Adaptive Automotive Chatbot

Tiago Miguel Moreira Ferreira

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Henrique Daniel de Avelar Lopes Cardoso

July 23, 2017
Adaptive Automotive Chatbot

Tiago Miguel Moreira Ferreira

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: Doctor Pedro Souto
External Examiner: Doctor Luís F. Teófilo
Supervisor: Doctor Henrique Lopes Cardoso

July 23, 2017
Abstract

With the increasing number of everyday objects having an Internet connection and being able to send and receive data – the so called IoT (Internet of Things) – the need to have systems that interact with such objects has also grown. This need has grown so much that at this time most manufacturers already ship software designed to interact with their products. However, this typically requires the user to install a separate application for each product and, most of the times, to navigate through complicated menus just to perform some simple tasks.

The advent of chat bots has proven to be a worthy topic of research and development, with general acceptance from the target audiences, who view them as a replacement to standard mobile applications, one of the reasons being that bots can be used in their favorite messaging application. Most of the bots available in the market respond only to specific user input; however, with the recent advances in Artificial Intelligence and Natural Language Processing, this approach is being shifted towards engaging conversations with the users, serving as a way to engage the user with the product. This has been proven with the creation of personal assistants by some big companies like Google or Apple.

This dissertation aims to create a chat bot that extends the relationship of the users with their cars, allowing a user to easily make location-aware operations, check the car current status and perform operations on the car just by talking with the bot. For instance, the user can locate where the car is parked or even get notified when there is a problem with the mechanical system of the car. This is done by sending the car data through an OBD-II dongle with permanent 4G connectivity to an application server. The server receives user input that is processed using state of the art natural language processing systems as a service such as Amazon Lex, and sends commands back to the car if necessary. The bot learns user patterns by modeling past trips using the ARIMA statistical model. Since this chat bot is meant to be used while driving using voice to communicate, some challenges arise that are related with how to interact and direct the conversational flow, understanding the differences between texting and talking, along with accurately and at timely make proactive suggestions to the user.

To evaluate this work, the application was tested with usability tests performed by WIT Software personnel, where at the end, the participants were asked to fill a questionnaire regarding their experience. The pattern recognition and predictions were also tested against real data. The results from these tests were promising, with the pattern recognition algorithm returning values that can be considered accurate, and positive feedback from testers along with ideas for future work.
Resumo

Com o crescente número de objetos do dia-a-dia que contêm uma ligação à Internet sendo capazes de enviar e receber dados – a chamada Internet of Things (IoT) – a necessidade de ter sistemas que interajam com esses mesmos objetos também cresceu. Esta necessidade cresceu tanto que nesta altura muitos dos fabricantes destes objetos já distribuem software desenhado para interagir com os seus produtos. No entanto, isto requer que o utilizador instale uma aplicação diferente para cada produto que queira utilizar e que, na maior parte das vezes, seja obrigado a navegar por interfaces e menus complicados para realizar uma simples tarefa.

O advento de chatbots provou ser um digno tópico de estudo e desenvolvimento, com a aceitação por parte do publico alvo como uma substituição relativamente a aplicações móveis standard, sendo uma das principais razões o facto de bots poderem ser usados nas suas aplicações de troca de mensagens preferidas. A maior parte dos bots existentes atualmente no mercado, apenas respondem a input especifico por parte do utilizador; no entanto, com os avanços recentes nas áreas de Inteligência Artificial e Processamento de Linguagem Natural, esta abordagem tem sido alterada para uma conversacional, servindo como uma maneira de envolver o utilizador no produto. Este conceito foi provado através da criação de assistentes pessoais por grandes empresas como a Google ou Apple.

Esta dissertação tem como objetivo criar um chatbot que extenda a relação do utilizador com o seu carro, permitindo que este realize operações que tenham em conta a sua localização, verificar o estado do seu carro e realizar operações sobre este, apenas falando com o bot. Por exemplo, o utilizador pode localizar onde está estacionado, realizar ações baseadas na sua localização, ou até ser notificado quando existe algum problema mecânico com o carro. Isto é obtido através da transmissão de dados a partir do carro com um adaptador OBD-II com ligação 4G permanente para um servidor aplicacional. O servidor recebe input do utilizador que é processado usando sistemas atuais de processamento de linguagem natural as a service, tal como o Amazon Lex. O bot aprende com o utilizador modelando as suas viagens através do modelo estatístico ARIMA. Uma vez que este bot é feito para ser usado usando através de comunicação por voz, alguns desafios levantam-se que estão relacionados com como interagir e direcionar o fluxo de conversação com o utilizador, percebendo as diferenças entre falar e mandar mensagens de texto.

Para avaliar este trabalho, a aplicação foi testada com testes de usabilidade com trabalhadores da WIT Software, onde no fim, os participantes convidados a fazer uma avaliação da sua experiência através de um questionário. O reconhecimento de padrões e previsões foram também testados com dados reais. Os resultados destes testes foram promissores, com o algoritmo de reconhecimento de padrões a retornar valores que podem ser considerados precisos, e feedback positivo juntamente com ideias para trabalho futuro por parte dos testadores.
Acknowledgements

First of all I would like to mention my parents and grandparents. I am deeply thankful for everything that they have done for me, for their support to my decisions while sacrificing themselves to my good, eventually leading to this moment.

I would also like to thank all my professors that accompanied me through this academic journey, specially to my supervisor that always supported me during this dissertation. To all my friends, old and new, that made this journey a lot easier; I’ve made friends for life. Additionally I would like to thank WIT Software, my supervisors inside the company that provided me with all the means to do this dissertation work.

Last but not least, I have to thank my girlfriend, she was always there for me during this whole project, motivating and supporting me even in my long periods of procrastination. Thank you.

Tiago Ferreira
“Time is what prevents everything from happening at once.”

Albert Einstein
# Contents

1 Introduction .......................................................... 1
   1.1 Context ......................................................... 1
   1.2 Motivation and Goals ......................................... 2
   1.3 Dissertation Structure ....................................... 3

2 Automotive data and intelligent systems ...................... 5
   2.1 Automotive data acquisition .................................. 5
      2.1.1 Proprietary applications .............................. 6
      2.1.2 On-Board Diagnostics .................................. 7
   2.2 Displaying information inside vehicles ................... 9
   2.3 Natural Language Processing ................................. 9
   2.4 Using NLP to build bots .................................... 10
      2.4.1 Tools to build bots .................................... 11
      2.4.2 Speech Recognition .................................... 13
      2.4.3 Existing bots .......................................... 14
   2.5 Detecting patterns in time-series ......................... 15
   2.6 Summary ..................................................... 17

3 Chatbot for automotive integration ........................... 19
   3.1 Architecture ................................................ 20
      3.1.1 Fetching the vehicle’s data .......................... 20
      3.1.2 NLP platform ............................................ 21
      3.1.3 Providing functionality when driving ............... 21
   3.2 WIT Bot Platform ............................................ 22
      3.2.1 Engine .................................................. 22
      3.2.2 Gateways ............................................... 25
      3.2.3 Human Assistance ....................................... 25
   3.3 Adapting to the user ......................................... 26
   3.4 Summary and conclusions .................................... 26

4 Implementation ..................................................... 29
   4.1 Server - WIT Bot Platform .................................. 29
      4.1.1 Technologies ............................................ 29
      4.1.2 Platform integration ................................... 31
   4.2 Mobile Application .......................................... 36
      4.2.1 External libraries used ................................ 38
      4.2.2 Establishing connection to the server ............... 39
      4.2.3 Connecting to the infotainment system .......... 40
## CONTENTS

4.2.4 Parsing server messages and display content ........................................ 40
4.2.5 Use cases ......................................................................................... 41
4.3 Summary and conclusions ...................................................................... 42

5 Experiments and results ................................................................. 47
  5.1 Questionnaire .................................................................................. 47
  5.2 Pattern Recognition experiments ....................................................... 48
  5.3 Usability tests .................................................................................. 49

6 Conclusions and Future work ............................................................ 51

References ..................................................................................... 53

A Questionnaire Form ................................................................. 57

B Questionnaire Results ............................................................... 61
List of Figures

2.1 myChevrolet application .............................................. 6
2.2 BMW ConnectedDrive .................................................. 7
2.3 EML327 OBD-II Bluetooth dongle ................................. 8
2.4 Dependency parsing example ......................................... 11
2.5 Chatbot categorization [cha17a] ....................................... 12
2.6 Example conversation flow ........................................... 13
2.7 Linear Regression example using a gold prices over time dataset [gol17] .................................................. 17
2.8 ARIMA One-Step-Ahead Forecast example using a gold prices over time dataset [gol17] ........................................ 18
2.9 ARIMA Seasonality forecast example using the Mauna Loa Weekly Atmospheric CO2 Data [co217] ........................................ 18

3.1 Proposed architecture ................................................... 20
3.2 Proposed architecture ................................................... 21
3.3 Example BPMN flow ..................................................... 25

4.1 BPMN flow implemented ................................................ 34
4.2 ARIMA model fitting analysis obtained while fitting driver data from the prototype application ........................................... 38
4.3 ARIMA results using real driver data obtained through the use of this application (dark areas in the plot represent the forecast confidence intervals) .... 39
4.4 Bot chitchat interaction ................................................ 42
4.5 Application main screen ................................................ 43
4.6 Application map activity ................................................ 44
4.7 Application OBD warnings ............................................ 44
4.8 Example notification warning .......................................... 45

5.1 ARIMA execution for low amount of data ................................ 50
5.2 ARIMA execution for medium amount of data (using 3 as parameter limit) ............................................... 50

A.1 Questionnaire form first page .......................................... 58
A.2 Questionnaire form second page ........................................ 59

B.1 Questionnaires’ question 1 results. .................................... 62
B.2 Questionnaires’ question 2 results. .................................... 62
B.3 Questionnaires’ question 3 results. .................................... 62
B.4 Questionnaires’ question 4 results. .................................... 62
B.5 Questionnaires’ question 5 results. .................................... 63
B.6 Questionnaires’ question 6 results. .................................... 63
LIST OF FIGURES

B.7 Questionnaires’ question 7 results. .................................................. 63
B.8 Questionnaires’ question 8 results. .................................................. 63
B.9 Questionnaires’ question 9 results. .................................................. 64
List of Tables

2.1 NLP bot development tools comparison ................................. 9
2.2 NLP bot development tools comparison ................................. 14
2.3 Speech recognition tools comparison ................................. 15
5.1 Pattern Recognition algorithm variances comparison .............. 49
5.2 Usability tests results .................................................... 50
## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
</tr>
<tr>
<td>NFC</td>
<td>Near Field Communication</td>
</tr>
<tr>
<td>GB</td>
<td>GigaBytes</td>
</tr>
<tr>
<td>CAN</td>
<td>Controller Area Network</td>
</tr>
<tr>
<td>OBD</td>
<td>On-Board Diagnostics</td>
</tr>
<tr>
<td>PID</td>
<td>Process ID</td>
</tr>
<tr>
<td>DTC</td>
<td>Diagnostic Trouble Code</td>
</tr>
<tr>
<td>ECU</td>
<td>Engine Control Unit</td>
</tr>
<tr>
<td>GPS</td>
<td>Global Positioning System</td>
</tr>
<tr>
<td>NLP</td>
<td>Natural Language Processing</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>POS</td>
<td>Part-Of-Speech</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>ARIMA</td>
<td>Autoregressive Integrated Moving Average</td>
</tr>
<tr>
<td>AIML</td>
<td>Artificial Intelligence Markup Language</td>
</tr>
<tr>
<td>BPMN</td>
<td>Business Process Model and Notation</td>
</tr>
<tr>
<td>XPDL</td>
<td>XML Process Definition Language</td>
</tr>
<tr>
<td>SMS</td>
<td>Short Message Service</td>
</tr>
<tr>
<td>RCS</td>
<td>Rich Communication Services</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>CSV</td>
<td>Comma-Separated Values</td>
</tr>
<tr>
<td>AIC</td>
<td>Akaike Information Criterion</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

This dissertation aims to develop a chat bot that makes use of the latest technologies in Natural Language Processing and the large amount of data that vehicles are able to provide nowadays. In this document the topics mentioned above will be explained in detail with the inclusion of examples. This introductory chapter will further describe the context, motivation and goals in which this dissertation comes from.

1.1 Context

The Internet of Things (IoT) is a concept that goes back to the nineteenth century [AIM17] and it is a fast growing ecosystem, still on its infancy, with a large room to grow [BDB16]. It builds on top of the concept of devices having a connection to the “outside world” using technologies like Near Field Communication (NFC), Bluetooth Low Energy or an Internet connection to send their data to the cloud [BDB16].

Due to its large scope, IoT has had a lot of impact in the market [NS17], one good example of this being the Amazon IoT button [ama17], which is a programmable device that allows to generate a custom order with just one click. But this world of the Internet of Things can also extend to vehicles since these can also be a good source of data. On [MRH+12] it is estimated that with 1 million vehicles generating only GPS data and simple IDs, in a single day, the total size of the data generated would be around 144GB. Considering that the average vehicle not only generates GPS data but also internal sensors data like speed information, fuel level and others, the amount of data expected to be generated increases significantly.

A good way to manage all this information and show it in a presentable way to the end-users is through applications. Another option that has been gaining popularity is the use of bots. Most of the bots available on the market can only respond to specific user inputs, but with the advances in Natural Language Processing, a sub-section of Artificial Intelligence, bots are becoming intelligent
systems that try to capture an intention from the user and take an appropriate action or present the correct information.

This dissertation is integrated in a project from the company WIT Software, that aims to create a bot platform, a framework to build bots and integrate them easily and quickly with all the different messaging applications (Facebook Messenger, Slack and others), as well as establish a connection between the bots, promoting cooperation and increasing their value. This document will focus on the analysis of the development of a bot designed to operate using automotive data.

1.2 Motivation and Goals

This dissertation makes sense in the context of the current state of vehicles’ information and connectivity, and the evolution of natural language processing techniques aiming to exploit these two realities to build an engaging and useful product to the user. To do this, there is a need to study and implement the components necessary to build a product that succeeds in combining these two technical areas, beginning on how to fetch information from a vehicle and how to build a bot to fulfill the user needs.

Analyzing the possibilities on how to interact with the user, a natural language processing bot has clear advantages over standard smartphone applications since these have a learning curve to adapt to the interface; moreover, it is often the case that to perform a simple task, the user has to do a lot of navigation. Other than that, the trend of installing new applications is stalling - an average user usually does not browse the application store to find applications of their interest, instead most installs come from recommendation or publicity. This is where a bot gains competitive advantage since it can be embedded in the users’ most used applications: messaging ones.

The main objective of the project is to implement a bot that understands the intentions of users and uses this information to enhance the relationship with their cars. For this, the bot must be able to have both reactive and proactive actions. For example, lets say that the user wants to know details about one specific sensor in the car and says “What is the fuel level?”; the bot must be able to respond with the correct value. In the case any problem occurs in the vehicle, the bot must proactively warn the user with messages like “The Transmission Control System has malfunctioned. You should visit a mechanic”.

Another objective is that the bot is able to learn with the user. To be able to get the user to use the bot, it has to provide useful and contextualized information; for example, if the user always makes the same trip in the morning, the bot should check the fuel level to see if it can complete the trip, warning the user if not. One other interesting objective is to integrate the bot and make it communicate with other bots in the WIT Bot Platform, increasing its value. With this functionality, the bot could for instance communicate with a bot that controls home appliances and automatically open the house gate when the vehicle is approaching.

Not only must this bot work with text, but it also should work when the user is inside the car, issuing voice commands – an interaction mode that reduces the danger of affecting driver’s security.
Introduction

Developing a bot like this raises some interesting challenges:

- Efficient communication with the user
  This is important because as said before, the bot needs to be able to be used while driving. To accomplish this, the bot must use the best possible way to interact with the user while still providing relevant information;

- User security
  Because important data is being transmitted between the user and the bot, it is important to make sure that the correct person is getting access to the correct car data, not violating user privacy. One other aspect is user physical security inside the vehicle and because of this, the best way to make the bot available to be used by users in-vehicle must be studied;

- User adaptation and personalization
  How to adapt to each user providing the best possible service, capturing user routines and daily patterns in order to provide with proactive and useful messages;

- How to provide suggestions
  There is also the need to know when and where (based on location), the bot can infer assumptions about the user and proactively send messages. This is a big issue because if the bot does not communicate with the users when they need it, then no value is being added. On the other hand, if the user is bombarded with messages, then it has the opposite effect: the bot is being too intrusive and the user will not use it anymore.

All these topics will be explored in this dissertation.

1.3 Dissertation Structure

Besides this introduction, the dissertation contains 5 more chapters. In Chapter 2, the state of the art is described and related projects are explored. In Chapter 3, the state of the art is analyzed, identifying a problem and a solution is proposed using the previously studied technologies. In Chapter 4, the implementation of the solution is presented, explaining how every component works, from the bot platform to the NLP engine and the pattern recognition module. In Chapter 5, the solution is evaluated regarding user acceptance as well as usability and adaptation to the user, in order to see whether or not the solution is feasible. Finally, in Chapter 6, the conclusions are exposed and an analysis of future work is made.
Introduction
Chapter 2

Automotive data and intelligent systems

This project addresses the theme of automotive data acquisition as well as Natural Language Processing and so, it is important to study the current state of the art. In this chapter, it is presented a background and the currently existing solutions and tools that tackle these themes as well as a comparison between them.

2.1 Automotive data acquisition

As mentioned in Section 1.1, cars can produce a large amount of data, and there are several ways to gather such data. The concept of Internet of Cars was first introduced in 2012 [SS12], where a pilot project used the car’s license plates as bar-codes to identify the car through 25 road-side cameras and used that information mainly for visualization purposes and raise awareness to a form of Internet hidden in plain view. More recently, in [JQM14], a programming framework named CARLOG was presented. This framework combines automotive data gathered from the CAN (Controller Area Network) bus of the car, with data gathered from external sources (weather, for instance) to provide developers with abstract methods to build automotive applications.

Another area where car data is very important is in racing competitions, mainly Formula 1 [f1d17]. Here, data gathering and analysis plays a big role, since competition rules and the current state of automotive engine development do no allow any big differentiation between teams. So, to gain an advantage over their competitors, engineers have to analyze data from practice runs in order to adjust the car’s body, as well as live communication during the race so that the driver can adjust the driving style in order to achieve their goal: victory.

Today, there are several ways to gather vehicle information as cars are equipped with new technology regularly. This makes it possible for companies like Veniam to develop technologies where cars are wireless hotspots that cooperate with each other to reduce the costs of data transmission [MLLPLF15]. In this section some solutions on how to gather that information will be presented.
2.1.1 Proprietary applications

One way to access car data is through the car constructors’ own applications. Because every car model is built differently and although there is a standard interface to provide car data to external sources, there are always differences in the sensors and the way they provide their respective data. These applications use proprietary protocols to access car sensors and therefore are able to provide the user a more complete experience and actions that other applications cannot, the most common examples being reading car lock/unlock status and lock/unlock the vehicle.

Chevrolet’s My Chevrolet [myC17] is a good example of that. It allows the user to check their vehicle information as well as perform some simple actions on it such as start and shut down. A screenshot of the application can be found in Figure 2.1.

Another good example is BMW’s ConnectedDrive [bmw17], that has the same functionality as My Chevrolet, but also has a reverse data flow, meaning that when the user connects the smartphone to the car’s infotainment system, it automatically syncs the user’s calendar and contacts, being able to read incoming SMS messages and emails. If those messages are meeting arrangements then the car displays to the driver the meeting request and an option to accept or decline it. By connecting to other BMW vehicles connected to the network, it is also able to know traffic light timers and read information from signs to aid the vehicle’s self-driving system and provide the user with the
best possible path to the desired destination. Figure 2.2 shows the application’s interface in the car’s infotainment system.

2.1.2 On-Board Diagnostics

On-Board Diagnostics, OBD for short, is an interface that allows access to the vehicle’s data following a well defined protocol and is available in any vehicle from 2000 or newer by legislation. This interface connects to a vehicle’s on-board computer and the engine control unit to allow both diagnostic data and trouble codes lookup. It has several modes where it is possible to specify what PID (Parameter ID) to read, and one mode to check all DTCs (Diagnostic Trouble Code) that are emitted by the ECU (Electronic Control Unit) when a value of a specific PID surpasses its limit. There are two possible ways to obtain this data, using Bluetooth or 4G dongles, both having their strengths and weaknesses that will be detailed in the following sections.

2.1.2.1 OBD-II Bluetooth dongles

One way to obtain OBD data is through an OBD-II Bluetooth dongle. This has the advantage of being able to obtain data from every single PID that exists without depending on external factors like the lack of network availability or service down-time. One approach to use these devices is explored in [AADN+16]; here the proposed architecture for an automotive data acquisition platform uses a dongle like the one in Figure 2.3 together with an Inertial Measurement Unit to measure the vehicles acceleration and a Raspberry Pi connected to a Wi-fi hotspot to send the data to a server. Because Bluetooth has a nominal range of only 10m [LSS07], in order to implement solutions with these devices you have to be very close to the vehicle, or use additional devices to be able to fetch the data remotely or get geographical information, as OBD can not access the vehicle’s GPS.
2.1.2.2 OBD-II 4G dongles

The other alternative to obtain OBD data is to use 4G dongles. These provide access to the data anywhere and come with an integrated GPS module, but are tied to the mobile carrier provider that establishes the 4G connection.

Currently there are 2 major providers of these devices that supply public APIs for developers, these being Automatic [aut17] and Vinli [vin17]. Both these platforms support the following services:

- **Platform**, where the developer can manage the devices associated with the project and perform operations such as list all the vehicles or devices currently active (one device is not tied to a single vehicle);

- **Telemetry**, where it is possible to read, just like with the Bluetooth devices, the vehicle’s PID;

- **Diagnostic**, to consult the currently active DTCs, providing not only their code but also an understandable description of the problem;

- **Real time events**, that can be subscribed and therefore receive notifications from them, either by web-sockets or HTTP POST requests.

On top of this, Vinli also provides the possibility to create custom rules and associate them with events. These rules can be either a geospatial or parametric boundary. Geospatial boundaries
define a circular geographic region and notify the user when the vehicle enters or leaves that region. Parametric boundaries follow the same approach but for telemetry PIDs, for example “RPM between 2000 and 4000”.

### 2.2 Displaying information inside vehicles

To show drivers information (relative to any kind of data), inside their vehicles is a hard task, mainly because of the regulations regarding user distraction and safety. Besides native integration with the car infotainment system (only car manufacturers can do this), there are 3 major frameworks that allow to build applications that then display their contents and interact with car systems (audio for instance).

Table 2.1 shows a comparison between those tools in terms of what kind of content can be displayed, if external certification is needed and if those technologies are available to the end-user easily. From this, we can conclude that although MirrorLink is the only one that allows for full dynamic content to be presented, it is also the only one that needs an external certification process to verify that content is according to standards and does not violate any security measure. Because of this and the existence of much simpler APIs, although limited to media player and messaging applications, Android Auto and CarPlay are gaining traction and threatening MirrorLink’s market lead.

### 2.3 Natural Language Processing

Artificial Intelligence is the science of making intelligent machines, and so, these need to understand user input. This can be provided to the machine in several ways, the most common is through well defined graphical user interfaces (GUI). But in order for a machine to be considered intelligent, it must pass the Turing Test and therefore understand the language in which humans communicate. In [WS16], a study conducted in 2014 is described, where judges evaluated whether or not they were talking with a machine based on how well it could understand the questions and respond accordingly.

Natural Language Processing is a subarea of Artificial Intelligence that focuses on extracting meaning from natural language input. In [CWB+11] an approach on how to do this is shown. This process consists on four major tasks:

- Tokenization
Automotive data and intelligent systems

- Part-Of-Speech tagging and parsing
- Named Entity Recognition
- Dependency Parsing

Tokenization is the process of splitting a document or sentence into tokens that can be words or even phrases. This process is trivial for the English Language, but for languages where there are no explicit word boundaries, like Chinese or Japanese, it is a complex task [SLC17, Chapter 3]. Based on tokenization, it is also possible to label each token with a named entity, identifying if the token is a location, an institution, or other.

Part-Of-Speech tagging and parsing consists on labeling the generated tokens with their syntactic role and build a tree with the sentence’s syntactic constituents (e.g. Noun Phrase).

Dependency parsing is considered the hardest, and aims to identify the dependencies of the nodes from the resulting tree of the POS parsing process. With this, it is possible to infer meaning in sentences and therefore understand it (Natural Language Understanding).

Doing this from scratch is a hard task, but there are some good tools that do it, the most efficient being Google SyntaxNet [syn17] and Stanford CoreNLP [cor17]. They both are built using neural networks and mostly trained for the English language, but provide options for other languages. In Stanford CoreNLP there is the possibility to import a different parser from the official parsers available. With Google SyntaxNet, there is the possibility to train a model from scratch using their neural network by proving training data for POS and dependency tagger.

In [syn17] it is also described why it is hard to do dependency parsing, proving a good example for the English language. In this example, shown on Figure 2.4, the sentence “I saw the man with glasses” is analyzed. This can have a couple of different meanings, as “glasses” can be related with the pronoun “I” or the noun “man”, and that gives the sentence two different meanings. This problem is attenuated by exhaustively training the dependency parser. In Google SyntaxNet, the training data came from the standard corpora of the Penn Treebank [pen17] and OntoNotes [ont17], as well as the English Web Treebank [eng17], which contains among others, real conversation transcripts.

In [SLC17], a review of several NLP tools is made in the context of opinion mining and sentiment analysis, using several evaluation techniques.

2.4 Using NLP to build bots

One application of NLP is to build conversational applications, or in other words, chatbots. This serves as a way to save resources from companies as it provides solutions to problems that would need a real human being to solve. This is specially important in the costumer service area, where employees have a script that they have to conduct in order to solve the costumer problem. In [CL15] a chatter bot is presented, evaluating how it performs with a simple architecture with just a knowledge and conversation engine, but also with a more complex approach using natural language processing and helper modules including sentiment analysis.
Chatbots can carry conversations either on an open or closed domain [cha17b]. Open domain conversations mean that the bot can identify and answer to user intent whatever the theme is. This is generally hard because it is impossible to define rules (containing a response) to each and every single possible topic, meaning that a deep learning model is needed to use available information, past conversations and context to generate an answer. Most bots available use a rule-based approach because more often than not, they are used to build customer service systems where there is no need to build intelligent agents to keep an active conversation with the user, or when a conversation is needed, it follows a well-defined and previously made script. Taking this in consideration, chatbots can be split into four categories, as shown in Figure 2.5. On this image we can see a visual representation of what was said in this paragraph.

Another aspect that influences difficulty in bots that use generative-based models is the conversation length. In longer conversations, there is the need to be aware of context, detecting whether or not the user enters a new context and storing all the needed variables.

To build chatbots there are two options: use a third party tool to handle all the AI and conversational part of the bot, detecting the user intent and responding accordingly (more detail on this in Section 2.4.1), or build a machine learning model to generate answers.

### 2.4.1 Tools to build bots

Wit.ai, api.ai and Amazon Lex are the current state of the art services to build bots, providing an easy to use interface and developer APIs, with a machine learning back-end that enables all bots in their systems to learn with each other in order to better recognize user intentions. These tools only work with closed domains and take a mixed approach on how to generate answers as there is the possibility to automatically answer with pre-defined sentences (rule-based), or call external code that generates an answer and communicates it back to the service (generative-based). All these services are based on the following concepts:
An intent is the user intention, for instance “BookTrip” defining that the user wants to book a trip. These intents are triggered by user input that has to be specified previously by using training sentences. Because the platforms use machine learning algorithms so that bots learn with each other, similar sentences to the ones used in training will also trigger the intent. These intents call actions that are not more than code that will execute when that intent is active and all the Entities/Slots are defined. Entities are variables that belong to intents and need to be defined in order to complete the intent. Using the same example as before, the “BookTrip” would need an entity to represent the date, and another to represent the location.

These services are also able to understand context, meaning that the same user input can trigger different actions depending on the situation, whether or not the user has already engaged on an intent or not. Figure 2.6 shows an example conversation flow, meaning that a message inside that flow will have the same context.

Although they are based on the same concepts, there are differences between these services. Table 2.2 compares the tools mentioned previously in terms of the number of languages supported, if they support voice inputs or not, if they natively integrate with external messaging services like

Figure 2.5: Chatbot categorization [cha17a]

- Intent
- Entities/Slots
- Action
- Context

Table 2.2
Facebook Messenger, if they proactively ask the user for required slots in an intent, and the price. With this comparison we can see that the main differences between the tools are in the languages supported, their ease of integration, and slot completion. Although currently supports only one language (US English) and has a small fee per request, it takes advantage of the fact that it is integrated in the Amazon Web Services ecosystem. As for integration, although Amazon Lex and wit.ai do not support native integration, they provide a documented API making it possible to build a simple webservice to connect them to messaging platforms.

2.4.2 Speech Recognition

Although, as seen before, tools to build bots already support voice input, there might be occasions where the voice has to be transcribed beforehand.

Currently some big companies provide speech recognition as a service through fairly easy to use APIs. Analyzing Table 2.3 that shows a comparison of the most popular APIs nowadays, we can conclude that two stand out more than the others: Google Cloud Speech and Wit.ai. These services provide support for over 40 languages, being able to target large audiences. As for the
fact of context awareness, Google takes a big lead, by having by far the largest userbase and being able to learn from users, and also take information from their devices such as location, to provide better results.

### 2.4.3 Existing bots

Chatbots have been developed for a long time, some interesting ones in the education area. In [BMS14] a study was conducted to measure whether or not students would engage more in classes by learning basic concepts of computer science building a chatbot with some premade templates. The results were clear: students participated 5 times more when using the chatbot rather than the traditional learning tool, due to gamification processes and being able to evaluate work they have in a conversational way.

Also in the topic of education, chatbots have also been used to teach foreign languages to students [FC06] and proved to be useful since student chatbots could repeat content endlessly and not get bored or lose their patience, allowing students to study whenever they want to.

In [KHB07], a chatbot is used to investigate the feasibility of using one to support negotiations, allowing students to talk to the bot about their C programming language skills.

Although some interesting work has been done in the past, not until now have chatbots become really popular. Taking into consideration Figure 2.5, we can point out at least one successful bot in each of the three feasible areas:

- **Closed domain / Retrieval-Based**: In this area fall most of the consumer assistance bots, one of them being the Pizza Hut ordering bot, that allows the customer to order a pizza using Facebook Messenger or Twitter.

- **Closed domain / Generative-Based**: Yeshi, a bot that impersonates an Ethiopian young girl who has to walk over 2 hours to get clean water; it comes with capabilities like geolocation, media sharing and Stripe integration, to raise awareness to the cause.

- **Open domain / Generative-Based**: This is the area that has really proven the concept of chatbots, mostly popularized by the creation of personal assistants by some big companies like Google with the Google Assistant [goo17], or Apple with Siri [sir17]. These assistants

<table>
<thead>
<tr>
<th>Tool</th>
<th>Languages supported</th>
<th>Voice input supported</th>
<th>Integration</th>
<th>Slot completion</th>
<th>Price per request (Voice/Text)</th>
</tr>
</thead>
<tbody>
<tr>
<td>api.ai</td>
<td>14</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Free</td>
</tr>
<tr>
<td>wit.ai</td>
<td>49</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Free</td>
</tr>
<tr>
<td>Amazon Lex</td>
<td>1</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>$0.004 / $0.00075</td>
</tr>
</tbody>
</table>
Automotive data and intelligent systems

Table 2.3: Speech recognition tools comparison

<table>
<thead>
<tr>
<th>Tool</th>
<th>Languages supported</th>
<th>Context aware</th>
<th>Price per request</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google Cloud Speech</td>
<td>80</td>
<td>Yes</td>
<td>Free (up to 60 minutes) / $0.006 per 15 seconds</td>
</tr>
<tr>
<td>wit.ai</td>
<td>49</td>
<td>Yes (if using in conjunction with the bot service)</td>
<td>Free</td>
</tr>
<tr>
<td>Microsoft Bing API</td>
<td>5</td>
<td>No</td>
<td>$3.775 per 250 minutes</td>
</tr>
<tr>
<td>IBM Watson</td>
<td>9</td>
<td>No</td>
<td>Free (up to 1000 minutes) / $0.02 per minute</td>
</tr>
</tbody>
</table>

aim to be the perfect bot, by providing the user with answers to trivia by searching in the respective search engines, manage and detect user activity patterns while trying to engage in a conversation.

There are also some bad examples, like Tay from Microsoft. It was launched in early 2016 in the social network Twitter, and wanted to be the first of its kind, a fully self-learning bot. But just after some hours it became an extremist and racist bot \[tay17\], and had to be shut down.

2.5 Detecting patterns in time-series

One way to learn user behavior is to detect regular patterns that occur over a fixed timespan by analyzing existing data. Detecting patterns in time-series can be useful in a wide range of scenarios. For instance, to detect everyday user activities like detecting on which days a person goes to the laundry, or to predict CO2 ratios in air by taking into account previous measurements over long periods of time.

To be able to make these predictions, first the raw data must be processed to build an appropriate time-series. \[CVLC00\] describes the process of preparing and pre-processing the raw data, in this case medical data gathered from Isokinetic tests that analyze muscular fitness of patients. To detect patterns two options were explored. The first one uses exact pattern matching that tries to match all possible patterns in the data with known patterns stored in a database. The second one uses similar pattern matching using Euclidean distance, an algorithm similar as the one before. This approach is also explored in \[BC94\] by using Dynamic Type Warping. This approach consists on trying to match and align templates with the time-series so that some distance measure is minimized.

Another approach is explored in \[XSC12\], where this time genetic algorithms are used and tested against eleven groups of patterns with increasing difficulties. This method turned out to be very effective with accuracies over 90% in all tests.
A different approach from the ones stated above, is to use statistical methods to perform time-series analysis.

This can be done using a wide variety of manners. Curve fitting is one of them, where a function is constructed, using interpolation for example, to better fit a series of points. Related to this, there is statistical inference where models that better fit the data are created. Regression analysis is a well known method to perform statistical inference.

Within regression analysis, the most well known and relatively easy to apply method is Linear regression. Linear regression, as pictured on Figure 2.7, outputs continuous results, allowing to predict the value of the variable on a certain time. An easy example to understand this is to imagine a situation where variable X contains the date and variable Y contains price of gold. With linear regression, you can predict the price of gold given a date. As we can see, this fitted model, although it captures the ascending trend, does not fit the data very well. This can be improved using more complex models such as ARIMA (Autoregressive Integrated Moving Average).

ARIMA, as described on [TR01], is an attractive technique because it can capture complex stationary (observations fluctuating around a fixed mean), non-stationary (increasing or decreasing) and seasonal patterns. ARIMA also allows forecasting, both on short and long term, from one to an arbitrary amount of observations ahead. In [TR01], this technique is used to model activity on I/O devices, but to more clearly demonstrate and compare this technique, Figure 2.8 shows the ARIMA model applied to the same example as before (price of gold over time). As we can see, using One-Step-Ahead forecasts (forecast the next observation from each point), this model adapts much better than the one fitted using Linear Regression, better capturing the ascending trend although no easily observed seasonal pattern can be detected. Using data that contains seasonal patterns, like CO2 measurements over time, and applying an ARIMA model to it, represented on Figure 2.9, the model can capture both the overall uprising trend as well as the monthly ups and downs, as seen from the low forecast interval (areas highlighted in dark gray in the figures) making this technique really good to work with data like this.

An ARIMA model is composed of three different parts [AMMG14, Section 3.4]: autoregressive (AR), integrated (I) and moving average (MA). The autoregressive part AR(p) describes the correlation between the current value being analyzed in the time series and past values, where p is the order of the AR model, that is, the number of lags (past values) included in the model prediction. The integrated I(d) part represents the nonstationarity of the time series, if whether or not the data contains a trend, and d is the differentiation order needed to make the time series stationary. Finally, the moving average MA(q) considers that the time series is an unevenly weighted moving average of random errors, meaning that the deviation of the current value is equal to a weighed sum of the deviations of q past elements. An ARIMA model can also be defined to take in consideration data seasonality, in other words, the repetition of certain patterns in data over a fixed time period. To add this to a regular ARIMA model, an extra component needs to be added where the three parameters explained above also need to be defined, with the addition of an extra parameter s that represents the season length. For instance if s = 12, then we’re saying that a pattern repeats 12 times in a year, or every month.
Automotive data and intelligent systems

Given this, an ARIMA model is defined as \( ARIMA(p, d, q) \). If seasonality is added then the model is represented as \( ARIMA(p, d, q) \times (P, D, Q)_s \).

2.6 Summary

In this chapter, several concepts as well as related work were introduced within the context of automotive data collection and the development of bots using Natural Language Processing, so that the theme of this dissertation is clearly understood.

Regarding automotive data and its collection, we can conclude that there is a large amount of data that can be retrieved from vehicles coming from its sensors and location. This data can be obtained using different devices, both having their pros and cons, a clash between the dependency on third-parties and availability of data. Although there are some scientific projects and some industrial applications of this data, there are not many solutions available to the end-user.

As for Natural Language Processing, the current state of computer hardware made it possible to build and train more complex models using neural networks, providing better results when it comes to understand text and extracting intentions from it. Taking advantage of this technology are services that allow the creation of chatbots, providing the developers with an easy to use interface and a machine learning back-end making it possible to develop bots with ease.

To analyze user behavior, by taking into account actions performed over time, a wide variety of methods already exist, from exact pattern matching to statistical models, each adapting better than the others depending on the type and arrangement of the data provided. This makes it possible to integrate suggestion systems in the bot by forecasting user actions, that way making the bot connect and adapt with the user.
Figure 2.8: ARIMA One-Step-Ahead Forecast example using a gold prices over time dataset [go17].

Figure 2.9: ARIMA Seasonality forecast example using the Mauna Loa Weekly Atmospheric CO2 Data [co217].
In Chapter 2, the state of the art on automotive data retrieving and building intelligent systems with natural language processing was analyzed. Given this, it is possible to conclude that cars are able to produce enormous amounts of data from their internal sensors; although cars already come with integrated 4G routers and manufacturers have access to that data, there are not a lot of solutions that provide that data to the vehicle owner – the ones that exist are proprietary and do not provide much utility to the user.

With this in mind, a chatbot alternative will be presented, with the objective of providing the user with an easy to use chat interface, integrated in the most popular messaging applications, to serve not only as a mean to access vehicle data, but also to serve as a concierge\textsuperscript{1} service. To do this, not only will the bot support basic information querying (e.g. “What is the level of fuel?”) and location-aware operations, (e.g. “Show me nearby restaurants”), but also pro-actively notify the user when something goes wrong with the vehicle and learn user tastes and patterns, so that it is possible to provide the user with relevant information in each case. An example of this is that if the user makes the same trip every morning, then if the vehicle does not have enough fuel to complete that trip, it will warn the user to refuel first. One other interesting functionality is the capability to communicate with other bots in the WIT Bot Platform: for instance, when the user gets close to home, the automotive bot communicates with the home bot to open the garage gates.

The decision of developing a chatbot instead of a smartphone app is based on the fact that the rate on which users are actively searching for new applications in the respective platform store is declining, the majority of installations coming from recommendation. Furthermore, messaging applications are the most used amongst smartphone users. This is where chatbots shine, with the recent evolutions in NLP and the wide spectrum of tools to build bots (some of which are described on Section 2.4.1), and their capability to integrate with messaging applications. Bots do not require the user to install a separate application, providing content in a way users are used to

\textsuperscript{1}Special, personal, and attentive service
when talking with other persons, avoiding the need to learn a new interface and navigate through complicated menus to complete simple tasks.

### 3.1 Architecture

Figure 3.1 represents the basic architecture of the proposed platform. The bot will be integrated in the WIT Bot Platform, a framework currently being built by WIT Software that provides an easy way to integrate bots. The framework allows quick integration with several messaging applications like Facebook Messenger or Slack by abstracting this integration into one single API, that receives OBD information from the car, processes user intents through a NLP system and provides answers to the user.

Figure 3.2 shows a more detailed diagram of the data flow and technologies used. When the user says something to the bot, the voice is transcribed using Google Speech Recognition so that the app can give visual feedback to the user. The transcript is then sent to the platform that calls Amazon Lex. which sends back the intent to the platform through an Amazon Lambda function. The intent is then processed, with a possible use of Google Places APIs; after this process is done, the result is then sent back to the app using web-sockets and presented to the user.

#### 3.1.1 Fetching the vehicle’s data

To fetch a vehicle’s data, the device that will be used is Vinli OBD-II dongle. As mentioned in Section 2.1.2.2, it provides data availability anywhere through a REST API and does not depend on buying other devices. Because it also comes with an integrated GPS sensor, it fulfills all the bot’s data needs to get information about the car and get location-aware notifications using custom events.
3.1.2 NLP platform

The chosen NLP platform is Amazon Lex. Between using NLP parsers like Google SyntaxNet or ready-to-go services, the choice was simple because using NLP parsers would mean having to spend a lot of time building code to analyze the parsing results and understanding the user intention.

Between wit.ai, api.ai and Amazon Lex, the deciding factors were the easy integration with other Amazon Web Services, a high quality speech recognition providing no need for an intermediate speech-to-text parser, and having the best context-aware conversations, meaning that if the user does not provide any of the slots for an intent (explained in Section 2.4), it will automatically ask the user for them.

3.1.3 Providing functionality when driving

Because using the phone while driving is illegal and would cause the driver’s security to be in danger, when the user is driving and for demonstration purposes, the bot will be enabled using an Android application connected to the vehicle’s infotainment system using MirrorLink [mir17]. This will allow the user to avoid the need for typing text messages. For this reason, the bot will communicate through voice messages only, minimizing user distraction by not having to look to the screen for a long time.

Communicating through voice raises some challenges on how to direct the flow on an appropriate way. Because the user is talking and not typing, the bot must not have very complex flows because the tendency is that when the user is typing to a text chatbot, auxiliary inputs like
Chatbot for automotive integration

quick-replies aid the user through the conversational flow and because of this, flows can be more complex; on the other hand, when the user is communicating thorough voice, with no auxiliary input methods, the flow is harder to be conducted.

Taking this into consideration the automotive bot will minimize user input, by having quick and concise answers and not asking too much questions, but instead, let the user take initiative when initiating or continuing a flow, and focus on giving proactive and relevant information.

3.2 WIT Bot Platform

WIT Bot Platform is, as the name indicates, a platform that provides an ecosystem where bots can live and interact with each other as well as external systems. To do this, the platform can be divided in three different and independent sections that provide all this functionality and that together allow to build bots in a simple manner.

3.2.1 Engine

The engine is the core of this platform. It allows for the creation of bots, either from scratch or using pre-built templates for customer service, enterprise productivity and retail bots. By being integrated in this platform, all bots have access to a set of extra functionalities other than basic conversation building. These extras are the following:

- **Conversation Engagement**
  Pre-built 10k sentences to allow chitchat engagement conversation, with the bot’s most frequent asked questions (Bot profile, Emotions, Greetings, Jokes, Tastes, Smiles reaction);

- **Multi Conversation Sets**
  Configure and deploy multi bot personalities, in several languages;

- **Pre-built Telco domain specific dictionary**
  Allows advanced natural language processing thanks to a set of synonyms to telco dictionary;

- **Emotional Response**
  Sentiment analysis calculated in each interaction, to enrich conversation and emotional response;

- **Steer back**
  Actionable behavior that brings back the user when conversation starts to go out-of-flow;
Chatbot for automotive integration

- **Transactional Conversations**
  Make your conversational flow transactional, and force a commit or rollback before user changes subject;

- **Fall back Strategy**
  Allows cascade search for a good answer when trigger activation fails.

This platform can be described as a digital brain that, as the Human brain, has 2 sections, in this case, representing how bots “think”. One side of the brain is dedicated to Natural Language Processing/Understanding, to correctly identify the intent of the user; the other is dedicated to Machine Learning, that is, learn what the user does with the bot, how those actions are performed and with that, provide the user with a better experience.

### 3.2.1.1 NLP Integration

NLP integration is a crucial part of the platform; without it, the user could only talk with the bot using predefined sentences. This integration is made by processing user messages and sending them to an external supported NLP service (Wit.ai, api.ai and in the case of this dissertation, Amazon Lex). The service then returns the intent, as mentioned in Section 2.4.1, or an error message if it does not understand.

If the NLP service could understand the user intent, then the platform is ready to start the appropriate flow for that purpose, as detailed in the following section. Contrarily, if an intent could not be extracted from the user message, the platform falls back to the **Conversation Engagement** functionality, to provide the user with a friendly answer that would make the bot look intelligent and not immediately ask for clarification; in fact, this only done as a last resort.

To do this **Conversation Engagement**, the platform has a parser that searches a set of AIML files for an appropriate answer. Listing 3.1 shows an example of content that could be present in AIML file, that basically contains categories that define a pattern and a template. When a user sentence is captured by the pattern, meaning that it matched the expression, the bot will answer with the content defined in the template tag. These expressions can be used for exact string matching, as the first category in the example listing shows, or similarly to regular expressions, using wildcards to also match arbitrary words or characters (in the second category we can see an example of that, where in front of “infant” could be any word, represented by the wildcard *).

Other than a fall back method, these AIML files can also serve as a way to replace the NLP service and build simpler bots by prepending the string with an intent prefix on the template, instead of specifying a simple raw text string like in the example. This way, the platform will recognize that it has to start a certain flow.
3.2.1.2 Flows

Flows are where the logic of the bot resides. A flow processes the current intent by connecting with possible external services and dealing with possible user follow-up. Flows are defined using BPMN files with custom WIT parameters that allow for a graphical representation of logic processes. An example flow is shown in Figure 3.3.

Here we list all the different nodes that are able to be used within WIT Bot Platform, those being (left to right in the example):

- **Start Node**
  This is the flow starting point and is activated when the intent of the user is equal to the Engaging pattern property of this node. At this point, the system keeps track that the user is currently in a flow, and the Steer back system will return to this flow automatically;

- **Service Task**
  This kind of node is where all the data management and processing, connection to databases or third party APIs is made. This data is then passed on to the following nodes in the flow;

- **User Task**
  Task where the user is prompted to select an option or the flow waits for user input to continue;

- **Conditional Node**
  This kind of node allows to fork the flow, making possible within the flow to perform different actions based on the set value of a variable;

- **Message Task**
  Task where the system can send messages to the user. The type of messages currently supported are text messages and other UI auxiliaries like carousels that show a list of cards, a
Chatbot for automotive integration

Figure 3.3: Example BPMN flow

single card that contain an descriptive image, text and a button to easily access a bot functionality, and quick-replies;

- **End Node**
  This is where the flow ends, clearing the user from the active flow that was set in the start node.

After this flow is built using a BPMN editor, the next step is to export both the BPMN and XPDL file and pass them as arguments of an existing parser in the platform. This parser will interpret these files and generate a JavaScript file where it is possible to write custom code for the nodes described above, and therefore, providing logic to the bot.

### 3.2.2 Gateways

Currently WIT Bot Platform provides support for messaging on various different channels like Skype, Slack or Facebook Messenger through an abstraction layer, meaning that in the flows, there is no need to know what service the user is on – the gateway automatically formats the message to the format needed by the said service.

This support is done not only with third party services, but also directly with carrier networks, meaning that the user could access some of the bots developed via SMS or applications using RCS.

For this dissertation, these standard channels will not be used, but instead, a new one will be created to communicate with a MirrorLink-enabled Android application.

### 3.2.3 Human Assistance

Particularly in costumer assistance bots (but no only), there is the need to take into consideration the costumer state of mind, and that can be a weak point of the bot. To face this, the platform has a web monitoring station, where human employees can monitor the current as well as past interactions from the bot. Here a real-time sentiment analysis is kept track of, and if it falls below a certain point where the person can be considered angry, a request is made to the human assistant to take charge of the conversation to try and ease the situation.
3.3 Adapting to the user

So that the bot can adapt to the user and be perceived as more useful than just a chatbot, by also being an assistant that helps the user by providing proactive suggestions related to their habits, the platform must implement a pattern recognition module.

To integrate this module on the platform the choice was to write a Python script that is scheduled to run at opportune times. This script implements and fits an ARIMA statistical model described in Section 2.5 to driver data. To prove this concept while providing useful information to the user, the data that will be used is travel data, more specifically the mileage that each user travels on each of their trips.

This information is gathered from the OBD dongle whenever the user finishes a trip: the device triggers an event and sends data to the server, and that data is stored in a CSV file containing the date of the trip and the distance traveled. An example of possible data is shown in Listing 3.2. The script analyzes user weekly patterns assuming users usually repeat the same travel pattern each week, and therefore the model will be able to produce more accurate predictions. After the script is run, it returns the most likely mileage that the user will travel in the next day and with that, the server can cross information to provide useful suggestions to the user.

3.4 Summary and conclusions

The state of the art that was explored in the last chapter was analyzed and a problem was found, the lack of applications that connect the users to their cars, despite the large amount of data available.

With this in mind a solution was proposed, describing its architecture, technologies to use and the importance of NLP in building bots over a simple application to perform the objectives described. This solution is integrated in the bot platform currently being developed by WIT Software, providing the perfect environment for the development of this bot. This integration is useful
Chatbot for automotive integration

to the development of the bot, but also for the platform itself that is expanded with new functional-
ities and connection with external services. How this platform works and how it will be extended
is also detailed in this chapter.

With this solution, users can have a service valuable to them available both while driving or
outside the car, in an easy and accessible way.
Chatbot for automotive integration
Chapter 4

Implementation

For facing the challenges of interacting with the user while driving and still providing useful information, a solution was designed on Chapter 3. Following this architecture, a prototype was developed in order to test and validate the feasibility of this solution, as well as to identify possible improvements.

In this chapter, the prototype is presented, both the server-side (WIT Bot Platform) and the mobile application, while providing a deep explanation on essential topics such as how the communication is made between the two components, how NLP is used and how to perform user pattern recognition.

4.1 Server - WIT Bot Platform

As described in Section 3.2, the centralized server of the bot is integrated in the bot platform being developed by WIT Software. Here, all user requests are processed, their data is managed and the pattern recognition algorithm is run.

This component is essential: without it, the development would have to be made entirely on the mobile application, rendering the process a lot more complicated, specially concerning the pattern recognition component.

The platform provides both REST endpoints to allow requests from external services like Amazon Lex and the OBD device, and a Websocket connection to be permanently connected when the application is activated. More detail on this in the following sections.

4.1.1 Technologies

The platform is implemented using state of the art web technologies detailed below:

- Node.js

  Node.js [nod17b] is an event-driven, non-blocking I/O JavaScript runtime that allows the
creation of easily scalable web apps. It also has its own package manager, npm, that allows users to quickly import code from already existing libraries

- **Express.js**
  To ease the creation of an HTTP server within Node.js, there is Express.js [exp17]. This package is a web application framework that provides a robust set of features for web and mobile applications, including a large set of HTTP APIs.

- **python-shell**
  *python-shell* [pyt17] is a library that allows to run Python scripts in Node.js. It provides simple yet effective inter-process communication and error handling. This package is useful to run the pattern recognition script that is written in Python.

- **node-schedule**
  *node-schedule* [nod17a] is an utility package that allows to schedule jobs to run at a certain time. While native Node.js scheduling is interval-based (run every X milliseconds), *node-schedule* is time-based (run every first day of the month) allowing for more flexible scheduling. This is used to schedule the pattern recognition job every morning.

- **websocket**
  *websocket* [web17a] is a package that implements WebSocket protocol versions 8 and 13 for Node.js. It provides high-level easy to user APIs for the creation of WebSocket servers or clients. This is used to establish a permanent connection between the platform and the Android application.

- **MongoDB**
  MongoDB [mon17] is an open-source NoSQL database, meaning that data is not stored in tables with classic tabular relationships, but instead in flexible, JSON-like documents, meaning that fields can vary from document to document and data structure can be changed over time. This is the main advantage that Mongo brings to this project, the ability to develop flexible data models in order to adapt to any situation.

Because user account linking the pattern recognition functionalities are done in different modules and do not require a database connection, the data model for the platform is very simple. It only contains one collection - *Users* - and each user object stores only its ID. When an intent is being processed, a temporary payload object is also stored in the database. The user ID is an ID that the client application sends to the server at the moment of authentication that identifies the user with an account linking platform that will be further explained in Section 4.2.2.1.
4.1.2 Platform integration

4.1.2.1 NLP integration

Amazon Lex, the NLP parser that is used in this project, provides a web interface as well as an API to define the bot and its intents. These intents can have slots that can be defined with one of the already available types (AMAZON.NUMBER for instance), or using custom slot types.

To train the bot into recognizing intents or custom slot types, sample utterances need to be defined. These utterances are sentences that Lex uses as reference to detect the intent/slot, not just by exact pattern matching but also using a backend deep neural network to find similar matches.

In this implementation 4 intents were defined: *FuelIntent*, *HungryIntent*, *NextResults* and *LocationIntent*. These do not have any slots that need to be filled, again reinforcing the idea of simple interactions while using voice conversations.

After Amazon Lex successfully fulfills the intent (fills all the slots, if any), it provides 2 options: send a default message back to the user, or run custom code through an Amazon Lambda function. Because we want to run our own flow, based on the user intent, a Lambda function is used that just makes a POST request to the server with the result of the Lex intent fulfillment.

4.1.2.2 OBD connection

The OBD connection is made through the Vinli service SDK, that allows to system to connect to a device. When a user is registered, the connection is established and a reference to the client object is saved on the database. Establishing this connection is only the first step; then, the platform registers the device to receive notifications to all the needed events, these being:

- *trip-started*
- *trip-stopped*
- *dtc-on*
- *dtc-off*

This subscription is essential, since the events provide crucial data to important functionalities of the bot, *dtc-on* and *dtc-off* to proactively warn the user when car trouble codes appear, and *trip-started* and *trip-stopped* to start and feed the pattern recognition algorithm.

Other than these subscriptions, the SDK allows direct access to live data of the car, through a method that accepts a valid PID and returns its value.

4.1.2.3 API

To allow connection from essential external services, the platform needs to implement new REST endpoints to process and receive data from those external sources. To do this, a new subsection
Implementation

where endpoints are defined was added to the platform. Here, the Express server that the platform initializes is extended and listens in the following routes:

- **POST /Lex**
  This route is where the server receives requests from Amazon Lex through a Lambda function, as detailed on the Section 4.1.2.1. When a request is received, the platform gets the intent name and session attributes received from Lex and prepends the name with the string “INTENT:” - necessary to be detected by a flow’s engaging pattern - and then starts the appropriate flow;

- **POST /Vinli/notifications**
  This route receives requests from the OBD services that contain information of the event that was subscribed (more information in the previous section), and takes the adequate actions: either warn the user about a problem with the car in case of a `dtc-on` event, or perform the adequate pattern recognition actions in case of `trip-*` events.

### 4.1.2.4 WebSocket connection

To enable communication between the platform and the mobile application, a WebSocket connection is open. WebSocket [web17b] is a communication protocol providing communication between channels over a single TCP connection. It allows to send and receive messages in an event-driven fashion without the client needing to poll the server for responses.

As mentioned in Section 4.1.1, the package `websocket` was used to implement this protocol in Node.js. On the platform, a WebSocket server is created tied to the Express HTTP server, so that a connection could be made to the same URL as the HTTP server, but using the WebSocket protocol. Listing 4.1 shows how this is done, and also the process of accepting or rejecting a connection. The connections are accepted if the origin is recognized as not being one from a web-browser (should only accept connections from mobile devices), and the client sends the correct protocol key to the server. This key is a custom string that serves as a way to identify the client to the server; in this case, we only accept connections with the “automotive-protocol” key.

After the connection is established, the server can receive messages with two kinds of data, either string or binary data. To easily separate the registration and authentication logic from all the other messages, the chosen approach was to send the authentication data as a JSON string, and all the other messages as binary JSON data.

### 4.1.2.5 Flows

As mentioned before in Section 3.2.1.2, the logic of the bots developed in this platform is implemented using flows. To provide the desired functionality and some example use-cases to this bot, the flow represented in Figure 4.1 was implemented.

It has 4 entry points:
Implementation

```javascript
wsServer = new WebSocketServer({
  httpServer: server,
  autoAcceptConnections: false
});

function originIsAllowed(origin) {
  ...
}

wsServer.on('request', function(request) {
  if (!originIsAllowed(request.origin)) {
    // Make sure we only accept requests from an allowed origin
    request.reject();
    console.log((new Date()) + ' Connection from origin ' + request.origin + ' rejected.);
    return;
  }
  var connection = request.accept('automotive-protocol', request.origin);
  ...
}
```

Listing 4.1: WebSocket server creation and connection filtering

- **PlaceSearch**, that captures when the user wants to search for a place nearby, that is, when the NPL service in this case returns the user intent as being either “HungryIntent” or “FuelIntent”;

- **Filter**, that captures when the user wants to filter the results returned by a previous run of the flow when captured by the **PlaceSearch** start node;

- **Next**, that captures when a user wants to navigate to the next result in the set previously returned; this node is activated when the NLP service’s response identifies the intent as being “NextResults”;

- **LocationIntent**, that captures when the user wants to navigate to a certain place, and is activated when the NLP service returns the intent as being “LocationIntent”.

When the **PlaceSearch** start node is activated, the next task in the flow is a Service Task named “FindPlaces”, that communicates with the Google Places APIs with custom parameters depending on what kind of places the user wants to get (restaurants or gas stations in the implemented use-cases) and returns the first 5 most relevant results. This task also builds a carousel payload that will be sent to the user application in a later stage, and saves a copy to the user object in the database. The next step is made by another Service Task (“ProcessPlaces”) that sees what is at the time the first element in the user places payload, builds a custom message in the format “You (also) have {place.name} nearby.”, and removes that element from the array. This answer, along with the intent and the payload, is then sent in the “SendResponse” send task. After this, the flow ends.
If either the *Filter* or the *Next* starts nodes are activated, the “ProcessCurrentIntent” service task checks to see if the user already interacted with the bot and if the last intent that was processed was the *PlaceSearch* intent. If it was, then the flow advances to the “ProcessPlaces”, whose logic was described earlier; in the case of the *Filter* intent, this task also filters the payload array to contain only the places that meet the required parameter (i.e. Italian restaurants). If the user’s previous intent was not a *PlaceSearch* intent, then the platform sends a message saying that it could not process that action because no context is available.

Finally, when the *LocationIntent* start node is activated, the same process of checking the intent is made. In this case, if the intent was the *PlaceSearch* intent, the service task “ProcessCurrentPlace” is executed and creates a response in the format “*{place.name} is near {place.vicinity}.*” that is sent to the user. Vicinity is a field returned from the Google Places API that represents a human-readable string of a location near the place.

Both the *PlaceSearch* and the *LocationIntent* intents need to access the device location. To do this, two option were explored: using the OBD device GPS location, and using the device’s internal GPS, both oh these having their pros and cons. By using the OBD device location, the user can save battery life by not needing to have the smartphone location active but these location updates are not so frequent (once every 5 seconds [vin17]). On the other hand, by activating precise location on the smartphone settings, and consequently improving the battery consumption, the application is able to receive more precise location updates (using both the GPS sensor and network connectivity to infer location) and only receive them when the device actually changed its position and therefore save some process time of the application. With this in mind, the chosen approach is to use the smartphone location in order to get more accurate results and with this allowing the user to still be able to make use of the functionalities that use location, both inside and outside the vehicle.
4.1.2.6 Pattern Recognition

To perform the pattern recognition, data must be fed into the system so that the algorithm can fit a model to that data. This storage of data is done when events described in the previous sections are triggered, namely trip-stopped. When this event is triggered, the user stopped the trip that he was making, and an object with the trip details is sent to the registered endpoint mentioned in Section 4.1.2.3; with this information, the distance traveled can be fetched and then saved into a file. Given this structure, a CSV file is saved for each user where the file name is the user ID within the platform and the content is list of pairs (date, distance) in the format “{YYYY-mm-dd}, {Km}”. To avoid missing data in the files, at the end of each day, an entry is added to the file simulating a trip with 0Km.

When it comes to run the script, it is done using a timed task that is set at the moment of user registration, using the package node-schedule as referenced in Section 4.1.1 to schedule the task to run on a time-based manner instead of interval-based; in this case the task is set to run everyday at 9AM. Not only does the script run based on this scheduling, but also whenever the user starts a trip (the trip-started event is triggered). This is useful to not only provide suggestions every morning, but to check if the user has enough fuel to complete the remaining kilometers that were forecast.

This script implements the ARIMA statistical model. This model was chosen, because as seen in Section 2.5, it is able to better capture tendencies in time-series and because it can incorporate seasonality, it performs well with defined patterns that repeat constantly.

Because Python is a well known scripting language and provides better options for development using statistical models and machine learning with libraries that are widely supported and that are constantly being improved, the decision was to use Python for the pattern recognition in time series. One of these libraries and the one chosen for the implementation is statsmodels [sta17]; this library implements both the seasonal as well as the non-seasonal versions of the ARIMA model as well as many other, providing a lot of flexibility that other libraries don’t.

An ARIMA model needs to be fitted into the time-series data to be able to predict values in the future. This is done by adapting its parameters p, d, and q that respectively represent the order of the autoregressive model, the differencing degree and the order of the moving-average model. This can be done in a couple of ways. One is by manually studying the data and experimenting with the parameters and evaluating results with resulting graphs shown in Figure 4.2 (can be build using the statsmodels library) that show if the model residuals are normally distributed (good sign), follow a linear trend and present a seasonal trend (if true, it is a bad sign meaning that seasonal data is being lost). Another method is to exhaustively fit models using different parameters and compare the AIC (Akaike Information Criterion) of each. AIC measures the relative quality of statistical models and offers an estimation of the information that is lost using the model; because of that, a lower AIC generally means the model is best fitted to the data. ARIMA with seasonality also contains an extra three parameters: P, D and Q; these have the same meaning as p, d and q mentioned before but for the seasonality aspect of the model and their value can be found in the
same ways. With seasonality, an extra parameter is needed, that is the number of periods in each season and that must be set manually.

In this implementation, because data is not the same between all users and big differences in the patterns can exist, to determine the best parameters the latter approach shown in the previous paragraph is used and presented in Listing 4.2. First, a list of all possible combination of p, d and q parameters is made where these can take integer values from 0 up to 3; then, models using all the combinations are fitted and the one with the lowest AIC criterion is chosen to do the predictions, as shown in Listing 4.3. After this is done, the values from the predictions and confidence intervals are saved and passed back to the platform where the script was invoked, through stdout. These models all take the value of 7 regarding the seasonality, indicating that patterns occur in the data repeating every week. This was chosen because user patterns generally are almost the same every week, for instance a high number of kilometers traveled during week days maybe to go to work, and less on the weekends.

The function to implement the ARIMA with a seasonal aspect in the statsmodels library, as seen in Listings 4.2 and 4.3, takes as parameters the data, the list of parameters for the normal ARIMA model, the list of parameters that represent the seasonal aspect of the model and an extra parameter. enforce_stationarity is a boolean that indicates whether or not the data must be treated as stationary data. Stationary data means that the probability distribution does not change over time – patterns would always be the same, which is not completely true in this case since users can vary their patterns over time.

Figure 4.3 shows the algorithm applied to one month’s test data. As it is possible to notice, using the extensive search for the best possible parameters turned out to be successful as the patterns were correctly identified. Because the enforce_stationarity flag was set to false indicating that the data is not stationary, confidence intervals can also be gathered from the results and are represented as the dark gray area in the plot.

After the script is run, in the Node.js side, the forecast for the next day of the stored data is captured, meaning that each morning when the script runs the value returned from the script is the number of kilometers that are predicted to be traveled that day. With this, a request is made to the OBD service to get the values of the current fuel level and fuel consumption to calculate the autonomy of the car and compare it against the prediction. If the car does not have enough fuel, then the platform sends a push notification warning the user of so.

4.2 Mobile Application

The mobile Android application is the gateway to communicate with the user inside the car. Because the user is driving and would be illegal to use standard bot communications platforms such as Facebook Messenger, the chosen approach was to build an application that connects to the car infotainment system.

This application provides both a simple graphical interface presented on the dashboard, to which the user can look at and not be easily distracted, as well as voice communication, allowing
Implementation

```python
def evaluateModel(data):
    p = d = q = range(0, 3)

    # Generate all different combinations of p, q and q triplets
    pdq = list(itertools.product(p, d, q))

    # Generate all different combinations of seasonal p, q and q triplets
    seasonal_pdq = [(x[0], x[1], x[2], 7) for x in list(itertools.product(p, d, q))]

    lowest_aic = 1000000000
    lowest_param = None
    lowest_season = None

    for param in pdq:
        for param_seasonal in seasonal_pdq:
            try:
                warnings.filterwarnings("ignore")
                mod = sm.tsa.statespace.SARIMAX(data,
                                                  order=param,
                                                  seasonal_order=param_seasonal,
                                                  enforce_stationarity=False)
                results = mod.fit(disp=False)

                if results.aic < lowest_aic:
                    lowest_aic = results.aic
                    lowest_param = param
                    lowest_season = param_seasonal

            except:
                continue

    return lowest_param, lowest_season
```

Listing 4.2: ARIMA model evaluation

```python
def processData(data, order, season_order):
    mod = sm.tsa.statespace.SARIMAX(data,
                                     order=order,
                                     seasonal_order=season_order,
                                     enforce_stationarity=False)

    results = mod.fit(disp=False)

    # Get forecast 20 steps ahead in future
    pred_uc = results.get_forecast(steps=20)
    print(pred_uc.predicted_mean)

    # Get confidence intervals of forecasts
    pred_ci = pred_uc.conf_int()
```

Listing 4.3: ARIMA model fitting and forecasting
Implementation

Figure 4.2: ARIMA model fitting analysis obtained while fitting driver data from the prototype application

the user to talk with the bot by voice and listen to its responses without even taking the eyes off the road.

4.2.1 External libraries used

To make the process of building this application easier, some utility libraries were used, those being:

- Volley
  Volley [vol17] is an HTTP library that makes the process of creating and making requests to external APIs easy, without the need to manually manage the connection lifetime. It also provides custom callbacks when the request returned successfully or with any error;

- AndroidAsync
  AndroidAsync [and17] is a low-level network protocol library that contains both websocket client and server implementations, proving easy to implement callbacks for message exchanging making it simple in this case to establish a connection to the server;

- Picasso
  Picasso [pic17] is an image downloading and caching library that allows an easy way to dynamically load images into the interface, again without the need to manually create tasks to do the hard-work;

- Mapzen
  Mapzen [map17] is a mapping library, similar to Google Maps, that enables maps to be
Implementation

Figure 4.3: ARIMA results using real driver data obtained through the use of this application (dark areas in the plot represent the forecast confidence intervals)

loaded into an Android Activity. The main advantage of this library is that it provides free turn-by-turn navigation within the same map fragment and without the need of launching another activity.

4.2.2 Establishing connection to the server

The first and essential step to this application to work is to connect it to the server. Without this connection the application would be useless since it would not be able to access the OBD service to fetch car data, nor communicate with the platform to process user intents.

As mentioned in Section 4.1.2.4, the communication between the server and the application is made through websockets. To do this connection, as referenced in Section 4.2.1, the library AndroidAsync is used. This library’s implementation of a websocket client allows for an easy connection as the function’s signature only requires the URL of the websocket, the protocol (defined in Section 4.1.2.4) and a callback object whose onCompleted function is called when the connection is ready. In this function, a WebSocket object is able to be accessed and the message callbacks set. Just like the server, the client connection can also receive both string and binary messages. Every message sent from the server and received in the application is a string message.

4.2.2.1 Identifying user with server

When the connection is ready, the application sends the authentication data to the server as a JSON string. This JSON string contains, as shown in the example in Listing 4.4, the bot name that the application wants to authenticate to, the user ID, the Vinli device ID and the Firebase token that is associated with that mobile phone. Firebase [fir17] allows to add cloud services to applications; in this case Firebase’s cloud messaging service is used to receive push notifications from the server.
Implementation

```json
{  
  type: "auth",  
  botName: "AutomotiveBot",  
  userId: ...,  
  vinliId: ...,  
  firebaseToken: ...
}
```

Listing 4.4: Example authentication data

The user ID is an id generated from Amazon Cognito [cog17] that allows to link account logins from different platforms like Facebook or Google and associate them with a single user, that way the user can login with different platforms in different devices and still be recognized as the same person.

Only after the user is logged in, the app loads the necessary UI components for it to be fully used.

4.2.3 Connecting to the infotainment system

So that the application can be displayed, MirrorLink was used. MirrorLink is a free-to-use library that allows the creation of applications that can be enabled to be displayed in a car’s infotainment system. Other than just the display, MirrorLink allows the app to use other systems of the car, mainly, the audio system: instead of being played through the phone’s speakers, sound is output through the car’s speakers, allowing for a better experience. To add to this experience, then an application is being mirrored using MirrorLink, the user can press on the physical buttons that control the infotainment system (for instance those on a steering wheel) and the application will receive events as if keys on a keyboard were being pressed, this way allowing application interaction without having to use a touch screen.

4.2.4 Parsing server messages and display content

The mobile application receives from the server string messages in JSON format that can be of two kinds:

- Intent Messages
  When the application receives an Intent Message, it means that the bot is interacting with the user and the user interface must change to show the bot message and sound should be played with the same message. This kind of messages can be of several types, all making different kinds of interactions with the user and the UI. All these messages come with a `message` parameter that is the bot’s answer.

  -- Chitchat
  When the application receives a chitchat message, it means that the bot is unable to
Implementation

proactively continue with that Intent (not recognized by Amazon Lex) and is just responding with standard messages fetched from AIML files as referred in Section 3.2.1.1. In this case, no audio message is received, and voice must be synthesized. This is done using Amazon Polly [pol17] that receives a string a returns an MP3 file URL referencing the generated sound. While the sound is being played, a chat balloon is rendered with the text representation of the message. An example of this interaction is shown in Figure 4.4;

- Intent with payload
  These are the intents that present more information to the user. When this message is received, the application must play the sound that comes in it, but also render the payload into a carousel of cards displaying information about each item. An example of this is show in Figure 4.5;

- Next results
  This intent is only received when a carousel is active, meaning that the user intent that is being processed renders a carousel on the UI. When this intent is received, the application must slide the carousel by one position and play the received audio;

- Simple Messages
  Simple messages are just a sentence that the bot wants to play back to the user. Unlike chitchat, the message already comes with the audio file reference to be played. A chat balloon is also rendered in the UI;

- Location Intent
  This message is received when the user asks where a place is (contextualized within a carousel). The application sends the place location as well as the user current location to the Mapzen API to return a map with a drawn path between the two locations with the complement of turn-by-turn indications. This map is then render into a new application as shown in Figure 4.6.

- OBD Messages
  OBD messages represent the current state of the vehicle and can be sent proactively when a problem occurs or when the application made a previous request to telemetry messages, such as fuel level. If the message contains an object whose key is dtc-on, then a problem was found in the car and the bot must warn the user; this is done by incrementing the warning number in the toolbar as well as displaying a chat balloon. Just like chitchat, the application synthesizes voice. Telemetry messages just change the UI on the toolbar (fuel level and autonomy). Both these cases are shown in Figure 4.7.

4.2.5 Use cases

When using the application, the user can perform several actions:
Figure 4.4: Bot chitchat interaction

1. Click the microphone button and talk to the bot - the bot then answers using the process specified in the previous section;

2. Click the repeat button - this will play the sound of the previous bot answer and is done by always keeping a reference to the last message;

3. Click the warning button in the toolbar - this produces the same effect as the bot proactively warning the user as previously detailed and shown in Figure 4.7.

Additionally, and as detailed in Section 4.1.2.6, the application can receive push notifications. This is done when the platform executes the pattern recognition script that analyzes user past trips and mileage, and crosses the results with car information. An example of this is shown in Figure 4.8, where the bot recognized that the car did not have enough fuel to complete the distance the user usually travels at that day of the week.

4.3 Summary and conclusions

In this chapter, a detailed view on the implementation of the solution was presented, describing each component of the system and its usefulness and contribution to the end results. Not only this, but also component connection and communication was specified and detailed.

On the server-side of the prototype, the decision was made to use highly scalable and adaptive technologies like Node.js, facilitating both the development as well as user adaptation by using a noSQL database (MongoDB). Although facilitating the development on a high-level, some challenges arose, mainly on how to integrate the pattern recognition module that was built using a different programming language (Python). This challenge was quickly solved using a Node.js library that allowed communication with external Python scripts by reading stdout output.
Finally, on the client-side, standard native Android libraries were used to build the application, with the addition of external libraries to facilitate the websocket connection to the server, as well as library to build UI elements like card views and presenting them on a carousel.

Concluding, by having a well defined solution architecture (explained in Chapter 3), the implementation was much facilitated, allowing the creation of a prototype that fulfills all the goals of this dissertation.
Implementation

Figure 4.6: Application map activity

Figure 4.7: Application OBD warnings
Implementation

Figure 4.8: Example notification warning
Implementation
Chapter 5

Experiments and results

In order to check if the prototype is valid to be used in a real environment and be released in the market, it must be tested. This was done using three methods, first a questionnaire to validate the possible product adoption and usefulness; then some tests were made in order to validate the pattern recognition results and take conclusions on how to improve it; at last, after the prototype was completed internal usability tests were performed with WIT Software personnel to validate the bot interaction and the application’s usability.

In this chapter, the methods enumerated above will be presented more deeply with data visualization of the results.

5.1 Questionnaire

The questionnaire consisted on a series of 9 questions presented in Appendix A and was answered by 162 persons. In this section there will be, for each question asked in the questionnaire, an explanation for why that question was asked and evaluate the results.

Question 1 was made to evaluate the age group of the survey respondents. This is useful to understand if the group was concentrated on a younger (more receptive to new technologies) or older (less receptive) side. As we can see from the results in Figure B.1, the vast majority of the respondents (~94.5%) had between 18 and 30 years of age, an age group that is generally considered as receptive to new technologies.

The purpose of questions 2 and 3 was to see if those inquired already had any kind of experience using chatbots before and how they compared them to applications that only interact using a graphical user interface. From Figures B.2 and B.3, we can see that the majority (63.6%) had already interacted with at least one chatbot before, and around 98.4% think that chatbots are advantageous over GUI interfaces, with 56.2% considering that the advantage is very big (classifications 4 and 5), confirming what was said about the rise of bots in Section 1.2.
Experiments and results

Questions 4 and 5 were made to evaluate user satisfaction using conventional automotive data providers like modern dashboards or any other mobile application. From the results pictured on Figures B.4 and B.5, we can see that although there was a general satisfaction regarding the use of these tools, only 41.4% knew they existed.

Questions 6, 7 and 8 serve to see if there was need and interest of using this product. Figures B.6, B.7 and B.8, represent the result to these questions. Although the question of “Would you use this?” is kind of an ice-cream question, meaning that people could be inclined to answer yes about the use of new products, we can see there is the need to gather some kind of contextual information while driving, the area that this bot falls in, and most people would use it and most even consider paying for this service.

Question 9 evaluated how people considered the usefulness of the adaptive part of the bot, how useful would a pattern recognition system that gives proactive suggestions be. As we can see from Figure B.9, most persons found it really useful and a good complement to the system.

Finally, we can conclude that most of the target audience does not know or does not have easy access to automotive data, and finds that a bot that gives contextualized information while driving and proactive suggestions based on past events would be very useful, validating the purpose of this bot.

5.2 Pattern Recognition experiments

To see if the pattern recognition algorithm provides a good fit for driver data, some experiments were made. Table 5.1 shows the comparison of the algorithm’s execution (average of 100 executions) with different amounts of data - low for just a week’s worth of data, medium for around 2 weeks of data, and high for one month -, different limit for the model parameters (as explained in Section 4.1.2.6) and the resulting execution time and AIC. These tests were made using the same data for comparison within the same level of data amount. Instead of just forecasting in the future like (how the prototype works), in these tests the algorithm was set to start the forecast on a set date previous to the last known record.

Beginning with a low amount of data, we can see from Table 5.1 and Figure 5.1 that both variances performed with the same result (equal AIC) and with very little difference in the execution time. Because the amount of data is low, neither variances were able to capture the exact pattern, although some tendencies began to be captured but with a large confidence interval. Adding an extra week to the data, using limit 2 for the parameters, the algorithm produced the same visual results but with a slight increase on the AIC; using 3 as limit, performed much better, perfectly capturing the pattern as seen in Figure 5.2 and a much lower AIC. Increasing the amount of data to a month proved to help, as both variances were able to capture the pattern as shown in Figure 4.3, but with a large difference in the value of AIC between the two.

We can conclude that increasing the limit of the values parameters can take and therefore increase the model complexity and execution time can be useful, specially when working with low amounts of data, because it can result in models that better fit the data and produce better results.
Experiments and results

Table 5.1: Pattern Recognition algorithm variances comparison

<table>
<thead>
<tr>
<th>Amount of data</th>
<th>Parameter limit</th>
<th>Execution time (s)</th>
<th>AIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>2</td>
<td>1.16830</td>
<td>3.1287</td>
</tr>
<tr>
<td>Low</td>
<td>3</td>
<td>2.06672</td>
<td>3.1287</td>
</tr>
<tr>
<td>Medium</td>
<td>2</td>
<td>1.81751</td>
<td>-0.5688</td>
</tr>
<tr>
<td>Medium</td>
<td>3</td>
<td>5.29922</td>
<td>-233.5811</td>
</tr>
<tr>
<td>High</td>
<td>2</td>
<td>5.47359</td>
<td>-214.5624</td>
</tr>
<tr>
<td>High</td>
<td>3</td>
<td>60.52558</td>
<td>-679.5823</td>
</tr>
</tbody>
</table>

Contrarily, with large amount of data, if the application need to execute the algorithm in situations where execution time is critical, a better option is to use a lower parameter limit, decreasing both the overall number of parameter combinations and execution time, with the implication of possible worse results.

5.3 Usability tests

After the initial questionnaire and some experiments with the pattern recognition algorithm, usability tests (on a simulated environment) were performed to validate the bot’s use cases inside a car. These tests were performed internally in WIT Software with 5 volunteers, all of which drive every day to work and do not use chatbots regularly.

Table 5.2 shows the results of these tests, where the beginning of the interaction with the bot, the bot’s ability to provide both contextualized and out of context information, and the pattern recognition are classified on a scale from 1 (very bad) to 5 (very good).

Analyzing the results we can see that regarding the beginning of the interaction with the bot, most testers found it really easy to use, mentioning the fact that being able to activate the voice recognition with a click on a physical button inside the car was a big plus. When asked to test the bot by asking for information that needed to use user context (like location or car info), the testers were pleased with both the quality and the presentation of the answer. The bot did not perform so well with out of context information because the AIML default answers database is not big enough to cover a large amount of possibilities.

When the users were explained about the process and how the proactive suggestions are sent to the user, they found it appropriate but had some suggestions, such as to consider not only the total number of kilometers traveled during a day, but also the hour that the user makes the trips and push the notification based on that; another suggestion was to extend this system to automatically adapt the car’s comfort system to user preferences, like the air conditioning to the temperature that the user usually turns it on to.

Concluding, with these usability tests the bot’s usability inside the car was validated with a positive feedback from the test subjects. Also some feedback to possible future work was gathered.
Experiments and results

Figure 5.1: ARIMA execution for low amount of data

Figure 5.2: ARIMA execution for medium amount of data (using 3 as parameter limit)

Table 5.2: Usability tests results

<table>
<thead>
<tr>
<th>Tester</th>
<th>Beginning of interaction</th>
<th>Contextualized Information</th>
<th>Out of context information</th>
<th>Pattern recognition</th>
<th>Overall classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tester #1</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Tester #2</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Tester #3</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Tester #4</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Tester #5</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>
Chapter 6

Conclusions and Future work

The main objective of this dissertation, as said in Section 1.2, was to implement a smart chatbot that understands user intents and extends the relationship with their cars. With this in mind, a research of the state of the art of automotive data and chatbot implementations was made and the problem was identified: applications that use automotive data in an accessible way to the user are almost none and that provides a good opportunity to develop a solution of this kind with the help of NLP to communicate with the user using a new trend that are bots, and that can be relevant in the market.

Also in Section 1.2, some challenges on how to develop a solution like this arose:

- Efficient communication with the user

  When using a chatbot, communication is critical, and this point was explored in Section 3.1.3, where the fact that the bot must have simple flows to interact with the user was pointed out. This was even more evident during the development phase while developing the use cases. Given the fact that computer generated voices don’t yet have the same tone as human voices, long interactions caused difficulties mainly because of the lack of human likeness. This was tackled by using small and informative flows to keep the user interested in the application.

- User security

  One other challenge was keep user data secure. The chosen approach proved to be very helpful since the OBD device used was integrated in a third-party platform that provides an abstraction layer with a REST API that made data management relatively easy. One other aspect regarding user security that was addressed in this dissertation, is user access to the bot while driving - how to begin interacting with the bot. Looking back, the chosen approach was able to get the job done perfectly, but if other frameworks like Android Auto could provide better integration with car systems (like audio for instance), it would be better. This is mainly because of MirrorLink’s very strict rules regarding application user interface, that created delays while developing the prototype.
Conclusions and Future work

- User adaptation and personalization

User adaptation was a secondary objective of this dissertation and an interesting challenge that consisted in making the bot learn user behavior. While the implemented solution integrated a statistical model to tackle one kind of use cases - time series analysis -, from the feedback of the performed usability tests, there is the need to change this to a more complex solution maybe with collaborative models or even deep neural networks.

- How to provide suggestions

The last challenge was on how to provide suggestions to the user in a way that was not too intrusive and would make the opposite of the desired effect. The selected approach of providing suggestions in the morning, before the user started any interaction with the car, proved to be acceptable because it allows the execution of the pattern recognition algorithm to run during nighttime (crucial for large amount of data) while providing with a useful and friendly suggestion to keep the user from leaving the bot.

Having implemented the designed solution in Section 3.1, several tests were performed, from a questionnaire to inquire how the general public would accept this solution and how pattern recognition would be useful. Also regarding pattern recognition, some experiments were made to test how the model would fit the data under a number of different circumstances. Finally, usability tests were made to validate the ease of use and the purpose of the bot. With this we can conclude that the audience was very receptive to the solution and found the application relatively easy to use inside a car. The pattern recognition algorithm also provided very interesting results that can be considered accurate, making it a feasible solution that captured interest from outside companies like Vodafone.

This work and testing also opens up new possibilities for future work. One of the suggestions taken from the usability tests where the project was explained throughly was that the proactive suggestions could be improved in the way that it would analyze not only the total distance traveled in one day, but capture each trip individually and make suggestions based on that. This would mean that this system would need to be refactored into more of a profile-based system that would contain the pattern recognition algorithm would use data whose base time would be hours instead of days, with the complement of also capturing user preferences such as side mirror positioning, air conditioning temperature, favorite radio station and others. This would mean that when the bot recognized that the user would make a trip, it would turn on the heating, change to the preferred radio station and check the fuel level. To be able to gather such information a partnership would also be needed with car manufactured since the OBD interface does not provide it.

Finally, we can conclude that the goals of this dissertation were successfully completed and validated. A feasible chatbot was built that interacts with the user by voice, able to be user both while outside the car and while driving by connecting to the car’s infotainment system while a the same time fetching data from the car allowing it to make proactive warnings and suggestions by also analyzing past data.
References


REFERENCES


REFERENCES


REFERENCES


Appendix A

Questionnaire Form
Chatbot para integração automóvel

No âmbito do projeto de dissertação "Chatbot Automóvel Adaptável", do mestrado em Engenharia Informática e Computação, desenvolveu-se o seguinte questionário. O objetivo deste questionário é avaliar o interesse da utilização de um sistema alternativo aos sistemas que permitem obter informações sobre o carro, e que sirva como assistente pessoal enquanto conduz.

Mais informações ou sugestões podem-me contactar: tiagommferreira05@gmail.com

*Required

1. Qual é a sua faixa etária? *
   Mark only one oval.
   - [ ] < 18
   - [ ] 18 - 20
   - [ ] 20 - 30
   - [ ] 30 - 40
   - [ ] > 40

2. Alguma vez interagiu com um chatbot? *
   Um chatbot é um sistema conversacional computorizado que entende o significado das frases (por texto ou voz) enviadas por um utilizador e tenta responder o mais adequadamente possível, mantendo uma conversação.
   Mark only one oval.
   - [ ] Sim
   - [ ] Não

3. Como caracteriza a vantagem de um chatbot relativamente a uma interação exclusivamente através de uma interface gráfica? *
   Mark only one oval.
   
   1 2 3 4 5
   Não traz vantagens  Muito vantajoso

4. Conhece algum sistema eletrónico que permita obter dados automóveis em tempo real? *
   Computador de bordo sofisticado ou uma aplicação móvel por exemplo.
   Mark only one oval.
   - [ ] Sim
   - [ ] Não

5. Se respondeu sim à questão anterior como caracteriza a sua utilização?
   Mark only one oval.
   
   1 2 3 4 5
   Nada satisfatória  Muito satisfatória

https://docs.google.com/forms/d/13MOq8epMqDLgqVA2EZe7doj5zF4vgxxzK2YNchCBAg0/edit

Figure A.1: Questionnaire form first page
6. Costuma necessitar de obter informações enquanto está a conduzir ou em viagem num automóvel? *
   Um exemplo de informação seria restaurantes mais próximos.
   Mark only one oval.
   □ Sim
   □ Não

7. Se tivesse a oportunidade de comunicar com um bot através de voz dentro do automóvel de maneira a obter informações úteis e contextualizadas, utilizaria-o? *
   Mark only one oval.
   □ Sim
   □ Não

8. Estaria disposto a pagar por este produto? *
   Produto poderia vir incluído como extra com o automóvel
   Mark only one oval.
   □ Sim
   □ Não
   □ Talvez

9. Se o bot fornecesse alertas e sugestões proativamente analisando o seu comportamento e hábitos no dia-a-dia, como classificaria a utilidade desta funcionalidade? *
   Por exemplo analisar a distância que percorre nas suas viagens.
   Mark only one oval.
   1 2 3 4 5
   Inútil □ □ □ □ □ Muito útil

---

https://docs.google.com/forms/d/13MqOqWYbAq6Lcnh3DEZWx1tqgFkqgaxdQXz9HicCBGjspwedi

Figure A.2: Questionnaire form second page
Questionnaire Form
Appendix B

Questionnaire Results
Questionnaire Results

Figure B.1: Questionnaires’ question 1 results.

Figure B.2: Questionnaires’ question 2 results.

Figure B.3: Questionnaires’ question 3 results.

Figure B.4: Questionnaires’ question 4 results.
Questionnaire Results

Figure B.5: Questionnaires’ question 5 results.

Figure B.6: Questionnaires’ question 6 results.

Figure B.7: Questionnaires’ question 7 results.

Figure B.8: Questionnaires’ question 8 results.
Figure B.9: Questionnaires’ question 9 results.