data = scio.loadmat(filename, struct_as_record=False, squeeze_me=True)
    return _check_keys(data)

def _check_keys(dict):
    # Checks if entries in dictionary are mat-objects.
    # If so, todict is called to convert them into nestled dictionaries

    for key in dict:
        if isinstance(dict[key], scio.matlab.mio5_params.mat_struct):
            dict[key] = _todict(dict[key])
    return dict

def _todict(matobj):
    # Recursively creates nested dictionaries from MATLAB-objects

    dict = {}
    for name in matobj._fieldnames:
        elem = matobj.__dict__[name]
        if isinstance(elem, scio.matlab.mio5_params.mat_struct):
            dict[name] = _todict(elem)
        else:
            dict[name] = elem
    return dict
```

Listing 3.1: Proposed solution in Python using SciPy for reading MATLAB structures

3.3 Evaluation of Available Data

The types of information in the MATLAB structure is of a wide variety since all information moving through the system is logged and every part of the system gets its own branch in the resulting tree structure. Among this information, data that is deemed relevant is examined to evaluate if the conjectures made by the system are

correct. Data is not only collected data from the TSR part of the MATLAB-files but also from the Global Positioning System (GPS) and Controller Area Network (CAN). The data points from the GPS helps by providing longitudinal and latitudinal coordinates for each sign, while the CAN data provides a possibility to synchronise the synthesised data. In this section the data used for this project will be presented and explained.

3.3.1 Data from MATLAB files

Both during the test drives and the Resimulation (ReSim)s the system logs masses of information moving between the Software Component (SWC)s. For the purpose of this project, only some of the data collected is of interest, hence only the relevant parts concerning the vision, CAN time and TSR systems are extracted.

When running the previously written MATLAB scripts on the log files, .mat files are produced that contains large matrix structures with *all* information that was internally computed and what messages were intended to be sent out to the vehicle's mechanical systems. In this project the .mudp-file of interest contains data from the internal logging. These .mat files are used in other projects at Delphi Automotive and have typically been previously generated for all logs, they are not generated solely for the sake of this project.

The .mat file that hosts the data is constructed in a nestled structure that can be compared to a file system. To find the relevant data points one need to know in which part of the structure to look. The search path to a matrix logging all occurrences of traffic signs looks something like *mudp/information/logging/traffic_signs*. That is, the relevant data points from several parts of the entire car's systems that are connected to the on-board computer, are collected and organised for evaluation. The information that is extracted from the .mat files is:

- GPS coordinates of the vehicle
- Sign type and information (e.g. speed restriction: 70)
- Position of sign relative to vehicle for each frame
- Frame indexes of the occurrence of a sign. The frame index is an internal state variable identifying which video frame of the log file's video stream the object occurs in.
- Confidence value for the detection (see Section 3.3.6)
- Sign ID, a number identifying the sign in the video file

To save every bit of information concerning every sign that the systems detect is not feasible with respect to space and level of interest. For information that is static (such as type) during all frames the systems records the sign, there is no need to pick a specific frame to save the value from. However, the GPS coordinate only updates once every second between the discovery of the sign and the final frame in which it can be

seen, which roughly translates to one GPS update for 20 frame updates. In this case, it is most reasonable to pick the last frame the sign occurs in as this is the closest to the vehicle and thus closest to the position of the sign. More on this topic can be read in Section 3.3.4

How many frames the sign appears in is also relevant as this gives a clue to whether the system was mistaken, e.g. if a sign is detected during only two frames then probably it was something in the environment that was misinterpreted and the detection is treated as suspicious or erroneous.

3.3.2 CAN data

From the CAN, the 'CAN time' is extracted which is the only common reference point between the subsystems as different they run on different schedules. The CAN time represents milliseconds passed since the system was turned on, i.e. the driver turned the ignition key. All subsystems have CAN time but it may unfortunately be represented in different formats.

3.3.3 GPS Coordinates

The GPS module only receives updates once every second and does not calculate any coordinate estimations by itself. The GPS coordinates are stored in the MATLAB files as decimal degrees and only have a four decimal precision; meaning the accuracy of the position of the vehicle is about 11 meters.

Closer inspection showed that this limitation was not introduced with the creation of the MATLAB files but rather a limitation in the GPS receiver itself as it does not receive more precise coordinates. The technical specification states that the developer only guarantees an accuracy of within 15 meters in standard GPS mode, but the module should be able to further improve its positioning by using other GPS modes [5].

The frame index and the update rate of the GPS is not automatically correlated as they are supplied by different subsystems in the vehicle, so to approximately correlate the two to get the coordinate of a registered sign the CAN time was utilized. While the GPS system is based around seconds and saves its CAN time in this manner, the rest of the vehicles subsystems use CAN time in microseconds, so to correlate with the different formats the CAN time from the CAN branch was divided by a million for format compensation. This yields an as close estimate as possible, a circle roughly 11 meter in diameter surrounding the vehicle, acting as GPS coordinate of discovered signs.

3.3.4 GPS Coordinates of Traffic Signs

The original plan regarding the GPS coordinates for each individual traffic sign was to compute them as accurately as possible using the GPS coordinate from the vehicle's receiver and the TSR system's spatial estimations for the distance and angle to the sign from the vehicle, as shown in Figure 3.1. With these values it was deemed possible to calculate the sign's coordinates by using the Harvesine formula which is intended for

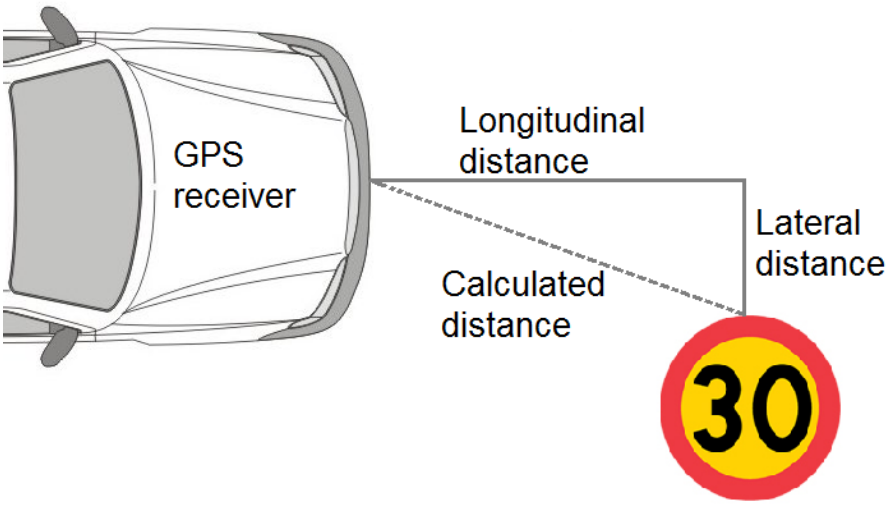


Figure 3.1: Longitudinal and lateral distances of the sign from the host vehicle are given by the TSR system. The calculated distance can be used for calculation of GPS coordinates of the detected traffic sign. Picture adapted from Volvo's press material.

calculating distances between two points on a sphere. By having one GPS coordinate, the distance to the sign and the ability to determine the heading based on the host vehicle's previous GPS coordinates, the second coordinate can be calculated [21].

Upon further inspection it was deemed that the curvature of the Earth can safely be disregarded as it is negligible when the distance between objects is in the range of 20 meters or less since the road is more likely to be a source on incorrect calculations.

Since the GPS only updates once per second, the selection of a starting point for the computation would have to be based on when the last GPS coordinate is received before the traffic sign dissappears from view. Preferably the point of origin for the computation would be as close to the sign as possible to minimise the impact of potential errors in the systems estimation.

At this point the idea of calculating GPS coordinates of signs was abandoned since the starting point was deemed to be too imprecise, due to the stored GPS coordinates only having four decimal points. Instead a more straightforward technique was implemented by a lookup of the GPS coordinates for the last frame the sign appears in the camera's visual feed. This coordinate is recorded in another part of the system but was extracted in much the same way as other values (see Section 3.3.1).

3.3.5 TSR Detections

The system detects several items in the environment around and in relation to the host vehicle, cars, trucks, lanes, etc. As this project is focused on the TSR, the information that has been closest scrutinised is what is registered when a sign is detected.

Among the information regarding a sign that is saved by the system is an internal id

(used for identification in Data Visulisation Tool (DVtool)), a sign type and a value if applicable (e.g. a speed limit of 70, type id 1), a confidence value (see Section 3.3.6), the position relative to vehicle, the frame index for the video file, the position in the frame, a variable denoting whether the sign is electronic, another variable denoting whether it is relevant to the vehicle's systems, and a number of internal variables that is not of interest to this project.

Ideally all of these variables have a value associated with them, but it happens that the classification algorithm makes mistakes. This makes for classifications of plain errors, which the tool produced by this project should be able to identify.

The TSR information discovered for a log is put in a dedicated matrix in the structure from the .mat file generated, even those that are erroneous. Some faults the system can discover itself (such as all values missing), but other combination of values are more difficult to determine if they are erroneous or not. After documentation studies it was decided that since the confidence value is the certainty the system made a correct detection and classification, it is used for primary classification of correctness in this project.

3.3.6 Confidence Value in TSR

As previously discussed the TSR system has its own functions for classifying traffic signs from visual input. The information it offers for each individual sign contain a value called 'confidence value', denoting how certain the TSR system is about its conjecture regarding that particular sign, acting as a type of built-in validation.

The confidence assumes a value between 0 and 1 with up to fifteen decimal points for important signs, e.g. speed limits, while only varying between 0 and 0.5 for what the system classify as less important signs, e.g. town signs. Generally, as the host vehicle gets closer to a sign the confidence increases and will only be set to 1 in the last frame a sign occurs in provided the system considers its classification correct.

This variable is computed in the processor constructed by a third party, and there has been no way to examine any documentation concerning how the system asserts its confidence. This is a frustrating situation as every evaluation and conjecture made during this project in connection with the TSR system in question is more or less dependent on this variable. The confidence value has been held as correct during the work of this project, but nothing can be said about its complete meaning.

During statistical evaluation the traffic signs not reaching a confidence of 0.95 will be marked as suspicious if considered an important sign. For the less important traffic signs, it is required to reach a value of 0.5 (which is also its maximum) to not be marked as suspicious. These figures are based on how other projects at Delphi Automotive use them, and what assertions they made. Unfortunately it was not possible to get a more accurate motivation, as again it was not possible to access the documentation.

3.3.7 External Data

In the project proposal there were three desired features stated that required additional data. These features were the road type the car was travelling on during the test suite, which country the car was in and what the meteorological conditions were. Base data for these features are not currently recorded by the testing equipment and will require incorporation of external data, retrieved from external sources. All features can be implemented using external calls to various online databases and are deemed feasible.

None of the above features were implemented as there were troubles finding sources that allowed enough calls per day to be able to fetch adequate data points for the previous logs (which are several petabytes). The background, limitations and what conclusions were drawn are covered in Section 5.1, discussing future work.

3.4 Data Handling

To facilitate handling of the data gathered, it was proposed to store it in a database since analysis can be done much faster and more convenient when stored in this fashion. Furthermore it would increase the logs searchability if one could traverse the information about the logs, rather than the logs themselves.

For this project the SQL dialect PostgreSQL was chosen as it is used for the main database at the Electronics & Safety department, and the goal is to incorporate the structure of the test database developed for this project into the main database.

To be able to insert the data into a database there is a need for a connector and in Python there are libraries to handle such connections. For this project SQLAlchemy was chosen which is an open source SQL toolkit that performs Object-Relational Mapping (ORM) which is a way for converting data between incompatible types in an object oriented environment. SQLAlchemy provides the benefit of making the communications with the database SQL dialect independent, i.e. the Python script can communicate with any type of SQL based database with SQLAlchemy acting as an interpreter [22, 23].

The queries are constructed as regular Python functions instead of detailed SQL statements. To insert data is a bit tedious, but a slightly more complex insertion is traded for easily constructed queries. An example of insertion into an existing table is seen in Listing 3.2, and an example of querying can be seen in Listing 3.3. Both listings are example code that is not included in the project.

```
-- Insertion of a 'user' into table 'user' with regular SQL statements
INSERT INTO users (name, fullname, password) VALUES (?, ?, ?)
('ed', 'Ed Jones', 'edspassword')


-- Insertion of a 'user' into table 'user' with SQLAlchemy
from sqlalchemy import *
from sqlalchemy.orm import sessionmaker

engine = create_engine('sqlite:///:memory:')
Session = sessionmaker(bind=engine)
Session.configure(bind=engine)
session = Session()

ed_user = User(name='ed', fullname='Ed Jones', password='edspassword')
session.add(ed_user)
session.commit()
```

Listing 3.2: Example of differences in insertion of data between regular SQL (top) and SQLAlchemy (bottom).

```
-- Querying the database 'users' with regular SQL statements
SELECT users.id AS users_id,
       users.name AS users_name,
       users.fullname AS users_fullname,
       users.password AS users_password
FROM users WHERE users.name = ? LIMIT ? OFFSET ? ('ed', 1, 0)


-- Querying the database 'users' with SQLAlchemy
our_user = session.query(User).filter_by(name='ed').first()
```

Listing 3.3: Example of differences in querying of data between regular SQL (top) and SQLAlchemy (bottom).

4

Prototype

The goal of this project was to design a tool that can extract high level information from the log files generated from test drives and Resimulation (ReSim) at Delphi Automotive. This chapter will present the final prototype that was developed with the functionality that was deemed feasible, based upon the evaluation of data previously made which is described in Section 3.3. Some decisions were made to facilitate the process and are discussed in Section 4.1.

The tool is built around a Python script, utilising the current analysis of the logs that is already in place at Delphi Automotive. The logs are subjected to a number of MATLAB scripts after they arrive back from the test drives (or during ReSim) which extracts the data and structures it. This is described in detail in Section 4.2. The tool proceeds by automatically collecting all low level data contained in the MATLAB files generated from log files produced either by a test drive or a ReSim. The low level data is then collated based on traffic signs, synthesised into high level information and inserted into a PostgreSQL database. The specifics around the database connections are covered in Section 4.3.

Once in the database the data can be analysed in whatever way the user wishes; e.g. finding logs containing faulty detections, how many signs was detected in a certain log, etc. In this project it has been used to show an example yield of correctly detected signs in a limited time period in Section 4.4.

4.1 Practical Decisions

The three points that need to be addressed in this tool is extracting data from MATLAB-files, synthesising the low level data into high level information and finally inserting that information into a database, i.e. moving the information from MATLAB-generated structures to a database table. The reason for not keeping the information in the MATLAB structures is a compromise between usability and

performance with the benefit of having the ability to perform analytic tasks on the resulting data set in the database while still being able to use the existing MATLAB files and the existing routines and techniques for generating them.

The decisions made in relation to different parts of the system are as follows.

Analysis: MATLAB To handle the structures that MATLAB produce, MATLAB itself would be the most suitable choice as it is easier to find the data points of interest using MATLAB. However MATLAB is not ideal when it comes to managing database connections nor is it very efficient in handling its own structures once they become significantly large. Hence it was decided against using MATLAB for anything other than using existing scripts for extracting data from the .dvl and .mudp files after ReSim.

Database: PostgreSQL All other projects at Delphi Automotive either already uses PostgreSQL, or plans to. It is an advanced open source object-relational database system, with focus on standards-compliance which promotes reliability, data integrity, and correctness. To be able to incorporate the test database into the current databases at Electronics & Safety department, it was built using PostgreSQL.

Connector: Python To facilitate handling of data of interest that is extracted from the .mat files, it was decided to use Python. There are alternatives to Python, but as it was meant to function as a connector between .mat files and a database it was decided that Python was sufficient for the task. Python offers good performance paired with a variety of open source libraries.

Python library: SciPy SciPy is an open source library for Python used for technical computing and offers functions similar to MATLAB's. More importantly, it was chosen as it offers methods for importing data as well as entire data structures from .mat files [20]. It is described in further detail in Section 3.2.1.

Python library: SQLAlchemy To handle the database connection Python was chosen as it is free and efficient, albeit with increased effort. To be able to handle the connections to the database SQLAlchemy was incorporated into the project. SQLAlchemy removes the need for dialect specific SQL statements, enabling the Python script to communicate with any type of SQL database [22].

4.2 Programmatic Structure

The prototype tool developed is divided into four parts, each designed to handle a specific area. The tool is designed to be executed from the command line and handles two types of input; a .txt-file with paths to individual log files (.mudp, .dvl or .mat), or the path to a directory containing a full set of log files.

Main method: Instantiates the tool. Handles checking of the input to the tool, if the input is valid it takes appropriate actions once it determines what type of input was received.

Sign class: A class implemented to store all information regarding each individual traffic sign detected as separate objects. Keeping it as a class-object makes identification of specific signs across several log files more efficient.

Database handling: Sets up the connection to the database, creating it if for some reason it would not exist. Initialises a session which communicates with the database for the rest of the Python functions.

MATLAB handling: Reads MATLAB-files and extracts relevant data as previously described in Section 3.3.1. Also catches and handles exceptions when data is missing or corrupt.

The tool has been structured in this way to facilitate overview of each part, help during testing as each part can be tested separately and with good programming practice in mind. Please see Figure 4.1 for an overview of the structure.

4.2.1 Methodology

The prototype tool is designed to handle two types of input: either a system path to a folder containing .dvl, .mudp and/or .mat files, or a text file with system paths to specific files of aforementioned types. Before proceeding with any calculations or data retrieval the tool verifies the validity of the input by verifying that the paths lead to a directory or folder.

Once verified, the tool proceeds by using a Python method called ”glob” which collects all file names of a specified type from a given location. In the case of a folder being supplied, the ”glob” method searches the specified path for files of interest, namely those ending with .mat (MATLAB files) with the help of the ”*.mat” call. If no such files exist, MATLAB is called on the entire folder applying the scripts discussed in Section 2.6 to create them, as they are also needed for other projects. If no .mat files are to be found after MATLAB has finished running the scripts, the folder is disregarded and an error is generated for the user.

If a list of files is given instead of a directory, and their existence have been verified, they are collected in much the same way but with the specific file names instead of ”*.mat”.

Upon completion, a structure is returned containing all file paths to the .mat files of the specified logs in a list facilitating traversal. For each .mat file the MATLAB-handler is called, reading the data into Python as a dictionary as previously described in 3.2.1

With the data stored in a dictionary, the tool starts determining what data is available, corrupt or altogether missing. The most important part of the data structure is the vision branch since it contains all information related to the Traffic Sign Recognition (TSR) system. If this branch is missing, the log file is useless since no

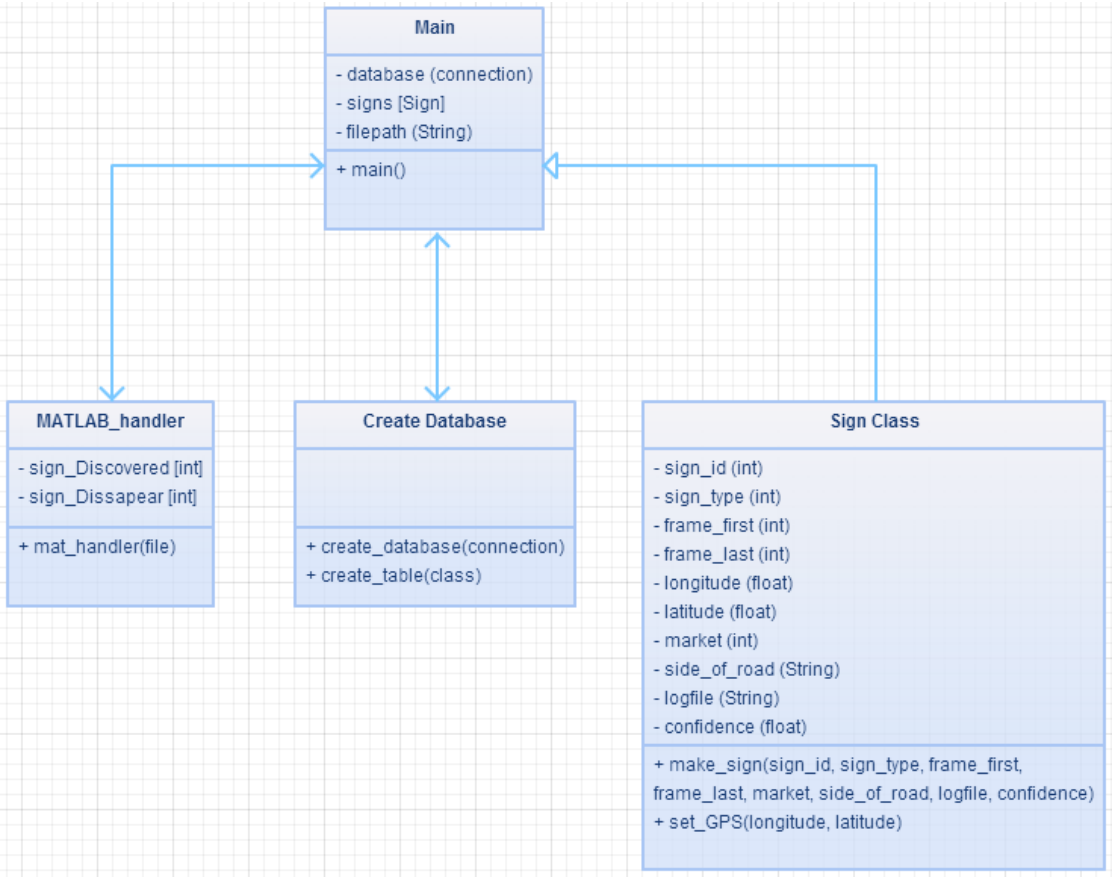


Figure 4.1: A simplified diagram of the prototype tool structure.

detections have been logged. Also the Global Positioning System (GPS) branch is closely checked as this often tends to be either missing or corrupt.

If data is available, the tool continues by traversing the traffic sign confidence matrix extracted from the .mat file in a row-by-row fashion, checking each column for confidence values differing from 0. If such a value is found it means that the TSR system has detected something. If it is the first frame this object is sighted in, the frame index is stored in a global array that consists of first sighting indexes, and also in the global array handling the last sightings. For every consecutive frame thereafter the object is detected, the last frame array will be updated with the new frame index.

When an object has failed to be detected for two video frames, the Traffic Sign object is created. All information regarding the sign is collected from the last frame the sign was detected in, with the addition of the prototype's internal representation of first and last frame.

The MATLAB-handler returns a list of all Traffic Signs that the TSR system has detected to the main method upon completion of handling the final matrix row. The main method proceeds by initiating a database session and inserting all signs found, and

finally committing the transaction.

4.2.2 Duplication Prevention

A problem introduced by dividing test runs into several log files is that parsing files individually may create duplicate traffic sign objects. The cause being the same sign occurring in the end of one log file and in the beginning of the consecutive log file.

The problem was solved with the usage of the two global arrays mentioned above. The traffic Sign object will not be created until the tool has failed to detect a sign for two frames. Since the arrays are global, they are kept consistent when starting handling of new files. Once a sign object has been created, it is removed from the global arrays to prevent duplication. This solution requires consistent ID assignment between consecutive logs to work properly, which luckily has been implemented by the developers of the TSR system.

4.3 Data Insertion

Information gathered from the .mat files and computed in the Python script is combined into an insertion query that is sent to a database. To facilitate this SQLAlchemy was incorporated into the project.

SQLAlchemy is an open source SQL tool kit makes the communications with the database SQL dialect independent, i.e. the Python script can communicate with any type of SQL based database. The queries are constructed as regular Python functions instead of detailed SQL statements. See Section 3.4 for a detailed description.

A small database was set up on localhost for testing of the database functionality in a controlled environment. Localhost was used for ease of access and proof of concept, a real implementation would require a dedicated remote host to enable easy access for remote users. Insertion functionality as well as querying was tested through SQLAlchemy. The dialect used was PostgreSQL, primarily because other databases at Delphi Automotive either currently uses, or plan to use, that specific dialect.

When creating the database one table is instantiated to hold all TSR discoveries, and contains all sightings of signs that the TSR system sees (including the erroneous ones). The detections are not accumulated in any way except for when a sign appear in multiple consecutive frames. Most often signs are in sight for at least 30 frames, a sign-object will not be created until the sign has been out of sight for at least two frames.

The one primary key for the table is the table's row sequence number. The TSR system does not provide any information that could be used as a unique id for consequent identification of spotted signs, nor any unique combination of output data due to a sign possibly appearing in several logs with imprecise GPS coordinates. To find a specific sign one need to know things like a rough GPS coordinate or what log it appears in. An example of what data is stored in the main table is seen in Table 4.1 and Table 4.2 continued.

UID	ID	Type	Confidence	First Frame	Last Frame
3	4	11	0.5	8641	8963
30	6	1	1	31569	31557

Table 4.1: First half of example table in database, UID being primary key.

Longitude	Latitude	Market	Side	Log file
11.9966	57.6877	7	Right	log_20140412_435671435.mudp
12.03329	57.6342	7	Left	log_20140412_435944872.mudp

Table 4.2: Second half of example table in database.

Once the information is organised in the database, conclusions can be drawn from it, specifically the number of correctly classified signs from which tables of incorrect signs can be produced.

4.4 Result

The created database is used for analysis of the data collected. The process of finding patterns in large collections of data is referred to as ”data mining”, with the intended usage of making huge data sets understandable and thus useful. This section will describe some ways the collected data can be used to draw conclusions regarding the performance of the TSR system.

Data mining is performed by creating queries that will produce a useful data set, usually in a statistical context. The database created can naturally be utilised by other services that could extract data sets automatically, but for the purpose of this project some examples were manually created to explain the use of such a tool. As an example set, all data for the 30 test drives in April 2014 was inserted into the database.

4.4.1 Definition of a Correct Sign

A detected sign is considered correct if it has the following properties:

- An ID value between 1 and 8.
- A Type value between 0 and 27 that represents what type of sign was detected, for example a speed sign for 70 kph has type id 1.
- A Confidence value between 0.95 and 1, or 0.5, depending on the type of the sign.
- Occurs in more than 4 video frames
- Is considered to be either on the left or right side of the road.

If the type of a detected sign is set to 0, the TSR system classifies it as a sign belonging to one of two supplemental sign categories, which are used for signs that are not considered vital for the system. For comparison one could mention a sign with a speed limit and a sign denoting distance to a city, only the latter will be classified as type 0.

A sign lacking the side of road variable is also marked as suspicious, as the TSR system clearly indicates that it cannot determine where the sign is located by returning a error indicating value. The side of road variable is based on the TSR system's lateral estimation of distance to the sign, if that is unknown a computation of more exact coordinates for the traffic sign cannot be made.

A traffic sign with confidence below 0.95 is treated as suspicious if belonging to the important sub-group, or 0.5 if belonging to the unimportant sub-group, will be flagged as incorrect for manual analysis. The 0.95 limit is based on how other projects at Delphi Automotive use them, and what assertions they made. Unfortunately it was not possible to get a more accurate motivation, as again it was not possible to access the documentation.

The reason for flagging signs appearing in fewer than 4 video frames is that a 'ghost' sign can appear where a previous sign was detected, suddenly appearing for a few frames and then vanishing. Observations of these events has led to the threshold being set to 4 frames as a first attempt at finding these events. It should be noted here that 4 video frames correspond to 8 internal system frames as it alternates between updating the radar and vision system.

4.4.2 Statistical Visualisation

To produce a complete performance evaluation for the period all signs were extracted and then grouped by type. The signs that did not fit in the definition of a correct sign (see Section 4.4.1) were deemed suspicious. The entire diagram is seen in Figure 4.2. Note that not all sign types are included, just the ones that the system detected. The performance of the TSR is the percentage of all signs that was deemed correctly classified by the system.

For April 2014 41037 detections were registered, 23197 was deemed correct, this gives 56.6% correct detections. Furthermore the average confidence value for each sign type can be seen, as a green dotted line in the figure and the correct classifications as a solid red line.

The two signs for Town are considered correctly classified with a confidence value of 0.5 for reasons known to the third party manufacturing the visual detection system. The type Unknown is for supplemental signs that the system recognises, and has been entered into the detections. They have been grouped together as there are very few sightings of these signs and they do not offer any insight into the functionality of the prototype.

Another conclusion from this figure is that there clearly are problems in relation to the "Stop" and "No Entrance" signs, as their percentage of correctly classified signs are very low but their confidence values are very high. There are clear problems with

	Unknown	Speed Limit Start	Speed Limit End	Highway Start	Highway End	Freeway Start	Freeway End	Town Start	Town End	Stop	No Overtaking Start	No Overtaking End	No Entrance	End Of All
Suspicious	1161	1527	9	122	47	7	0	1774	0	4085	33	4	9219	22
Correct	4603	9155	55	346	301	14	14	7319	4	23	255	62	898	148
Avg Conf Value	92,2%	94,0%	99,8%	91,0%	99,1%	92,9%	100,0%	52,9%	50,0%	100,0%	98,6%	99,9%	99,5%	98,3%
Correct Classifications	79,9%	85,7%	85,9%	73,9%	86,5%	66,7%	100,0%	80,5%	100,0%	0,6%	88,5%	93,9%	8,9%	87,1%

Figure 4.2: All signs detected during April 2014, in total a yield of 56,5% correctly classified signs.

these types of classifications, closer studies of these logs revealed that all types of signs that look alike the ”No Entrance” sign, such as e.g. bus lane signs, are classified as ”No Entrance” or ”Stop” rather than ”Unknown”.

The overall confidence values are high, even the town sign as their maximum is 50%. The average confidence value is taken over all detections and should not reach an average of a 100% except if something is wrong since it increases as the vehicle gets closer to the sign. An example of this the Stop sign, which has 4085 detections, only 23 are correct, but the average confidence value is 100% with 0,6% correct classified. Clearly this is not the intended behaviour and requires further analysis.

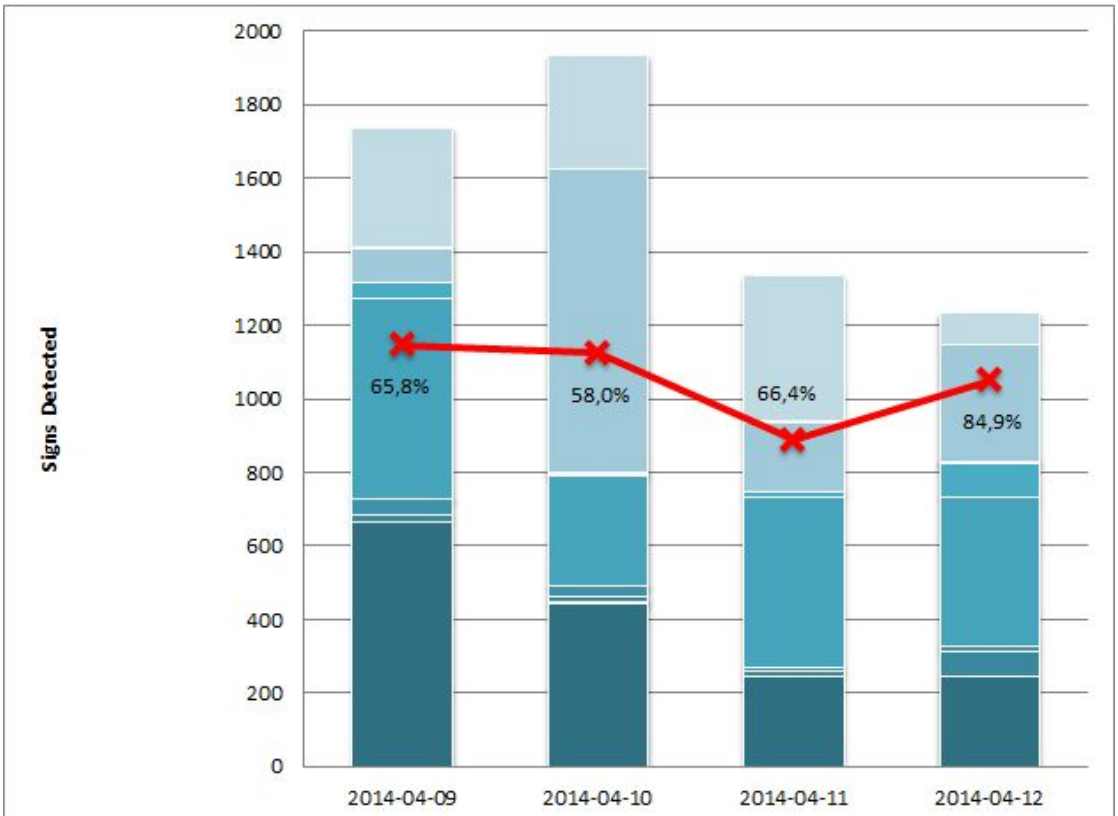
In the next diagram, Figure 4.3, four specific test drives have been chosen for analysis. They were performed during daytime for 4-6 hours, 9 to 12 April, 2014, in Sweden. The columns show the amount of signs detected per test drive. In this chart all detections are represented, the red line shows the percentage of correctly classified signs of each set, the exact percentages can be seen in Table 4.3.

Since most of the correctness of a sign classification is based on the confidence value, the average for each sign type has been included. As mentioned above the correct classifications paired with the average confidence value indicates problems in the TSR. Looking at both Figure 4.2 and 4.3 it is evident that certain sign types have higher values in both categories and thus the detections of these types are more credible. Such signs are primarily ”Speed Limit Start”, which is arguably an important sign in the traffic environment. Less important ones, such as ”No Entrance” are weaker in their classifications. On could of course argue that all traffic signs are of importance, but

there is a priority system in place in the TSR used that differs between certain types of signs, only some are considered important, such as speed limit or highway.

As mentioned these are just examples of what could be extracted from the data set. In Table 4.3 it could most definitely be of interest to include other statistical data, such as average number of frames a sign is detected. The different frame indexes in TSR and Data Visulisation Tool (DVtool) are both limited variables and will start over from zero once the maximum is reached. That presents difficulties when trying to determine an average and the current result is not very interesting as it is negative quite often. This is again something that would be suitable to implement in a future complete tool.

Comparing Figure 4.2 and 4.3, one can draw the conclusion that even if there are common misclassification problems they are not very apparent when looking at small data sets.



	2014-04-09	2014-04-10	2014-04-11	2014-04-12
Unknown	324	312	399	88
End Of All	4		4	
No Entrance	93	823	185	318
No Overtaking End				2
No Overtaking Start		4	4	2
Stop	43	8	13	90
Town Start	546	297	462	405
Freeway Start	2	2		
Highway End	43	29	9	15
Highway Start	17	14	17	71
Speed Limit End		2		
Speed Limit Start	667	446	245	244
Correct Classifications	1145	1124	888	1049

Figure 4.3: All signs detected during four test drives, each 4-6h long during daytime, 9 - 12 April 2014. The red line denotes the correctly classified signs according to Section 4.4.1.

	2014-04-09		2014-04-10		2014-04-11		2014-04-12	
	Correctly classified signs	Average confidence value	Correctly classified signs	Average confidence value	Correctly classified signs	Average confidence value	Correctly classified signs	Average confidence value
Unknown	85,6%	99,3%	88%	99,9%	83%	98,4%	93,8%	99,7%
Speed Limit Start	92,7%	97,4%	87,7%	97,1%	77,6%	95,7%	90%	98,9%
Speed Limit End			100%	97,2%				92,4%
Highway Start	57,1%	85,7%	76,5%	87,5%	76,5%	85,7%	76,1%	92,4%
Highway End	100%	100%	100%	100%	89,7%	96,2%	76,7%	100%
Freeway Start	0%	100%	100%	100%				
Freeway End								
Town Start	81%	54,4%	84%	53,7%	83,8%	54,4%	76,9%	52,2%
Town End								
Stop	0%	100%	0%	100%	4,7%	100%	0%	100%
No Overtaking Start			100%	100%	100%	100%	100%	100%
No Overtaking End							100%	100%
No Entrance	9,6%	99,8%	28,6%	98,5%	20,4%	98,7%	52,7%	96,5%
End Of All	100%	99,4%			50%	100%		

Table 4.3: Table showing the correctly classified signs and the average confidence value for all detections and sign types, for the test drives in Figure 4.3.

5

Discussion

During this thesis project a tool for automated analysis have been designed and the possibilities of including further features utilising external data have been investigated. There have been some unexpected technical difficulties and the lack of documentation has been a problem.

The outcome and conclusions of this project are confined by two factors; the confidence variable in that it is of such a closed nature and the available Global Positioning System (GPS) coordinates due to their imprecise nature. The confidence value while still presenting an obstacle, is not a variable that can be accessed and is thus not a constraint as such, more of a part of the environment that this project was produced in.

The GPS inaccuracy makes it much harder to identify traffic signs in crowded environments where they can be placed close together. A more precise hardware setting would be able to provide a distinction between traffic signs, potentially in combination with a solution of several database tables. One table in such a solution could store information regarding unique traffic signs while another holds metadata regarding each log file. By combining these two tables a test drive could be paired with the unique traffic sign objects.

Nonetheless, it is the opinion of the authors that the project brings a new aspect to the concept of Traffic Sign Recognition by not focusing on the collection of but rather the availability and usefulness of said data.

The investigations into possible features have showed that with some effort they are all feasible, the real problem lies in securing sources for various information. The most interesting is perhaps the concept of using OpenStreetMap or a similar service for access to various information regarding the GPS positions of the test car.

It is the hope that in some way this project has contributed to the progress of a fully autonomous car, even if it initially may only be for limited use. By helping in the testing process of improving the collection and sorting of traffic signs, complete traffic sign lists

for individual roads may in the future be compiled thereby enabling car trains to travel on our roads in a safe manner. The benefit of such lists would be that even when traffic signs are missed the cars would still be able to drive according to the rules, effectively emulating a driver's knowledge of often travelled routes.

The different approaches taken to implement the prototypes are regarded as sound. Choosing the MATLAB files as a data source over a raw data extraction proved to save time and Python proved to complete its task faster than MATLAB. The one point that can be debated regarding these choices is whether a different programming language should have been selected. The selection of feasible languages becomes quite small since such a language needs to have better performance than Python in combination with MATLAB support, as well as an intuitive language structure. Due to the limited time frame this project was produced under, the level of ease with which the tool could be programmed with was a factor in choosing programming language.

During the studies for this project, very few projects were found that covered the same area, except for the image recognition part, and most projects are dating back a few years. The data collected is not really of any academic use, but rather of use to different companies or departments. An automated system for identifying and reporting faulty signs to the responsible governmental department could perhaps be of interest for future use. With millions of cars and with the potential of one camera and GPS receiver in each, infrastructural damage with could be automatically reported within minutes of occurring.

The classification of suspicious traffic signs is based on observed phenomena both during this project and other projects at the same department, occurring in the examined log files, a 'real' classification would need further more precise definitions. Also, the signs covered are 'important' as signs focusing on informational matters (such as school signs) have been group together in the results. These signs would of course need to be included in a 'real' classification as the system is designed to handle them.

An area of future improvement is an automatic table and graph generator for the database. Using raw SQL or SQLAlchemy statements to find datasets of interest and automatically generate a visualisation of that data could greatly help in spreading information regarding performance issues in the system. The results supplied are just an example of what can be achieved with a database of the type implemented. While just an example, it does point out some issues with the data extracted.

With the functionality shown, focus can also be switched onto the area of performance and investigate what can be improved upon. If a solution such as this were to run on a computational cluster, what would the implication be, and how should some type of concurrency be devised? The automisation was outside the scope of this project, but using the prototype design supplied in this report a continuation of the tool could be developed in a future project.

5.1 Future Work

There are features that could not be implemented with the sole support of the existing local data as they required input from external sources. The features specified were road type classification (Section 5.1.1), country of origin lookup (Section 5.1.2), and determining weather conditions (Section 5.1.3).

During the literature studies some articles were found that attempted something similar concerning the road type classification that sparked the idea of investigating the possibility of using online map resources and include that data in the prototype tool. This was expected to be difficult due to there not existing any global dedicated database on the topic of road to road type correlation [16–18].

The country determination problem is something that can be done either on- or offline. It was considered a low priority during this project as it would be automatically solved with a successful implementation of a road type correlation as described above.

Concerning the weather information there is a sensor in the system that can sense rain (that activates the windshield wipers) but that is not good enough. The requirement is information regarding type and amount of precipitation, temperature, clear skies of thick clouds etc. This type of information is known to a meteorological service provider, and a suitable service was found through examining where other companies that require weather information fetched their data from. As with the road type correlation, this was an unexplored area in terms of accessible databases, but in the end it was not accessibility that was the issue.

For these three features and the sources investigated they all fell on the same point: calls per day. In the department of Electronics & Safety there are several petabytes of logs as the test cars are continuously driven for long periods of time. To be able to efficiently incorporate the requested features without making the already ten hour long post-analysis significantly longer, would require efficient access to the services found. This bottleneck problem could not be solved during this project, the details for each feature are stated below.

5.1.1 Road Type Classification

One of the ideas behind the project proposal from Delphi Automotive's side was to potentially eliminate or at least minimise the manual ground truthing of road types. Ground truthing in this case is the process of identifying certain objects in the videos and determining what they are, e.g. types of roads. The reason for establishing what type of road a vehicle has driven on is that customers state requirements regarding what road types tests are run on, e.g. 25 000 km highway. These demands are placed to make sure vehicles on our roads can safely handle different types of roads and conditions without failing.

The idea was to correlate the vehicle's GPS coordinates with road types. The first source thought to provide the service needed was Google Maps (see Section 2.7). However, reading the API for Google Maps showed a huge problem as the API stated the following: [24]

Users of Google Maps free service:

- 2 500 requests per 24 hour period.
- 5 requests per second.

Users of Google Maps for Work service:

- 100 000 requests per 24 hour period.
- 10 requests per second

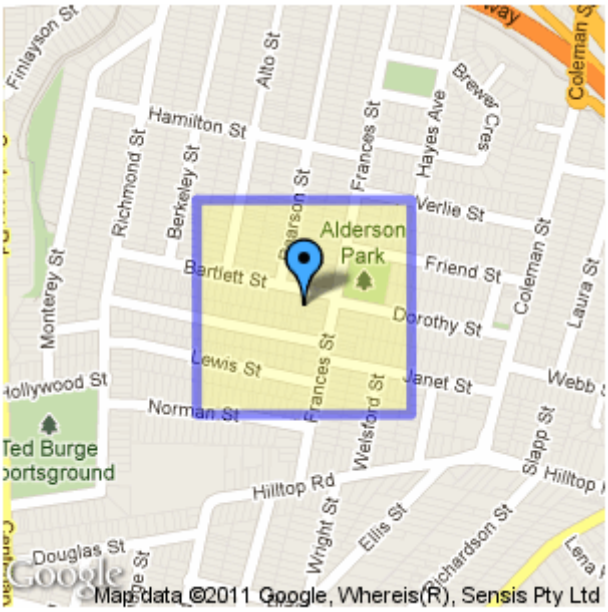


Figure 5.1: An illustration of the concept of a bounding box [24].

A standard log file has one GPS position update every second. So even for a short log file that only spans 60 seconds and therefore requires 60 lookups, the Google lookup alone will take 6 seconds. That is if the company is willing to pay for the service, using the free alternative it will instead take 12 seconds. As comparison, the test program written handles almost two entire log files in one second, leading to the lookup being a potential performance bottleneck.

Furthermore, Google Maps does not provide any command to look up information using GPS coordinates. Searching for an address will return the coordinates, but not vice versa. Feeding the web version with GPS coordinates works fine, but assigning a person to this is not a desired solution.

An alternative was found: OpenStreetMap, a collaborative mapping wiki. The benefit of OpenStreetMap compared to Google Maps is that data can be downloaded from OpenStreetMap and stored locally. Once downloaded, it can be processed through Osmosis, a command-line Java tool designed for the task. Osmosis has a function for correlating GPS-positions to roads through a square bounding box through giving the program 4 coordinates, each a corner of a square, as illustrated in Figure 5.1. Any node within the bounding box will be returned as a result from this query [25, 26].

OpenStreetMap represent roads by lines between two points, which can be several

kilometres apart. This generally means that the road must have either a starting or end point within the bounding box. Intersections between roads are also considered nodes so these will also yield valid returns when they are within the bounding box [25].

With the precision of the test car's GPS coordinates only being accurate to within 15 meters (see Section 3.3.4), basing a bounding box on these coordinates there is a tangible risk nodes will not be properly detected and thus no road will be identified. Furthermore, it is highly likely that the two nodes will occur in different log files, which means a solution must retain its last value between log files.

There is a good chance this could be solved through tweaking the size of the bounding box, but would require tweaking to get it just the right size. The tweaking part would be setting the bounding boxes large enough to find the nodes of the road the car is currently on without accidentally getting data about unnecessary byroads. The size of the box could perhaps be made dependant on the velocity of the car at the moment, since the faster the car goes the further it will be between GPS points.

Solving this problem with the OpenStreetMap system might seem complicated, but by only returning roads (excluding tram lines, air lines, waterways etc.) and comparing the last node visited with the new one, the road that has just been travelled (that is, has an entry in both nodes) should be the only common factor. Subsequently, the road type of all sampled data points between these nodes are set to the road type of the current road.

Also, correlation between the classification system used by OpenStreetMap and the different country dependant classifications must be done for each country since there are differences in road construction between them.

As a final note, implementation of the correlation method described above has another benefit: OpenStreetMap contains more data regarding roads than just road type, and since the lookup is already in place it is a moderately small task to extract additional information. Perhaps most interesting is a tool to graphically show which route a vehicle has driven, and with all the nodes already extracted, it would merely be a question of plotting them in OpenStreetMap [25].

In the end, it was decided against implementing such a system as it was considered to require a lot of time spent on details to provide the desired function.

5.1.2 Country lookup

There are two ways of attempting a country lookup, either on- or offline. The online method has a clear advantage in the sense that all the technical bits are taken care of and it is just a matter of supplying the service with GPS coordinates. However, considering the amount of existing log files, even if just one coordinate per log file was used for a look up the daily limitation would soon be hit, as discussed in the previous section.

An offline approach is therefore favourable but comes with its own set of problems. Country borders are readily available from various sources such as thematicmapping.org. With such a mapping, the Geospatial Data Abstraction Library for Python shows some promise in assisting with the implementation [27, 28].

Implementation of either solution seems feasible, but was low prioritised since a successful implementation of road type correlation would yield this information as well, as OpenStreetMap also contain country data regarding each street.

5.1.3 Meteorological Data

Implementing a function for weather look-up was high on the list of desired features that came up along the way, and does not appear too demanding to construct. However, it proved quite hard to find a source for global weather data with a reasonable amount of allowed calls per minute. The service found, which seems promising, was Weather Underground. They have a developer's API with extensive documentation and even a free mode for developers to get going.

But just as was the case with previously mentioned Google Maps, Weather Underground has a limitation on the amount of incoming calls allowed per minute.

Weather Underground free service:

- 500 requests per 24 hour period.
- 10 requests per minute.

Weather Underground Drizzle service:

- 5 000 requests per 24 hour period.
- 100 requests per minute

The "Drizzle" costs 20 USD per month, which is quite reasonable for that amount of calls, but does not include historic data, only current. Historic Data is free for the free user but adds 500 USD per month to the "Drizzle"-user. There are even higher user levels available and they offer custom packages tailored to a company's needs. [29]

An idea to limit the number of calls performed would be to only check a few values for each log file, say 1 or 2 evenly distributed over the log's timespan. With this technique, the day limit of 500 requests would still be hit after 500 to 250 log files and a schedule would need to be set up for when to run the script and with which files.

There is also the tempting possibility to import the historic data and use it to classify all the logs according to their meteorological conditions, but considering that there are several petabytes of logs and the limited calls per day, this would take much to long to be interesting. As said there is the possibility to come to an arrangement with Weather Underground, but it was considered to be outside of the scope of this project to investigate a possible negotiation.

It was decided against attempting to implement such a function at this time due to the above restrictions, leaving it for future implementation.

6

Conclusion

This project set out to facilitate the testing process at the Electronics & Safety department by making low level data accessible and comprehensible through synthesis. The collected low level data was derived from several subsystems, and was fused into high level information that makes the system behaviour and parameters easily accessible for analysis and evaluation.

A prototype of the suggested tool was implemented and the workflow is described in this report. The tool synthesises information from several subsystems, i.e. Global Positioning System (GPS) coordinates, visual feeds and Controller Area Network (CAN) data, and is fused into high level information, i.e. information suitable for graphs or tables. The low level data is retrieved from log files generated during test drives, synthesised into MATLAB structures, read into Python dictionaries, and finally inserted into a database for enabling easy data retrieval and examination. As an example of this, some diagrams were generated to demonstrate the type of high level information that can be extracted from the database with the use of data mining, these can be found in Section 4.4.2.

This type of statistical representations can be used to show what improvements have been yielded from an upgrade of a Software Component (SWC) can be clearly presented and analysed, for instance by storing data concerning different Resimulations (ReSims) in different tables and comparing them. To synthesise data from several subsystems is a vital part in the evaluation process as several of the systems need to function in harmony. Though as the data in the test equipment is in a low level format, the fusion into high level information is required for human assessment. Statistical data can be used to help streamline the testing and verification process by minimising manual input, saving a lot of time and resources that is better spent on making the traffic environment safer for everyone.

The final prototype is written in Python and it is used as a connecting step between the MATLAB files (containing the data from the log files) and a PostgreSQL database,

and has proved efficient and expedient during construction, handling of data points and analysis. Furthermore, Python has during our tests proved to be faster than MATLAB in many aspects. This has led us to recommend the Electronics & Safety department to look into the substitution of MATLAB with Python on the calculation cluster.

During this project, a great deal of effort has been put into finding data sources for the functions Delphi Automotive wanted to discern whether they were feasible to implement, either in a complete tool or this initial prototype. The areas that have been looked into are all deemed feasible to implement in a complete tool, and are presented below:

Road type: To be able to determine what type of road the vehicle is on the tool will need a connection to OpenStreetMap, or similar service, which is further discussed in Section 5.1.1. Recognising road types based on speed signs, road width etc. is a quite heavy computational operation and a solution using external data lookups could save time and energy.

Country lookup: The lookup support is partially already in place as there is a data point for which *market* the vehicle is currently in, but does not narrow it down to a single country. Implementating a more exact function is not considered a huge undertaking and is advisable (see Section 5.1.2).

Meteorological data: Adding meteorological information is viable and a possible solution is to utilize Weather Underground, and there is no apparent reason not to try to implement this in the future as it shows promise in delivering the services wanted; Specifically the possibility to use their archives of historical weather conditions and mapping it to previously recorded logs (see Section 5.1.3).

Precision of GPS receiver: During this project it was discovered that the precision of the GPS module used in the test cars at Delphi Automotive appear to be too imprecise for the desired usage. It is strongly recommend that Delphi Automotive reviews its hardware and hardware settings for the GPS before trying to implement advanced functions which need any GPS coordinates. See Section 3.3.4 and 5.1.1 for further details.

This project fits well in this general movement in the field of safety systems as they need not only be developed but also thoroughly tested to guarantee proper function. The results from this project are promising and the area of Active Safety Systems (ASS) is without a doubt one that will grow immensely in the foreseeable future, and with more data being collected the need for automatic data manipulation will only become more dire.

To conclude, the tool designed extracts low level test data from log and MATLAB files, and synthesises that data into high level information which can easily be comprehended by a human. Completely developed it is the opinion of the authors that this tool would increase the efficiency of the testing and verification process at the department of Electronics & Safety at Delphi Automotive.

Bibliography

[1] Adaptive Cruise Control with Steer Assist [Homepage]. Gothenburg: Volvo Car Group; 2013 [updated 2013 Jul 05; cited 2014 Sep 29]. Descriptional text. Available from: https://www.media.volvocars.com/global/en-gb/media/videos/49628/adaptive-cruise-control-with-steer-assist.

[2] Eskandarian A. Handbook of Intelligent Vehicles. Springer; 2012.

[3] How does GPS work? [Homepage]. Amsterdam: TomTom; 2013 [updated -; cited 2014 Sep 29]. How it all began. Available from: http://www.tomtom.com/howdoesitwork/page.php?ID=6&CID=2&Language=1.

[4] Hofmann-Wellenhof B, Lichtenegger H, Wasle E. GNSS - Global Navigation Satellite Systems: GPS, GLONASS, Galileo, and more. Springer; 2007.

[5] Garmin International I. GPS 16x Technical Specifications. Garmin International, Inc; 2008.

[6] Mathworks: R2014b Documentation [Homepage]. Natic, MA, USA: Mathworks Inc.; 2014 [cited 2014 Oct 5]. Traffic Warning Sign Recognition. Available from: http://www.mathworks.se/help/vision/examples/traffic-warning-sign-recognition.html.

[7] Stein GP, Gat I, Hayon G. Challenges and solutions for bundling multiple DAS applications on a single hardware platform [Research article]. Jerusalem: Mobileye Vision Technologies Ltd.; 2008.

[8] Zaklouta F, Stanciulescu B. Real-time traffic sign recognition in three stages. Robotics and Autonomous Systems. 2014;62(1):16–24.

[9] Greenhalgh J, Mirmehdi M. Real-time detection and recognition of road traffic signs. Intelligent Transportation Systems, IEEE Transactions on. 2012;13(4):1498–1506.

[10] Khan JF, Bhuiyan SM, Adhami RR. Image segmentation and shape analysis for road-sign detection. Intelligent Transportation Systems, IEEE Transactions on. 2011;12(1):83–96.

[11] Dagan E, Mano O, Stein GP, Shashua A. Forward collision warning with a single camera. In: Intelligent Vehicles Symposium, 2004 IEEE. IEEE; 2004. p. 37–42.

[12] Creusen I, Hazelhoff L, de With P. A semi-automatic traffic sign detection, classification, and positioning system. In: IS&T/SPIE Electronic Imaging. International Society for Optics and Photonics; 2012. p. 83050Y–83050Y.

[13] García-Garrido MA, Ocana M, Llorca DF, Arroyo E, Pozuelo J, Gavilán M. Complete vision-based traffic sign recognition supported by an I2V communication system. Sensors. 2012;12(2):1148–1169.

[14] Eichner ML, Breckon TP. Integrated speed limit detection and recognition from real-time video. In: Intelligent Vehicles Symposium, 2008 IEEE. IEEE; 2008. p. 626–631.

[15] Cova MJP. Global Localization of Vertical Road Signs using a Car Equipped with a Stereo Vision System and GPS [Thesis]. Faculdade de Engenharia da Universidade do Porto. Rua Dr. Roberto Frias, s/n 4200-465 Porto PORTUGAL; 2011.

[16] Yan WY, Shaker A, Easa S. Potential Accuracy of Traffic Signs' Positions Extracted From Google Street View. Intelligent Transportation Systems, IEEE Transactions on. 2013;14(2):1011–1016.

[17] Kim GH, Sohn HG, Song YS. Road Infrastructure Data Acquisition Using a Vehicle-Based Mobile Mapping System. Computer-Aided Civil and Infrastructure Engineering. 2006;21(5):346–356.

[18] Hu J, You S, Neumann U. Approaches to large-scale urban modeling. Computer Graphics and Applications, IEEE. 2003;23(6):62–69.

[19] Undocumented MATLAB [Homepage]. Undocumented MATLAB; 2014 [cited 2014 Dec 1]. Improving Save Performance. Available from: `http://undocumentedmatlab.com/blog/improving-save-performance`.

[20] SciPy.org [Homepage]. SciPy developers; 2014 [cited 2014 Nov 10]. Homepage. Available from: `http://www.scipy.org/`.

[21] Calculate distance, bearing and more between Latitude/Longitude points [Homepage]. 31 York Terrace, Cambridge CB1 2PR: Movable Type Ltd; 2014 [cited 2014 Nov 21]. Distance. Available from: `http://www.movable-type.co.uk/scripts/latlong.html`.

[22] sqlalchemy.org [Homepage]. The Python SQL Toolkit and Object Relational Mapper; 2014 [cited 2014 Nov 20]. Homepage. Available from: `http://www.sqlalchemy.org/`.

[23] Mathworks: Pricing and Licensing [Homepage]. Natic, MA, USA: Mathworks Inc.; 2014 [cited 2014 Nov 12]. Database Toolbox. Available from: `http://se.mathworks.com/pricing-licensing/index.html?prodcode=DB&s_iid=main_pl_DB_tb`.

[24] Google Maps API Web Services [Homepage]. 1600 Amphitheatre Parkway, Mountain View, CA 94043: Google; 2014 [cited 2014 Nov 26]. Usage Limits. Available from: `https://developers.google.com/maps/documentation/geocoding/#Limits`.

[25] OpenStreetMap Wiki [Homepage]. 132 Maney Hill Road, Sutton Coldfield, West Midlands, B72 1JU, United Kingdom: Openstreetmap Foundation; 2014 [cited 2014 Nov 27]. Map Features. Available from: `http://wiki.openstreetmap.org`.

[26] Osmosis - OpenStreetMap Wiki [Homepage]. 132 Maney Hill Road, Sutton Coldfield, West Midlands, B72 1JU, United Kingdom: Openstreetmap Foundation; 2014 [cited 2014 Nov 27]. Downloading, Usage. Available from: `http://wiki.openstreetmap.org/w/index.php?title=Osmosis&oldid=1096036`.

[27] thematicmapping.org [Homepage]. Bjørn Sandvik; 2014 [cited 2014 Nov 25]. Downloads - World Borders Dataset. Available from: `http://thematicmapping.org/`.

[28] GDAL - Geospatial Data Abstraction Library [Homepage]. GDAL; 2014 [cited 2014 Nov 25]. Homepage. Available from: `http://www.gdal.org/`.

[29] Weather Forecast and Reports - Long Range and Local [Homepage]. 300 Interstate North Parkway SE, Atlanta, GA 30339-2403: The Weather Channel Interactive, Inc.; 2014 [cited 2014 Nov 27]. Weather API for Developers, Historical Weather. Available from: `http://www.wunderground.com`.