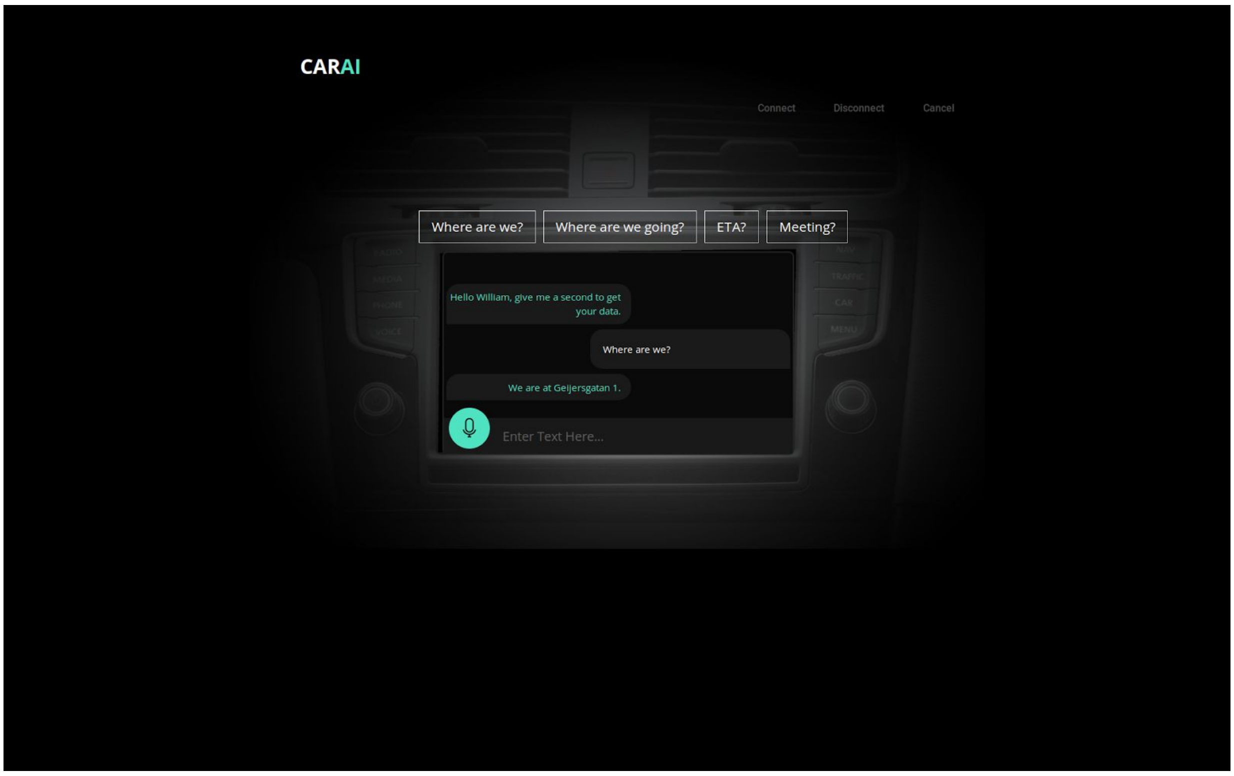


Hello William, give me a second to get your data.

Where are we?

We are at Geijersgatan 1.

CARAI
Development of an intelligent personal assistant for cars

Master's thesis in Algorithms Logic and Languages

William Axhav Bratt and Johan Ekdahl

Department of Computer Science
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2016

Master's thesis 2016

CARAI

Development of an intelligent personal assistant for cars

William Axhav Bratt and Johan Ekdahl

Department of Computer Science
Chalmers University of Technology
Gothenburg, Sweden 2016

CARAI
Development of an intelligent personal assistant for cars
WILLIAM AXHAV BRATT AND JOHAN EKDAHL

The Techno Creatives:
Supervisor: Oskar Hagberg, CTO The Techno Creatives

Academic
Supervisor: Olof Mogren, Department of Computer Science and Engineering
Examiner: Wolfgang Ahrendt, Department of Computer Science and Engineering

Master's Thesis 2016
Department of Computer Science and Engineering
Algorithms, Logic and Languages
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: An example of a run demo of the CARAI prototype seen from the graphical user interface

Department of Computer Science and Engineering
Gothenburg, Sweden 2016

CARAI
Development of an intelligent personal assistant for cars
William Axhav Bratt and Johan Ekdahl
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

This thesis aims to create a proactive Intelligent Personal Assistant (IPA) within a car environment, utilizing location prediction and a face recognition to provide the context for that proactive behavior. A Dialogue Manager was implemented to provide a dialogue driven interface. This was realized as a multimodal interface using both voice and graphical input/output. The location prediction was realized with different neural networks to varying degrees of success. The project resulted in a prototype which acts like a proof of concept for a proactive IPA within a car environment, the prototype demonstrated the proactive behavior that the thesis aimed for. However most of the modules that make up the prototype, excluding face recognition, do not preform well enough to be considered complete, but they do demonstrate a proof of concept.

Keywords: Machine Learning, Intelligent Personal Assistant, Location Prediction, Face Recognition, Dialogue Management, Important Locations, Car, Encog, OpenCV, Talkamatic.

Acknowledgements

First of all we, the writers, would like to thank The Techno Creatives for giving us the opportunity to conduct this thesis. Special thanks to Oskar Hagberg and Sandeep Gajula, Oskar provided feedback and support, and Sandeep provided the visual design of the interface, the project became much better because of them. Secondly a big thanks to Olof Mogren for helping us with feedback and providing us with a deeper insight into common practises of neural network implementations. We would also like to thank the development team of Talkamatic, special thanks to Fredrik Kronlid for providing us with the feedback and support needed to integrate Talkamatic with our prototype. Lastly thanks to Jeff Heaton for providing the neural network platform Encog as a free educational tool.

Without the people mentioned above, this project would never have been possible.

William Axhav Bratt, Gothenburg, June 2016
Johan Ekdahl, Gothenburg, June 2016

Contents

List of Figures

List of Tables

1 Introduction

The introduction will give the ruff outline of the thesis and thereafter follows an explanation the background that inspired this project, as well as the aim of the project that was a direct result of the aforementioned background. Lastly this section contains the limitations that were put in place, so that this project could be completed in a reasonable timeframe.

1.1 Preface

In the following thesis a prototype for an proactive Intelligent Personal Assistant (IPA) within a car environment is developed and tested. To achieve proactiveness, location prediction and an automatic profile selection is implemented along with a *dialogue manager* providing a dialogue driven interface with the information required to proactively act and ask.

1.2 Background

The car industry is moving towards becoming a more digital environment. Large computer related companies such as Google and Apple, are on the front line pushing new software solutions, right into the car. This is why digital applications, that has relevance to cars, can start to be developed right on the car platform.

Digitally aiding the driver, is becoming a topic of interest within the car industry. At the same time, the interest shown for digital intelligent personal assistants (IPA) within the mobile phone industry is growing. A reasonable next step is to combine these two fields, to have IPAs tailored for a car environment. One of the reasons behind this is a greater possibility to gather relevant information from the car environment, compared with a cell phone. An example is that one can, with the help of cameras, monitor the interior of a car. This extra information could be feed to an IPA, giving it more situational awareness. With this awareness a system like this, could take advantage of knowing who the users are. For example linking different settings like air conditioning and favorite music directly to the users, without or with very little direct interaction.

With the help of GPS and online capabilities, a system like this can remember previous meetings and traveled locations. This personal data, could possibly be used to predict an upcoming location or meeting automatically. Apart from providing help with localizing scheduled meetings, a device like this could also offload the driver by notifying other meeting participant that the driver will be late, if such is the case. Which would let the driver keep his or her focus on the road.

Robustness is something that is desired in all forms of applications, this combined with the car industries high demands for fault tolerance, makes robustness a crucial part of this type of software. A bug or flaw in a system, could lead to an accident even if that system is not part of the basic operations of the car. For example a malfunction in one of the quality of life applications could dangerously distract the driver, which in turn could lead to an accident.

When developing an application of this complexity for a car, hardware limitations must be taken into account. But something that can be assumed is that such an application has a connection to the Internet. This assumption is based on the growing trend of connected cars, there are already some newer car models that are already connected [AT&T, 2014]. This presents an opportunity to run heavier computations on an external server or cloud platform, thus reducing the applications computational strain on the cars local hardware.

This project attempts to utilize the potential powers in information gathering that can be achieved in a car environment, to implement a prototype for an IPA. Which main features are centered around: Destination prediction, handling multiple active users, as well as minimizing required user input, in a proactive manner.

1.3 Aim

The aim of this project is to develop a prototype for a proactive intelligent personal assistant for a car environment, that can predict the user's transits and use that information to initiate dialog. As well as perform the role of an interface between the user and the car.
The functionality of the finished prototype is expected to be as follows:

- Location and calendar features such as
  - Ability to proactively give the driver suggestions of locations that should be traveled to, based on earlier gathered location data.
  - Making a routing schedule for the driver and passengers of the car, by using previously gathered driving patterns as well as looking at their individual calendar entries. This routing schedule should be able to provide a GPS, with the data necessary to make a route complete with drop off points for the passengers. This feature should also be able to detect conflicting appointments considering estimated travel distances. Given no conflicting appointments, this system should function without user input apart from calendar entries.
  - Predicting geographically relevant locations corresponding to the calendar appointments and previously visited locations, either through remembering previous occurrences and destination, or through appointment entries.

- Automatic profile selection system
  - In-Vehicle computer vision system, that could set profiles without user input. The user profiles will contain GPS sampling for location prediction, and is therefore a key component in removing the need for user input in the location prediction. These profiles could later easily be expanded to include preferences, such as seat position and radio station.

1.4 Limitations

To make this project fit into acceptable deadlines some limitations were put in place. Firstly building a complete IPA demands a large amount of manhours and is usually done by large teams of developers. Therefore the goal of this project is to build a prototype showcasing some features rather than having a complete product. Secondly security is a large concern when managing a lot of personal data, but due the limiting timeframe, security will not be considered further than the standard security protocols such as SSL for online communication. This is something that needs to be considered for a full scale application. Thirdly Location prediction will be limited/tailored to be able to make accurate predictions in regards to limited areas. Also the exact hardware specifications of modern cars will not be considered further than their performance. Lastly this thesis will not cover the interaction design difficulties as this is not inline with the researcher's expertise.

2
Theoretical Background

The following chapter will detail the theoretical aspects behind the CARAI prototype starting with an brief explanation of neural networks. Followed by an explanation of DBSCAN and a sub chapter on image matching where three different matching algorithms are described. Both neural networks and DBSCAN will be used for location prediction, and one of the image matching algorithms will be used in automatic profile selection.

2.1 Neural Networks

Neural network or more specifically artificial neural networks is a broad field that is usually employed to solve complex problems where there are large amounts of data available. A neural network is a graph consisting of nodes that each have an activation function, the activation function modifies data that passes through its node. The nodes can be connected in various ways creating different types of networks with different characteristics and behaviors, to describe these networks it is helpful to think of them as a collection of layers containing nodes. In this way one can describe an neural network as a input layer, an output layer and the layers in between which are called hidden layers.In more complex networks the layer representation is very useful for describing the connections between the nodes, as connections between layer, however in this thesis the network type feed-forward is used which only has connections in one direction. A visual representation of a feed-forward network can be seen in figure 4.5.

After constructing a network by selecting: amount and size of layers, connections between layers, the activation functions for the nodes. The network needs to be trained, training in the context of neural networks is the process by which the activation functions are adjusted. In supervised learning which is the training method used in this thesis the weights are adjusted with known input-output combinations, running the input trough the network and scoring the networks output against the expected output and adjusting the internal activation functions accordingly.

A simple example, teaching a neural network to behave like a NAND-gate. To do this one would construct a neural network to have two inputs and one output, like a NAND-gate. Then train the network on the NAND-gate behaviour (1 NAND 1 = 0, 0 NAND 0 = 1, ...) for several iterations of training. Once the training has been completed, feeding the neural network with a combination ones and zeroes should yield a correct result or something close to the correct result depending on length of training and how well the network suits the problem, the NAND in our example might get a result of 0.998 for an input of 1 and 0. As long as there is a way to model a problem as having defined inputs and a score-able result, neural networks could feasibly solve any problems provided sufficiently available training data. For example there has been successful attempts at teaching a neural network to play GO. GO is an ancient Chinese board game which is impossible to brute force with today's computer power. ALPHAGO managed to beat the world's best GO player by utilizing the power of neural networks. [Google, 2016a]

2.2 DBSCAN

Density Based Spatial Clustering of Applications with Noise (DBSCAN), is a clustering algorithm made by a team from the Institute for Computer Science at the University of Munich. DBSCAN is a density based clustering method, which is noise resistant and works as follows:

DBSCAN constructs clusters by dividing the points that make up a cluster into two groups, these groups are *core points* and *border points*. To decide what points are considered core and *border points*, one must first decide the parameters minimum points (minP) corresponding to density and epsilon (ϵ) corresponding to what is considered close. *Core points* are the points that are in the densest part of a cluster and are defined as having at least minP points within ϵ distance. *Border points* are the points which are not surrounded by minP amount of points, but are within ϵ of at least one *core point*. Points which are neither *border* nor *core points* are considered noise and thus not part of any cluster. One cluster is a set of points containing all the core points that are within ϵ of other core point of that set, as well as all *border points* associated with those core points. [Ester et al., 1996]

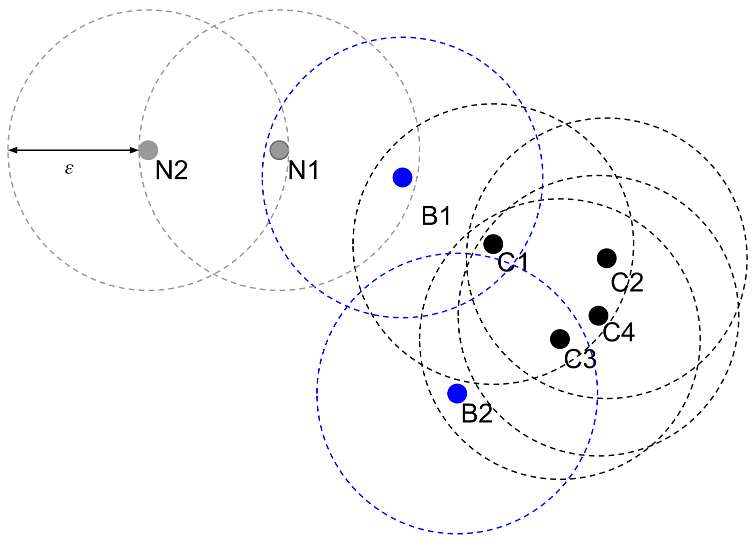


Figure 2.1: Figure illustrating the DBSCAN algorithm. Assuming minimum points set to 4 and a max epsilon of ϵ. The C points have enough points within a distance of ϵ to become *core points*. The B points do not have the necessary amount of points within the distance ϵ to become *core points*, but are in range of *core points* and as such become *border points*. Lastly the N points do not have the necessary amount of points to become *core points* and are not within ϵ range of a *core point*, as such they are considered noise.

2.3 Image Matching

In this section there will be explanations of the three image matching methods, that were considered in the implementation of the automatic profile selection described in section 4.2.2.

2.3.1 Eigenfaces

Eigenfaces is a way extract unique features from images using Eigenvectors. The resulting unique features can be used for a variety of purposes, such as representing images as a combination of these unique features. But the more interesting application in terms of this project is face recognition. Face recognition with Eigenfaces works by extracting Eigenvectors from the input image and comparing them with the Eigenvectors extracted from a training set of images. Resulting in a face recognition that is resistant towards lighting differences and different tilt of the head, but has the restriction of a set resolution as a side effect of how the Eigenvectors are extracted. [Turk and Pentland, 1991] [Team, 2016]

2.3.2 Fisherfaces

Fisherfaces is a way to use Eigenvectors to classify images rather then represent them. The classification is done in such a way that the distance, error, between the elements of the same class becomes as close as possible, while the distance between the classes become as great as possible. In terms of recognising faces this is very useful as face recognition is a classification problem. Fisherfaces share the need for a fixed image resolution. Compared with Eigenfaces, [Belhumeur et al., 1997] found that Fisherfaces were more resistant to lighting conditions.

2.3.3 Local Binary Pattern Histogram Matching

Local Binary Pattern Histogram matching (LBPH) traverses an image, for each pixel in that image the surrounding 8 pixels are stored as a color value in a new image. Each of these 8 pixels are thresholded in such a way that if the pixel's color value is higher or equal to the center pixel, it will be set to one and zero otherwise. This leads to each pixel in the new image being encoded with an binary number of eight bits, where each bit is the threshold of a neighbouring pixel. Finally the matching is done by matching the local histograms of the new image, with a training set of histogram sets extracted in the same way. LBPH is lighting resistant and does not require a set resolution for the image matching. [Team, 2016]

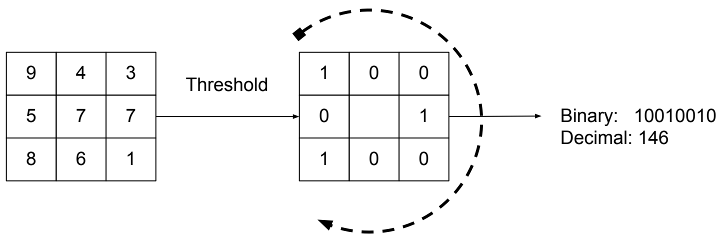


Figure 2.2: An example of the local histogram pattern algorithm of an image with 3x3 pixels. The threshold is in this example is 7 and produces an encoded decimal number of 146. The Binary pattern is read from the top left corner going clockwise. Inspiration taken from OpenCV

3 Related Work

In the related work section, papers related to location prediction are referenced. Furthermore existing platforms which are related to and have inspired this project, are described.

3.1 Location Prediction

Within the field of location prediction the primary focus, of research, has been to predict pedestrian movement, which is more complex in terms of the possible space of movement; Cars are restricted to roads and parking lots, whereas pedestrians are not restricted to follow predefined routes and may take any method of transportation. Predicting pedestrian movement usually involves a large amount of data and rely on that the data is continuously sampled, so that the path taken can be used in the prediction.

In the paper Mathew et al. [2012], a location prediction algorithm using Hidden Markov Models is presented. Their aim was to predict pedestrian movement across the entire globe. Their method for discretization of data was to divide the globe into triangles with a Hierarchical Triangular Mesh and group time according to day, night and if it was a weekend. The end result was a prediction accuracy of about 13.85%, average error in distance of 143.5 kilometers, and a median error distance of 4957 meters. In relation to our thesis this type of prediction is not directly applicable as the continuity of samples of an individual is not guaranteed. Users might be picked up in one place, dropped of in another and then be picked up in a third for example. Secondly the problem this thesis aims to solve with location prediction requires a higher degree of accuracy.

Another location prediction method for predicting pedestrian movement is presented in Gambs et al. [2012]. This method uses what is referred to as n-MMC which is a variation of the Markov Mobility Chain that takes into account the previous n locations. What makes this interesting in terms of this project, is that with the rather low n of two they achieved prediction accuracy between 70% to 95%.

Ashbrook and Starner [2003] make the observation that finding significant locations helps the prediction process. For example predicting that the user will move a few meters to the west is not very helpful, because it is likely that this is the same location. Their approach for finding important locations allowed for multiple resolutions, which was achieved with a hierarchical structure of locations, that in turn could be divided into locations. The predictions were made through a n-th order Markov chain, but its accuracy has not been documented. Unlike the well documented tests for finding the important locations, there was no mentioning of how well the predictions corresponded to reality. In relation to our thesis, what is of interest is their work on finding important locations, as it serves as a great point of inspiration for part of our thesis. What differs is that the solution suggested in our thesis, does not have the same need for multiple resolutions. As there is only one form of transit, by car, and therefore takes on a different solution to the what is essentially the same problem.

3.2 Existing IPA Platforms

3.2.1 Sirius

Sirius is an open-source IPA platform developed at Michigan University. The platform provides an automatic speech recognition tool, an image matching library, and a questions-answer library. Data exchange between these modules is handled by a python client-server architecture.
Speech Recognition capabilities are powered by either a combination of Hidden-Markov-Models and Gaussian Mixture Models, powered by Sphinx or with the help of a neural network powered by Kaldi. Both these automatic speech recognition libraries are further explained in section 4.2.6.1.
Sirius image matching is done through OpenCV where it compares an input image, with an already created image database. The software scores possible images from the database based on how similar the features of that image are compared to the inputted image. The highest matching result is returned. This results in a quick match that has no failure state, a best matched image is always returned.
Sirius‘s question answers module is powered by OpenEphyria (OE), OE is connected to a local copy of the Wikipedia Database. By the usage of different information extraction techniques OE creates relevant search queries that are given to the Wikipedia database. Depending on different scores returned by possible matches, OE chooses and returns the best answer possible [Hauswald et al., 2015]

3.2.2 OpenCog

OpenCog is an open-source project that strives to develop a platform for Artificial General Intelligence (AGI). It is developed and maintained primarily by the OpenCog Foundation. The project has an ambitious goal of being able to achieve an AI with *human preschool-level intelligence*, within years and not decades if their roadmap is followed. Currently the framework contains “a number of cognitive agents at varying levels of completion” [OpenCog, 2016]. Among other projects developers has created a Nao robot [ROBOTLAB, 2016] controlled by OpenCog and a learning Minecraft bot using this framework. [Goertzel, 2010] As of writing this thesis, OpenCog is an advanced library but unstable as it is under rapid development. OpenCog also does not have many pre-built features, which is unfortunate as pre built features are helpful when developing a project of this size in a short time frame.

4
Description Of The System

The following chapter will give a thorough description of the CARAI prototype, providing insight to design decisions and how the prototype works. Starting with describing the target platform, followed by a description of the system witch details all of the software modules of the prototype.

4.1 The Target Platform

The target platform for the software system described in this thesis, would have to comply with the following:

An in-car computer: Be able to run on computer hardware that, could feasibly exist or be installed in, a car. This means it should be compact, consume little power, and have robust hardware that preferably does not need active cooling. A computer that meet these criterias is a Raspberry Pi 2 Model B [PI, 2016b]. The Raspberry Pi can perform under passive cooling and is powered through a 5V USB port [PI, 2016a]. It is made to be an experimental board and therefore it is designed to be robust. The Raspberry Pi 2 Model B uses an Broadcom BCM2836 processor, which is a 32-bit quad-core ARM processor running at 900 MHz.
Alternatively a Next Unit of Computing (NUC) computer [Intel, 2016] could be targeted, these computers are more lenient when it comes to hardware, as they contain standard computer components, such as Intel Core processors etc. This means the software performance optimizations causes less of an issue, as the hardware is more powerful.

An in-car camera monitoring system: One or several cameras providing coverage each of the occupants in the car, this could for example be achieved by a centrally mounted camera, above the rearview mirror or placed on the dashboard.

A sensor array: Sensors that can detect when the interior of the car might have changed, making it possible to run face recognition only when it is necessary. An example of how this could be done is by having sensors that detect the opening and closing of the car doors, or alternatively seat pressure sensors. This approach was selected over an active face recognition approach where face recognition would always be running, as having face recognition always running posed performance concerns. More on this can be found in the discussion at chapter 7.

A CAN bus interface: The CAN interface is to access the car sensors and allows the prototype to both sample and set the car's GPS system. The CAN interface can also give possibilities to automatically set user specific settings. An example of this could be the positioning of mirrors, which is driver specific, and the seat position could also be set for each user's seat independently.

4.2 The System Architecture

The system is divided into several modules, that are managed both as client and server applications. The figure below represents what components exists and how they interact with each other. The following chapter will give a more detailed description of what components exists and what they do.
The system is divided into six different self contained modules. The Database (4.2.1) stores GPS logs together with user specific data. The Face Recognition module (4.2.2) is used to create an automatic profile selection. The Predictor module (4.2.3) is used to create prediction where one will travel based on travel history. The Talkamatic module (4.2.6) is used to handle dialogue management together with multimodal inputs and outputs to the system. MQTT (4.2.5) is a TCP/IP based connection protocol to handle connections between the modules. Finally the Car module is a representation of car signal. Each module will be further described in the referenced sections. How these modules are connected with each other can be seen in figure 4.1.

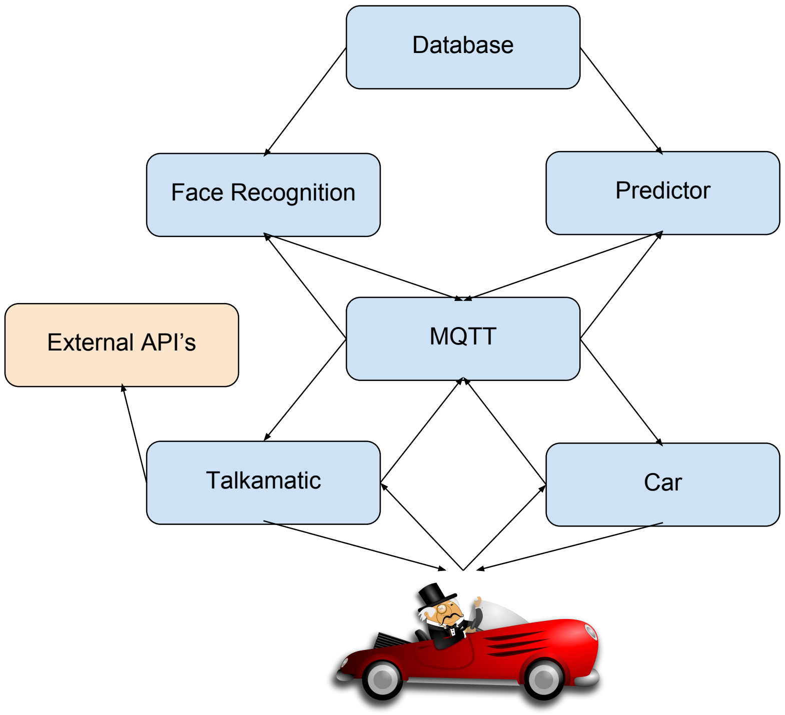


Figure 4.1: The boxes represent modules, the arrows represent which way information is sent and the man in the tophat at the bottom of the figure is the user. As can be seen both Face Recognition and Predictor is connected to the Database module, this is because Face recognition needs to know what name corresponds to the found ID and the Predictor needs to know the GPS log of a certain ID. The MQTT module is connected to and from every core module, acting as a data bus.

4.2.1 Database

The Database holds the names and GPS logs mapped to there respective IDs, the names are used to greet the user and the GPS logs are used in location prediction. MySQL [MySQL, 2016] was used as it has all the necessary features for the type of database layout that this project uses.
The layout of the database consists of two tables and six stored procedures. The tables are *PositionHistoryTable* and *UserInfo*, *UserInfo* maps ID's to user names while *PositionHistoryTable* maps trips to IDs. A trip is five columns containing two positions as longitude and latitude corresponding to a start and end point, as well as a date/time column containing a timestamp with the precision down to minutes. In the *UserInfo* table IDs are auto incremented primary keys pared with a name and yet to be implemented settings variable. The reasoning behind this setup is, firstly that every new user should have an unique ID even if they do not have a unique name. Secondly each user should be able to have any number of logged entries, including no entries. The stored procedures are as follows: *createUser* which creates a user, *deleteUser* which deletes a user, *enterPositionData* adds a trip associated with an ID, *getAllUsers* returns all users and their respective IDs, *getPos* returns a specified amount of trips associated with a specified ID, and lastly *getUser* returns

the name of a specified ID.

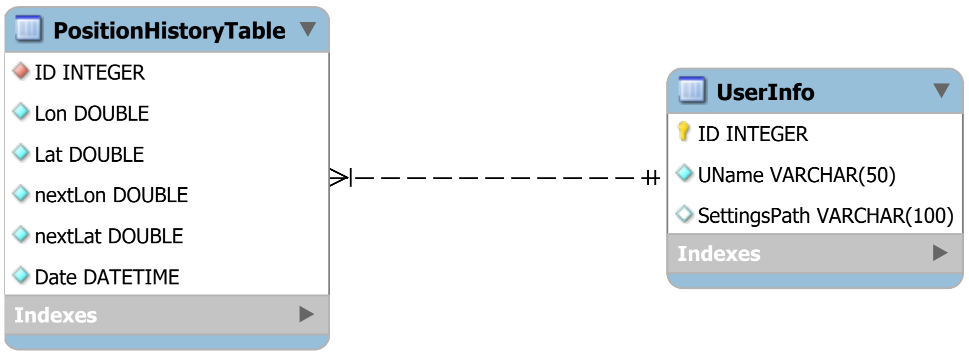


Figure 4.2: In this figure the database structure is illustrated, as can be seen the column name ID is shared between the two tables. This is because ID acts as an one to many mapping using the primary ID key from *UserInfo* table that represents a users identity, to map a users logged entries in the *PositionHistoryTable* where ID is not a primary key.

4.2.2 Automatic Profile Selection Through Face Recognition

The pre-study, at the start of the project, showed that OpenCV [Itseez, 1016] has all the necessary tools to build a face recognition algorithm, with the secondary benefit of performing all the computations locally. OpenCV has been used to implement similar features in Sirius. Performance wise OpenCV has been shown to run face recognition applications on smartphones [Liu, 2014] which are well below the hardware constraints set in this project. [Itseez, 2016]
To detect if registered users are in the picture, two standard methods from the OpenCV library are used. First the areas where faces are located is found using a cascade classifier, the faces are cropped to minimize the effect of the background. The resulting cropped images are then matched with reference images, of users, using the method of Local Binary Pattern Histogram Matching. A match will always be found, but with different confidence levels as LBPH Matching looks for the closest match.
Knowing which users are present in the car is useful as it removes of the need for the users to identify themselves. This in turn leads to an increased perceived intelligence as the application appears and to some extent does know who the users are. In addition to finding what users are present, the algorithm can also contextualize their location within the car, opening possibilities to for example address the driver. Detecting where in the car a user is located is done by dividing the image into a grid and depending on where the person's face is detected, in the image, it can be assigned to a specific seat. This method is scalable for a multi-camera setup by concatenating the resulting images and adjusting the grid appropriately.
In the prototype there are ten pictures taken, so this procedure is repeated ten times. After that the most frequent face is picked, per seat, and assigned a confidence value equal to the mean confidence value of all occurrences of that face. A minimum

confidence threshold is set so that unknown users will not be identified as other known users. The confidence is normalized to meet the expected confidence interval of Talkamatic, which will be described in detail in the Talkamatic sub-chapter, see section 4.2.6.2.

4.2.2.1 Improving Confidence

Using a re-normalization (values [0.6-1.0]) of the confidence yielded from the face recognition Talkamatic will respond accordingly: On a confident identification the user profile will be set; On a almost confident identification, the user will be informed of who the user was identified as; On an uncertain identification the user will be asked to confirm the identification; On a failed identification the user will be asked about the user's identity. In the case of an uncertain but correct identification, the pictures that were taken in this analysis are added to that user's profile so that future attempts will yield higher confidence. If the user answers no to a confirmation question, the confidence is lowered to unknown user and the system asks for the user's name.

4.2.2.2 Proposed Setup

A camera is setup so that it has coverage over all the seats, two possible placements could be centered on top of the dashboard or by the rearview mirror. Ten pictures are taken when a door is closed. The pictures are divided into different regions, each corresponding to a different seat. Then the face recognition software analyzes the pictures and finds faces on a seat by seat basis to find out if and what has changed.

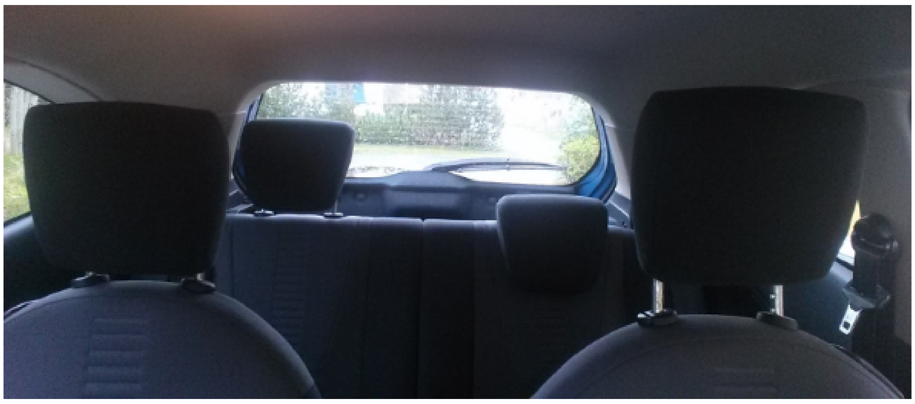


Figure 4.3: A view of a Ford KA from the rearview mirror. If the driver and all passengers keep their heads in a default position all faces are detectable from this view

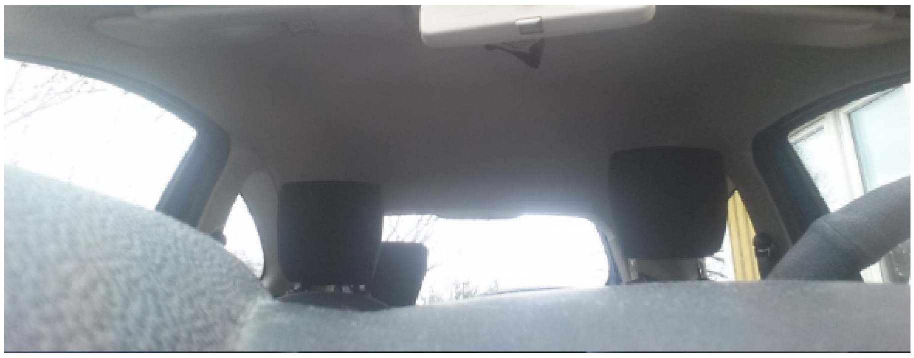


Figure 4.4: A view of a Ford KA, from the dashboard. Driver and front seat passenger are clearly visible in multiple head positions, where as the back seat passengers are hidden. More suited for a multiple camera setup.

4.2.3 Location Prediction

The primary feature of this IPA is that it can predict where the car will travel, by performing analysis on previous traveled data. The following section will describe in detail how the location prediction is performed with neural networks.

4.2.3.1 Gathering Training Data

To perform accurate machine learning, a good training data set is required. Collecting data presents two challenges, firstly how initially train the network, secondly how to collect data during the application's runtime.
To acquire personally relevant data for initial training of a network, GPS logs can be extracted from devices such as smartphones. Smartphones tend to passively logs GPS data for a lot of features, such as Google traffic, iCloud:Locate your device or Google locations. Because of new directives from the EU [EU, 2016], all data gathered is owned by the user and therefore must be made available to him or her and as such this data can be used, by the user, to initialize the neural network. Some prepossessing is required to extract the relevant data from the continuously sampled logs, see section 4.2.3.2.
Gathering data during runtime is not a great challenge, as the face recognition combined with sensors provide information of when and where users enter or leave the car. Logging a user's travel is done by combining where and when they entered the car with where they exited the car, storing it as an entry.

4.2.3.2 Training Data Preprocessing

As mentioned earlier most passively collected data is collected continuously. The data that is interesting is the start and end location of uninterrupted movement, in the following sections this shall be referred to as a path. The paths of interest to the prediction are paths taken by car. The distance between continuously gathered samples are often short, within tens of meters. Depending on the user this data could also contain high amounts of samples in close proximity to the user's home, which in its turn would mean that paths that would start in the home has its destination in a neighbouring room. These types of paths are not relevant for predictions related to car travel. The data must be filtered to make the data more relevant to predict car destinations. The most critical aspect of these filtering methods are that the paths taken must be intact after a point is removed. If a point is removed then the point before it, has to be connected the next point that is not removed by the filtering algorithm. Three culling techniques were tested to filter the data:

Box culling:
: Box culling is a method where a box of a certain size is placed over the sampled points. If the points are inside the box they are kept else they are removed. This is used to remove points that are sampled outside of a desired area, a town or a country.

Speed culling:
: Removes the entries that have a lower speed than a specified threshold or that has zero distance traveled or that has zero difference in time. Speed culling also has a threshold for the amount of slow samples required to signify a stop.

Distance culling:
: Distance culling uses the concept that if a path is longer than a certain distance than it most probably was traveled by car. The method uses two modes a standstill mode and an traveling mode. The traveling mode is triggered when paths are longer than a distance γ only the starting point is saved. Standstill mode triggers if the distance traveled is shorted than γ for x amount of samples. The mean off all standstill points are added as the destination point of the trip.

4.2.3.3 Selecting clustering method

Clustering is a the very broad field of grouping data points together through some metric of locality. But it is not only metrics that differ, different algorithms have different approaches and therefore different result when clustering the same data set. Examples of types of clustering methods are location, centroid, connectivity, and density based clustering. Each have their uses, but for the purpose of clustering location data only density based clustering seems reasonable, as dense regions is what would constitute an *important location*. There exists density based clustering algorithms that are noise resistant, in such an algorithm stray points will not affect the result of the clustering. Such stray points could be created if for example the driver's favorite coffee place is closed for a day, forcing the driver to find another. The clustering algorithm used in this project is one such noise resistant density based clustering algorithm, DBSCAN see section 2.2.

4.2.3.4 Neural Network

The neural networks were implemented in Java™, using the machine learning framework Encog [Heaton, 2016]. Three neural network configurations where implemented, each containing three hidden layers whose size are determined by Encog at the point of training. Each using TANH as the activation function for the first two hidden layers and a linear activation function for the last layer.

4.2.3.4.1 Network 1

Network 1 has: a GPS position, longitude and latitude, *day of week*, *time of day* as input and a GPS position as output. The longitude and latitude are double precision variables, *time of day* and *day of week* are discrete integer values from an ordered set ranging [0-1439] and [1-7] respectively. Ordered set means that unlike an unordered set where every element is equally different from every other element, the ordered sets elements are as different as the difference in index.

Inputs		Hidden Layer	Outputs	
GPS Latitude	Double	3 Hidden Layers Layers:	GPS Latitude	Double
GPS Longitude	Double	Layer1: Activation function: TANH Amount of Nodes: 2 + 7 + 1	GPS Longitude	Double
Time of Day	Set of ordered integers [0-1439]	Layer2: Activation function: TANH Amount of Nodes: 7		
Day of Week	Set of ordered integers [0-7]	Layer3: Activation function: Linear Amount of Nodes: 2		

Table 4.1: This table shows the structure of Network 1, time of day is represented in minutes

4.2.3.4.2 Network 2

Network 2 has: A *important location* ID, *time of day*, *day of week* as input and *important location* ID as output. *Important location* ID is an unordered set of integer values, ranging from, one to the maximum number of clusters. *time of day* and *day of week* are ordered sets of integer values that range from [0-1439] and [1-7] respectively.

Inputs		Hidden Layer	Outputs	
Important location ID	Set of unordered integers	3 Hidden Layers Layers: Layer1: Activation function: TANH Amount Nodes: Number of *important locations* + 7 + 1 Layer2: Activation function: TANH Amount Nodes: 7 Layer3: Activation function: Linear Amount Nodes: Number of *important locations*	Important location ID	Set of unordered integers
Time of Day	Set of ordered integers [0-1439]			
Day of Week	Set of ordered integers [0-7]			

Table 4.2: This table shows the structure of Network 2, time of day is represented in minutes.

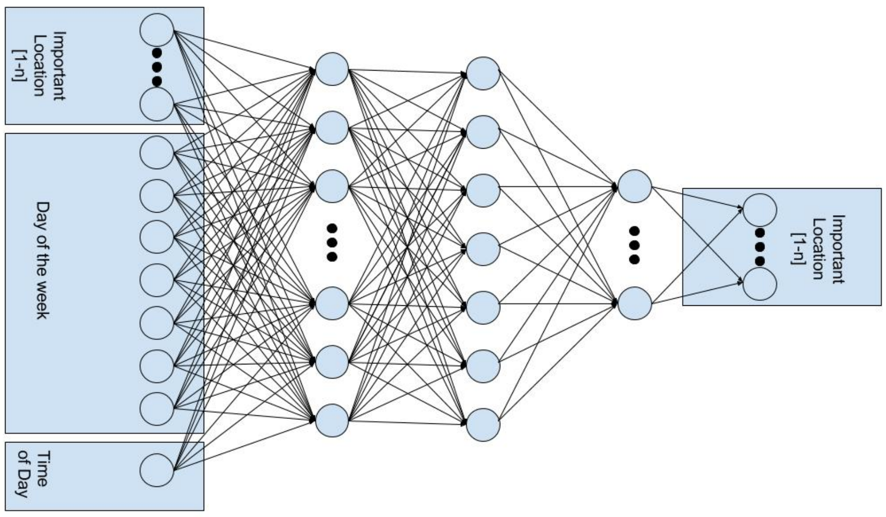


Figure 4.5: A visual representation of Network 2, a fully connected feed-forward neural network. Who has varying numbers of input and output nodes depending on the number of *important locations*, which in turn makes Encog vary the number of nodes in the first and third hidden layers.

4.2.3.4.3 Network 3

Network 3 has: A *important location* ID representing the current location, a *important location* ID representing the previous location, *time of day*, *day of week* as input and a *important location* ID as output. *Important location* ID is an unordered set of integer values, ranging from, one to the maximum number of clusters. *Time* *of day* and *day of week* are ordered sets of integer values that range from [0-1439] and [1-7] respectively.

Inputs		Hidden Layer	Outputs	
Important location ID	Set of unordered integers	3 Hidden Layers Layers:	Important location ID	Set of unordered integers
Important location ID	Set of unordered integers	Layer1: Activation function: TANH Amount Nodes: 2∗number of *important locations* + 7 + 1		
Time of Day	Set of ordered integers [0-1439]	Layer2: Activation function: TANH Amount Nodes: 7		
Day of Week	Set of ordered integers [0-7]	Layer3: Activation function: Linear Amount Nodes: Number of *important locations*		

Table 4.3: This table shows the structure of Network 3, time of day is represented in minutes.

4.2.3.5 Neural Network Prediction Procedure

The method put forward in this thesis that use neural networks for location prediction, consists of several steps.

Step 1: Gathering GPS data for the users in the form of start and end locations for their travels. It is important to not flood the prediction process with data that is irrelevant to the prediction, as this can lead to incorrect predictions. Filtering data is a possible solution but it is far easier to take correctly samples to begin with. To this end, samples are taken and logged individually when a user enters or leaves the vehicle. This is accomplished by running the face recognition when the doors of the car have been closed, as this constitutes the possibility of a user entering or leaving the car. Entering or leaving the car via windows, although possible, is an edge case and is not considered.

Step 2: The coordinates for the start and end points are grouped into points of interest, as it is incredibly unlikely that the car will stop at the exact same GPS coordinate every visit to a certain location. Therefor a clustering algorithm is used to group the GPS coordinates so that a start and endpoint goes from *important location* to an *important location*. Where the GPS coordinates of an *important location* is calculated as a mean of the associated coordinates. It should however be noted that one of the networks does not use this approach, but instead relies on the neural network's ability to classify locations.

Step 3: Paths corresponding to the start to end *important locations* along with the time from the start sample is feed as training data to a neural network. The training can and should be done on a remote platform as the training is quite demanding and generates a small and easily transmittable neural network.

Step 4: Using the model created by the training data, a prediction is created. Depending on the certainty of the prediction the driver will either: be presented the prediction, be prompted to agree or select one of several predictions, or the prediction will be ignored. The next step is to give the car's GPS the coordinates of the predicted location.

Step one should run for a considerable amount of time in between step two and three, weekly or monthly recalibrations are reasonable. A few new data sample points will not change the neural network much.

4.2.4 External APIs

There are some external APIs connected to extend the possibilities of the application, helping to provide a more complete experience. All these external components are primarily connected to the TDMs device interface, which will be further explained in the Dialogue Manager chapter 4.2.6.2.

4.2.4.1 Google Calendar

To be able to provide synchronized calendar information between the IPA interface and regular services, Google calendar was chosen. This partly because it is provided with an excellent API documentation as well as language and system independent libraries for easy and quick installation and development.
Google calendar provides the same support Google's standard web interface, this means full support for creation, modification and deletion of calendar events.[Google, 2016b]

4.2.4.2 Google Geolocations

Google geolocations is a sub-API under Google Maps. The data provided by this API is related to getting specific data from a GPS positions, for example street names. It is also possible to do the reverse by sending in a street name or name of a relevant locations, it can generate possible gps locations that match that description. [Google, 2016d]
The main feature that the IPA utilizes from this library is the position to name functionality. When the user asks the IPA where he currently is, the IPA can respond with the current street address rather than a longitude, latitude-pair.

4.2.4.3 Google Distance Matrix

The final Google APIs feature the IPA is using is the Distance Matrix API. This library provides functionality to get the distances between two given coordinates considering traversable roads. The other feature is that with this distance combined with real time traffic data, this library can provide an estimated arrival time to the destination. [Google, 2016c]

4.2.5 MQTT

Message Queue Telemetry Transport (MQTT) [ISO 20922:2016] is a publish subscribe messaging protocol. It is a TCP/IP server-client based protocol where a broker is set up as the server, and the different components of the program acts as clients who are connected to topics. If a client publishes new information all subscribed clients will be notified that their subscribed topic has been changed.
This is really useful since the full applications is built upon a lot of different modules, written in different programing languages. By sending a standardized JSON string between these modules, there are no language barriers. A new component written in any programing language language can be initialized to listen and speak with the system, requiring only an MQTT interface, as well as a JSON formatter/parser. In this project's case there are Java, Python and JavaScript components and all are connected with each other through MQTT.

MQTT has fault tolerance features *Wills*, a *Will* is a description of what should be done if a client should disconnect unexpectedly.

MQTT is very useful when testing and debugging, when using debugging tools such as Node-Red all topics can be monitored to verify that information is correctly sent and managed. There also exists a possibility to inject messages to specific components during runtime to test behaviours and or bugs. Node-Red makes injecting signals and messages easier. An example of a Node-Red flow, that was used in this project, can be seen in figure 4.6. [IBM, 2016b]

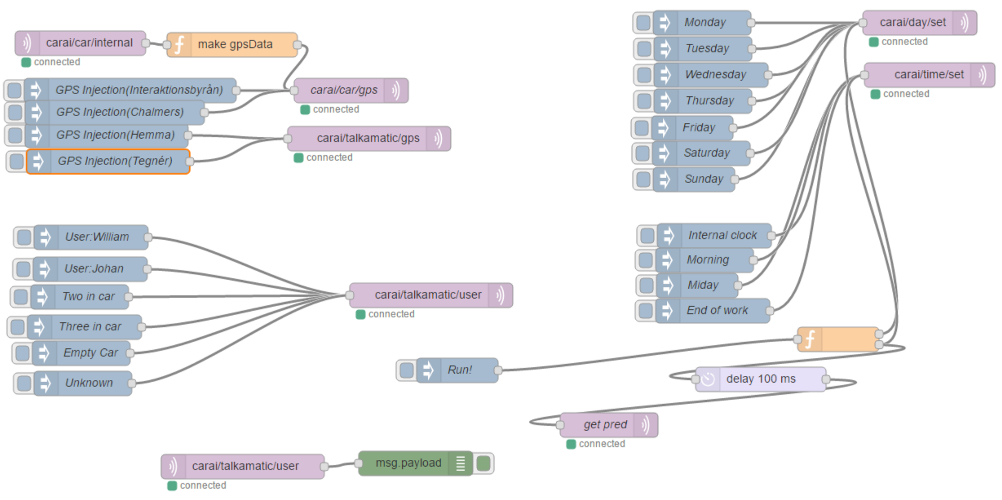


Figure 4.6: An example of a Node-Red testing flow. The blue squares are inject nodes where one can send data. The purple nodes are MQTT connection points and the orange are custom JavaScript functions.

4.2.6 Voice And Dialogue

Early on in the design and research process it was decided that Voice and Dialogue would be the preferred interface method for the IPA, as manual interaction is undesirable when driving. The dialogue management system consists of three parts: Automatic Speech Recognition, a Dialogue Manager and a Text-To-Speech synthesizer.

4.2.6.1 Automatic Speech Recognition

An ASR converts spoken words into text and different ASRs do this with different degrees of accuracy. The ASR software that was evaluated was the online solutions: IBM Bluemix, powered by IBM Watson [IBM, 2016a], Nuance Dragon and the offline solutions Sphinx [CMU, 2012] and Kaldi [Povey et al., 2011].
The offline solutions Kaldi and Carnegie Mellon University (CMU) Sphinx, are both open source ASR solutions. CMU has several projects among them the lightweight application PocketSphinx written in C, this is the library CMU recommends for real-time application. It has been successfully tested on embedded devices and on mobile phones, such as IPhone and Nokia. Sphinx4 is their most recent incarnation, written in Java, it is primarily developed for cloud computing and web services and therefore not encouraged to run locally on a lightweight machine. [CMU, 2012]

Kaldi is a C++ open source ASR application, it performs its synthesis through Gaussian Mixture Models (GMM), which is the conventional way to model acoustics. Most prevalent is the fact that Kaldi has a highly non restrictive licence. Tests from another research group found that Kaldi had a fast executing time of around 4 seconds. [Povey et al., 2011]

Bluemix is IBMs cloud based general purpose commercial API. With direct possibilities to implement Watson related interfaces, such as a Dialog interface for natural dialog management or Visual recognition as a machine learning interface for image recognition. [IBM, 2016a]

Pocketsphinx biggest strengths are, that it runs on a local machine and is open source, but its accuracy was not deemed good enough. Initial testing showed good results on the pre-trained dialogue samples, but when used with custom recorded audio, the accuracy was much lower. During tests, Kaldi did not perform as well as the performance claimed by Hauswald et al. [2015]. With far longer response times and with varying accuracy. The high response times could be due to the hardware or virtual machine setup, it was tested on. Both of the online solutions had better recognition, but had some delay due to the data transfer. The delay was not significant enough to obstruct the responsiveness of the system. In the end this project used IBM Bluemix, but a future implementation can employ any ASR without significant changes, as long as the ASR can provide an interpreted string.

Another aspect to be considered was how ASR should be integrated with the TDM. The two main interaction methods with an ASR are: Push-to-Talk and Wake-up-words [Këpuska and Klein, 2009]. Wake-up-word (also known as Trigger word or Magic word), implies that a device is constantly listening, only reacting to commands when a certain word is uttered. An example of this is Google Now, which is using the command “Ok, Google” to start figuring out the intention of the user. The problem with this solution is bandwidth limitations, the existing services are allowing limited usage of their service before requiring a subscription. This means running an always on application will use that limited amount of bandwidth. Because of this it was decided that the prototype should use Push-To-Talk, with a silent detection to determine when the command ends.

4.2.6.2 Dialogue Manager

In order to improve the dialog interface from a simple commands and raw string output, it was decided that a dialogue manager (DM) was to be integrated with the prototype. Talkamatics DM was chosen as Talkamatic offered to provide support with the development and integration.
Providing the Talkamatic engine with a custom Dialogue Domain Descriptions (DDDs), allows it to perform dialogues using the necessary rules and context provided by the DDD. The DDDs consists of four modules: Ontology, Domain, Grammar and Device.

The Ontology consists of variable declarations, predicates, and sorts. A sort is a collection of words that can be associated with a certain type of predicates. An example of this could be a contact list. The contact list contains all known contacts, if you want to select a user this utterance will be compared against the contact list sort. Here Talkamatic can verify that the user exists and that it has understood the command correctly.

Domain is where the plans are created. The plans are the general structure of how a command/procedure in Talkamatic should be executed and what should be done thereafter.

The Grammar module of Talkamatic contains the grammar of the specific DDD. Here one can create several possible utterance for a certain command, as well as define how Talkamatic should respond. The grammar is loosely connected to the DDD which allows for easy switching between several languages or language configurations.

Device creates the interactions with the outside environment. Because it is created with pure python code it is highly adaptable, to the specific use cases of the DDD is built for. In the case of the prototype that is developed in this thesis, Device handles for example the integration with MQTT. Furthermore the Device also has a local model of the car stored, which is updated with MQTT events. MQTT can also externally trigger events for example greetings. Device also connects the application to External APIs and connects Talkamatic to the database.

Confidence, and Talkamatics built in confidence management is an important aspect of this projects dialogue management. Dependant on the confidence values produced in the other modules of the program, Talkamatic will react differently creating the proactiveness the project strives to create. The confidence, with a value between 0 and 1, is set into 4 distinct parts: Low uncertainty, uncertain, certain, high certainty. If Talkamatic is given a result with Low uncertainty (a value between [0-0.4)) it will ignore the event as the certainty was to low. If the result is uncertain, a value between [0.4-0.6), it will acknowledge that an event has happened but will ask the user to resolve the uncertainties. If the result is certain [0.6-0.8) Talkamatic will acknowledge the result querying the user if the produced result is correct, with a yes/no question. Finally if the result has a high certainty [0.8-1.0] Talkamatic will assume that the produced result is correct executing the command on its own.

4.2.6.2.1 Example commands

The following are some example commands that are implemented into the prototype. In the examples the users utterance will be displayed as U> and the Talkamatic response will be displayed as S>

Customized Greetings The face recognition data is given to the dialogue manager, if there are any information that is unclear this will be resolved in a conformation question to the users. All users that are detected and confirmed are greeted in an introductory message. This also leads to the activation of their user profile and gives the opportunity to start logging location data.

Navigation When profiles are set and loaded, the location prediction module starts generating a prediction with a confidence. This confidence is verified as well and the published to the navigator. The verification takes in regard known named locations, this means it is not possible to just give the DM an address and it will know where it is. At the time of writing this table of known locations are hard coded into a hashmap in the DM. It is also possible to ask for the estimated arrival time during a trip.

Car Control There exist frontend possibilities to set the temperature of the air conditioning in the car.
- U> Set temperature to 20 degrees
- S> Ok setting temperature to 20 degrees

Calendar Through the Google Calendar API it is possible to query for the next meeting and ask for specific meetings within a week forward in time. Next meeting will also tell you how long time is left until the event is scheduled.
- U> What is my next scheduled event?
- S> Your next scheduled event is an Event Name in Time left

or
- Do i have a scheduled event on Thursday?
- S> Yes you have an Event Name at Scheduled Time

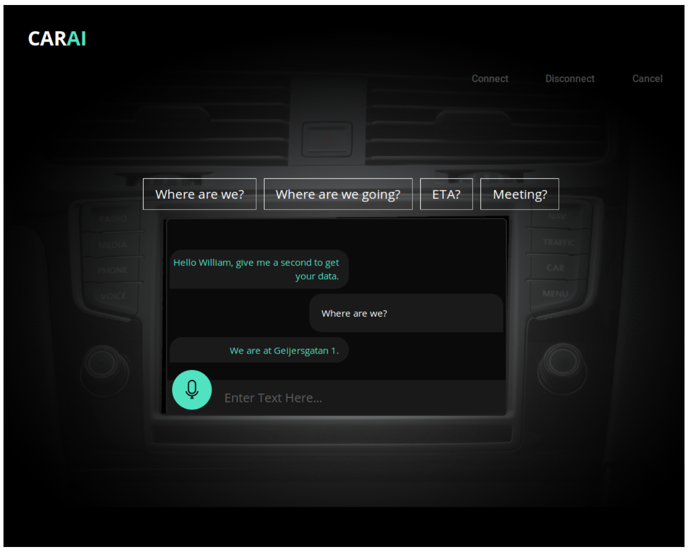
User Creation Another goal was to be able to create an user during runtime. If a new user enters the car, he would be able to create a new profile, and start logging new data directly and saving settings to a personal profile. The problem was the fact that the Sort collections in Talkamatic, has to have recognizers that are predefined. This means to be able to create a new profile, the recognizer must have access to all names and match against them. This is not an option as matching thousands of names will take too long. To go around this issue users must be entered through the device part of Talkamatic, which is a possibility but then Talkamatics standard utterance methods can not be utilized. Utterances are directly linked to voice commands in Talkamatic and although it might be possible to find an alternative solution, it would take to long to implement. Therefor on the user creation during runtime was scrapped so that it would not interfere with the design goal of seamless voice interaction.

4.2.6.3 Text-To-Speech

Text-To-Speech (TTS) converts text into spoken words and different TTS software does this with different degrees of accuracy. Two restrictions where early on recognized when it came to choosing and implementing a TTS module. Firstly it has to be compatible with Talkamatic, which is not directly compatible with all different TTS modules. Lastly the goal is to run as much software on the local machine as possible, to reduce delay which comes from uploading and downloading data to external servers. A program that met both these requirements was Festival, which is a linux based voice synthesizer developed by the University of Edinburgh. [Univeristy of Edinburgh, 2016]
The default voice package provided by festival was deemed a bit too robotic sounding. But there was possibilities to install other voices on top of the existing library, which were more natural sounding. These are provided by Carnegie Mellon University (CMU) and the chosen voice package was the CMU_SLT_ARCTIC package, which is recorded by an American female with experience of building synthetic voices [Black, 2015]. This change created an improved perceived experience due to its capability to reassemble regular speech.

4.2.6.4 Graphical Interface

Talkamatic also provides possibilities to integrate a graphical user interface (GUI) to the application. Through the available websocket interface a GUI was implemented, as having the ability to monitor and verify the application was desired. Furthermore after discussion with the Talkamatic team, it was also concluded that having access to quick commands can sometimes be more useful, than a pure voice based interface. Another aspect that was considered, was Talkamatics feature to provide a list of possible inputs when a question needs more information. These suggestions can be provided as buttons for quick interactions. The graphical interface combined with the Voice interface provides a multimodal experience where the user chooses him/herself how the system should be interacted with.

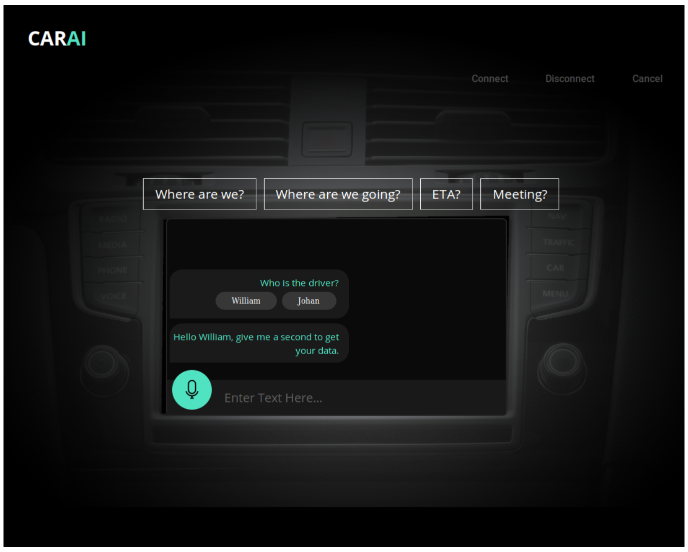


Hello William, give me a second to get your data.

Where are we?

We are at Geijersgatan 1.

Figure 4.7: A screenshot taken of the prototype, which displays the initialization of the system with the greeting, followed up by a question of where we are, which the system answers to.



Who is the driver?

Hello William, give me a second to get your data.

Figure 4.8: A screenshot taken of the prototype, which displays the initialization of the system. In this example the detection of the driver was uncertain and needed to be verified, the user used a button press to verify himself as William.

5

Experimental Setup

In this chapter, relevant information for the testing of the prototype will be detailed. Firstly a simpler comparison method will be described, which is used to compare the results of the neural networks prediction capabilities. After that follows a description of the datasets that are used in the testing. Lastly an overlook of what type of computer systems where used in the testing of the prototype is described.

5.1 Prediction Baseline

To have a point of comparison for the neural network implementations, a naive implementation was made: the Predictor Graph (PredictorG).

The graph used in predictorG is an unidirectional graph where a node represents a place and an edge represents a trip made from one node to the other. Each edge also keeps track of when the trip took place, with the date represented as the time of day, day of the week, and month of the year.

By feeding the algorithm paths consisting of a start and end node, as well as time information (currently time of day (t), day of week (d), month of year (m)), the algorithm constructs the graph mentioned above.

For the prediction step the day of week and month of the year are considered as Boolean features, either the day/month matches or it does not. While the time of day can match more or less depending on how close the value is.

Given a constructed graph the algorithm can predict an end node from a given node and time. The prediction is done by summing all the paths that are close in time and are tied for the most Boolean features, into their corresponding destination nodes. The sums are weighted sums over the time of day weighted negatively depending on the difference in relation to the inputted time of day. The largest of the sums is the predicted next node.

The following is a representation of a single node labeled A, this node has connections to node B every weekday with time variations between 8:00 and 8:45. Node A also has connections with node C at Tuesday at 10:00 and at Sunday at 8:00. All edges in this example have the same month value.

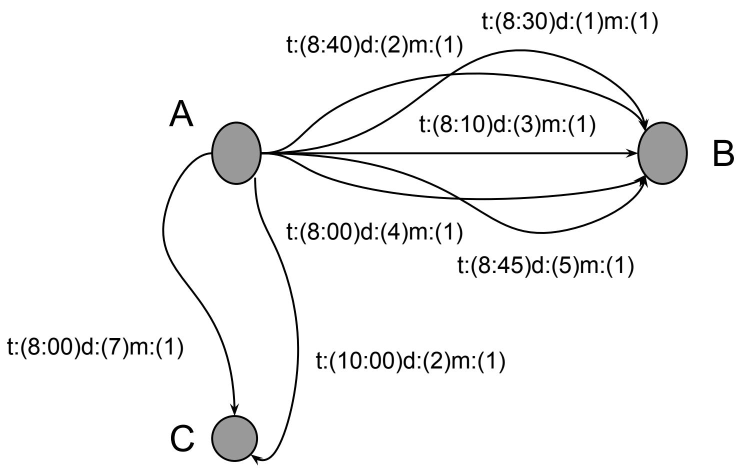


Figure 5.1: Visualization of a part of a graph produced by predictorG, A, B and C are nodes in the graph corresponding to locations. In this example each edge contains time, day of the weak and moth of the year. This picture shows only the paths from A so that the examples in this section becomes easier to follow.

For example if a prediction from node A in the above figure was made at 9:10 on a Tuesday. The algorithm would consider C:t:(10:00)d:(2)m:(1) and B:t(8:40)d:(2)m:(1) as viable candidates as they both have three matching Boolean features while the others have less. With the weight function 1/(1+time difference) in this example. And the "close" in time being +-50 min. Then the sums should be made as follows:

$$8:40 = 520min$$
$$9:10 = 550min$$
$$10:00 = 600min$$
$$B: \frac{1}{(1+abs(550-520))} = \frac{1}{(1+30)} = \frac{1}{31}$$
$$C: \frac{1}{(1+abs(550-600))} = \frac{1}{(1+50)} = \frac{1}{51}$$

Therefor B is the predicted node as $\frac{1}{31} > \frac{1}{51}$

5.2 Dataset

Three types of data sets were used to test the network's prediction capabilities. The three sets where the Geolife Dataset, a large collection of users; a generated data set, with predefined behaviours; and a personal dataset, gathered from location data passively gathered from a smartphone.

5.2.1 Geolife Dataset

The Geolife Dataset is a dataset made by Microsoft Research Asia, detailing moment of civilians mostly in the Shanghai area, it contains 182 users collected over a period of five years. The dataset consist of samples containing longitude, latitude, altitude, date and time, most is densely sampled data, 97.5%, meaning that its sampled at

1-5 second intervals or 5-10 meters. 73 out of the 182 users has also labeled the gathered data with method of transport, but these special cases was not utilized for the testing. [Zheng et al., 2008, 2009, 2010]

5.2.2 Generated Dataset

Judging the accuracy of predictions from a real life dataset proved difficult, as the intent of the sampled individuals is unknown. Therefore most predictions could seem plausible, to fix this problem it was decided that an artificial dataset would be generated, where what is likely, unlikely, and impossible is known respectively.
The dataset was constructed by stepping through a time period, 1992/11/19 to 2020/4/6, picking paths associated with each day. The paths chosen was based on a weighted random distribution, the paths and the corresponding weights can be seen in the table below. The Path column shows paths represented as *important locations*, L_1-L_{21}, and transitions indicated by arrows with a time of day indicated in minutes; The time given represents the center of a Gaussian distribution with a variance of 20 minutes. The Weight column details the weight of the path that ultimately determines the frequency a path would be picked in that day of the week. Below follows two of the seven tables representing the possible paths taken on that weekday.

Monday

Weight	Path
1	L_1 - 780 min $\rightarrow$ L_9 - 880 min $\rightarrow$ L_1
1	L_1 - 780 min $\rightarrow$ L_{10} - 880 min $\rightarrow$ L_1
40	L_1 - 480 min $\rightarrow$ L_2 - 690 min $\rightarrow$ L_3 - 750 min $\rightarrow$ L_2 - 1020 min $\rightarrow$ L_1
60	L_1 - 480 min $\rightarrow$ L_2 - 690 min $\rightarrow$ L_4 - 750 min $\rightarrow$ L_2 - 1020 min $\rightarrow$ L_1
10	L_1 -480 min $\rightarrow$ L_2 - 690 min$\rightarrow$ L_3 - 700 min $\rightarrow$ L_2 - 1020 min $\rightarrow$ L_9 - 1100 min $\rightarrow$ L_1
10	L_1 - 480 min $\rightarrow$ L_2 - 690 min $\rightarrow$ L_4 - 700 min $\rightarrow$ L_2 - 1020 $\rightarrow$ L_9 - 1100 min $\rightarrow$ L_1
18	L_1 -480 min $\rightarrow$ L_2 - 690 min $\rightarrow$ L_3 -750 min $\rightarrow$ L_2 - 1020 min $\rightarrow$ $L_1$0 - 1100 min $\rightarrow$ L_1
20	L_1-480 min $\rightarrow$ L_2 - 690 min $\rightarrow$ L_4 - 750 min $\rightarrow$ L_2 - 1020 min $\rightarrow$ $L_1$0 - 1100 min $\rightarrow$ L_1
6	L_1-480 min $\rightarrow$ L_2 - 690 min $\rightarrow$ L_3 - 700 min $\rightarrow$ L_9 - 740 min $\rightarrow$ L_2 - 1020 min $\rightarrow$ L_1
2	L_1-480 min $\rightarrow$ L_2 - 690 min $\rightarrow$ L_4 - 700 min $\rightarrow$ L_9 - 740 min $\rightarrow$ L_2 - 1020 min $\rightarrow$ L_1
4	L_1-480 min $\rightarrow$ L_2 - 690 min $\rightarrow$ L_3 - 700 min $\rightarrow$ L_{10} - 740 min $\rightarrow$ L_2 - 1020 min $\rightarrow$ L_1
8	L_1-480 min $\rightarrow$ L_2 - 690 min $\rightarrow$ L_4 - 700 min $\rightarrow$ L_{10} - 740 min $\rightarrow$ L_2 - 1020 min $\rightarrow$ L_1
2	L_1 - 480 min $\rightarrow$ L_2 - 690 min $\rightarrow$ L_3 - 765 min $\rightarrow$ L_1
2	L_1 - 480 min $\rightarrow$ L_2 - 690 min $\rightarrow$ L_4 - 765 min $\rightarrow$ L_1
1	L_1 - 480 min $\rightarrow$ L_2 - 690 min $\rightarrow$ L_3 - 765 min $\rightarrow$ L_9 - 810 min $\rightarrow$ L_1
1	L_1 - 480 min $\rightarrow$ L_2 - 690 min $\rightarrow$ L_4 - 765 min $\rightarrow$ L_9 - 810 min $\rightarrow$ L_1
1	L_1 - 480 min $\rightarrow$ L_2 - 690 min $\rightarrow$ L_3 - 765 min $\rightarrow$ L_{10} - 810 min $\rightarrow$ L_1
1	L_1 - 480 min $\rightarrow$ L_2 - 690 min $\rightarrow$ L_4 - 765 min $\rightarrow$ L_{10} - 810 min $\rightarrow$ L_1

Table 5.1: This table details the paths that can be taken on a Monday, in the fabricated data test. Weight indicates the frequency at which a path is picked, the minutes indicate the top of a Gaussian distribution with variance of 20 min and the L_X indicate the *important location* corresponding to cluster X.

Sunday

Weight	Path
40	L_1 - 700 min → L_9 - 990 min → L_1
20	L_1 - 200 min → L_9 - 990 min → L_{10} - 1040 min → L_1
30	L_1 - 222 min → L_{10} - 300 min → L_{13} - 700 → L_1
30	L_1 - 400 min → L_{13} - 500 min → L_{14} - 600 min → L_{15} - 700 min → L_1
40	L_1 - 500 min → L_{14} - 660 min → L_5 - 760 min → L_{14} - 900 min → L_1
40	L_1 - 500 min → L_{20} - 600 min → L_{19} - 800 min → L_{21} - 1000 min → L_1

Table 5.2: This table details the paths that can be taken on a Saturday, in the fabricated data test. Weight indicates the frequency at which a path is picked, the minutes indicate the top of a Gaussian distribution with variance of 20 min and the L_X indicate the *important location* corresponding to cluster X.

5.2.3 Personal Dataset

The personal dataset was derived from Google Locations, which is passively gathered data from Android Devices. The resulting data was about 3 years of personal traveling information from one of the thesis writers. The conditions for this gathered data is the most similar type of data, the project expects to receive as training data for an end user application.

5.3 Experimental: Computer Setup

The system will be tested on three computer setups: Raspberry Pi, ASUS Laptop and “The Dragon Computer”. The system hardware of the devices are the following:

Raspberry Pi:

A Standard Raspberry PI 2 Model B

- CPU: A 900MHz quad-core ARM Cortex-A
- RAM: 1GB
- OS: Raspbian
- Camera: Raspberry PI board camera

Notes: Tests are run with the graphical interface active, which will have an impact on performance.

ASUS Laptop:

ASUS UX302L

- CPU: Intel Core(™) I7-4500U Max Clock at 2.4 GHz Standard clock 770MHz
- RAM:8 GB
- OS: Windows 10
- Camera: Built-in Camera

Notes: Tests are run with Windows performance mode enabled.

The Dragon Computer:

MSI Dragon Nightblade

- Intel Core(™) I5-4460, Max Clock at 3.2GHz
- RAM: 8GB
- OS: Ubuntu 12.04
- Camera: Logitech C270

6
Results

The following chapter will display the academic results of the project. Starting the face recognition, followed by location prediction, ending with the dialogue interaction.

6.1 Face Recognition

The following section will display the results of the testing of the face recognition module. The tests were run on the “Dragon Computer” unless otherwise specified. The Face recognition results are divided into, Lighting conditions, confidence and performance tests.

6.1.1 Lighting Conditions

All the face matching methods mentioned in the theory 2.3 are intolerant to non-optimal lighting conditions. Automatic histogram equalization is performed to flatten the extremes within the image, but results are varying. The camera plays a big role in creating a good enough image as a baseline for image processing. Several cameras were tested, with different combinations of hardware. Beside the C270, a Microsoft Lifecam Cinema Webcam was tested for comparison on the same setup. The four images in figure 6.1 where taken in the same position with the same passive lighting and post processing techniques. The image quality varied greatly depending on what camera was used, the C270 provided the highest contrast while the ASUS laptop camera resulted with a very dark image. The other two are the most similar and are in between the C270 and ASUS laptop in terms of contrast.

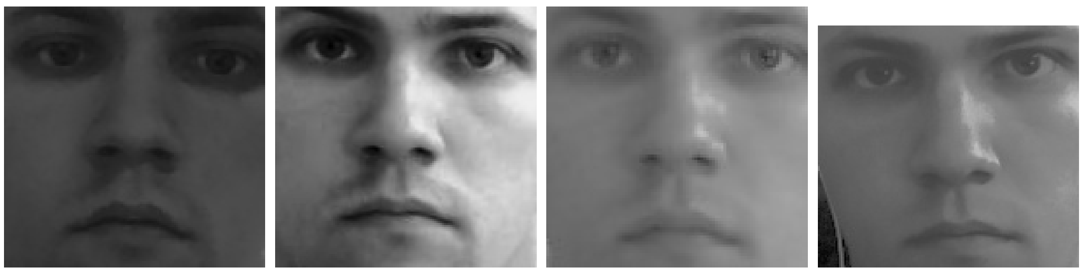


Figure 6.1: A comparison between face images of a ASUS Laptop camera to the left, in the left center is the C270 Logitech camera and to the right center is the Microsoft Lifecam Cinema. Finally to the right is the Raspberry Pi Camera.

6.1.2 Confidence

The cascade classifier provided by OpenCV accurately detects faces up to 5 meters, with a minimum distance of 0.2 meters. The accuracy of the matching is dependent on what distance the reference images are taken at. This means if all reference images are taken from an interval of [0-1.5] meters it will result in a lower confidence when compared to an image taken from [2-3.5] meters. The confidence produced by OpenCV is a distance (error) to closest matched image, this threshold is remapped to a confidence value between [0.6-1] to be compatible to Talkamatics confidence levels. This forces Talkamatic to always make sure that it knows who are in the car, if they are detected.

6.1.3 Performance Tests

Execution time of Sample method (milliseconds)

	Raspberry Pi	ASUS Laptop	Dragon Computer
	7298	10392	4444 (1588)
	6583	10020	4263 (1544)
	7417	9216	4291 (1519)
	7314	8605	4214 (1555)
	7250	7599	4224 (1538)
Average:	7172.4	9166.4	4287.2 (1548.8)

Table 6.1: Table shows the testing of execution time on each of the computer setups. The tests were run back to back without restarting the Java virtual machine. Note the parenthesis values on the Dragon Computer are execution time with the Microsoft Lifecam. The C270 prints an error message each frame slowing down the total execution time.

Maximum Performance load of Java Process (%)

	Raspberry Pi	ASUS Laptop	Dragon Computer
	75%	25%	22%
	73%	20%	23%
	80%	18%	23%
	76%	22%	17%
	77%	14%	17%
Average:	76%	20%	20%

Table 6.2: Table showing the results from testing of how much impact the face recognition module on the systems. The percentage displays the maximum load the function performed on each of the systems.

6.2 Location Prediction

The testing of the location prediction module consists of the testing of the separate subprocedures: training data culling, network prediction statistics, and prediction examples.

6.2.1 Culling results

To filter out the relevant data from the continuously sampled datasets two methods were tested. The first method is distance based culling and the second is a speed based culling. There were two test, the first was the total distance between points before and after culling with the two different techniques. This was done to get an approximation of how much information was lost. The second test was to find out how many points were removed, looking for a compression factor. The amount of *important locations*, clusters, that was found by using DBSCAN after the culling was also recorded.
The parameters for the test is the following: DBSCAN is using an Epsilon of 200m with a Minimal amount of points of 3. Speed culling uses a threshold 25.0 and distance culling uses a threshold of 25 meters with 10 points as a minimum amount of points.
The primary test were performed on the entire GeoLife dataset of 183 users. The following information could be gathered:

	Before culling	After Speed culling (%)	After Distance culling (%)
Entries per user (average)	136584.1	236.46 (0.17%)	865.83 (0.63%)
Average distance traveled per user	7102.57 km	5690.36 km (80.11%)	9360.72 km (131.79%)
Completely culled users		4 (2.2%)	25 (13.81%)
Average amount of clusters		18.45	39.83
Amount of users that have at least 1 cluster (%)		52.48%	70.71%
Amount of users with no clusters		20	26
Amount of users where clustering resulted in only noise points		67	28

Table 6.3: Table showing the statistical data gathered from running the culling algortihms on the GeoLife Dataset. The percentage value in the first two rows are in relation to *before culling*. Percentage value in row three is in relation to the total number of users (183).

The same tests were performed on the Personal Datasets where correctness could be verified by the owner of the data.

Personal Data Set 1	Before culling	After Speed culling (%)	After Distance culling (%)
Total Distance	739912,2 km	86167,7 km (11.645%)	12583,6 km (1.7%)
Amount of samples	646386	8081 (1.25%)	1369 (0.21%)
Amount of Clusters:		80	23
Personal Data Set 2			
Total Distance	16092km	7290,9 km (45.307%)	4120,7 km (25.607%)
Amount of samples	193250	1197 (0.619%)	869 (0.44%)
Amount of Clusters:		41	12

Table 6.4: Table showing the statistical data gathered from running the culling algorithms on the Personal Datasets. The percentage values are in relation to *before culling.*

When applying one of the culling algorithm to a users data, the ideal outcome would be removing a high percentage of points, while keeping the traversed distance almost the same with just a small decrease from converting the real paths to beelines. When looking at table 6.3 we see that *speed culling* reduced the data with a compression ratio of 577.62, meaning that only 0.17% remained, while 80.11% of the total distance was preserved. These numbers seems reasonable in terms of the compression and distance reduction, but it should be noted that when *speed culling* was under visual inspection using the personal data sets, it seamed to identify important locations poorly. *Distance culling* yielded a compression factor of 157.75, meaning that 0.63% remained, while the distance actually got inflated by 31.79%. The inflation of distance came from around a third of the users and is most probably because the *distance culling* algorithms tends to move points. This tendency was observed in the personal data set aswell, but there it was very minor. Movement of important locations can not be confirmed in the Geo-life dataset as there actual important locations are not known. The result of running the culling algorithms on the personal datasets can be seen in table 6.4, what is most noteworthy is the relatively high reduction in distance, but as mentioned previously in visual inspections the most important locations where present, further discussion of this matter can be red in the discussion, section 7.1.1.

6.2.2 Network types

There were three network configurations that were tried in the pursuit of a network suitable for predicting the destination of a trip. These networks where a network that used GPS coordinates (Network 1) and two networks that used *important locations* as an abstraction of the GPS coordinates (Network 2 & 3). Network 1 was the first type of network that was considered, as it seemed the simplest solution. Unfortunately it did not perform well, as the network tended to either converge all predictions to a single point in the center of the dataset or over/undershoot predictions with unacceptable margins of several kilometers.

6.2.3 GeoLife test

To test how well the networks, accompanied by *Distance culling*, performed on a larger set of real user data, that was not one of the two datasets that was used during development, the GeoLife dataset was used. See description in section 5.2.1. The test consisted of training each network for each user from the Geolife dataset, logging the training and validation error for each combination of user and network. If a user had to few entries to train all the networks, that user was discarded. This process 92 out of total of 184 users meaning that about 48% of users had too few entries to train the most demanding network, Network 3. Training error is what degree of error the network has when predicting the training set. The validation error is the degree of error when predicting the validation data. The validation data was thirty percent of the training data that was held back for validation from the original training set.

Network 1	
Training Error	Validation Error
1,826%	1,9%

Table 6.5: This table shows the averages of training error and validation error, for 92 neural networks of type Network 1 that was trained with 92 users from the GeoLife dataset, see section 5.2.1.

Network 2	
Training Error	Validation Error
1,4%	2,6%

Table 6.6: This table shows the averages of training error and validation error, for 92 neural networks of type Network 2 that was trained with 92 users from the GeoLife dataset, see section 5.2.1.

Network 3	
Training Error	Validation Error
10,3%	19,67%

Table 6.7: This table shows the averages of training error and validation error, for 92 neural networks of type Network 3 that was trained with 92 users from the GeoLife dataset, see section 5.2.1.

Training error difference		
1 vs 2	1 vs 3	2 vs 3
0,4%	−8,477%	−8.9%

Table 6.8: This table showcases the difference between the training errors of the tables 6.5, 6.6 and 6.7.

Validation error difference		
1 vs 2	1 vs 3	2 vs 3
−0.69%	−17,768%	−17.07%

Table 6.9: This table showcases the difference between the validation errors of the tables 6.5, 6.6 and 6.7.

6.2.4 Generated Data Test

To determine the neural network's ability to capture patterns corresponding to our expected input, a test was devised using the generated data from section 5.2.2. The test consisted of feeding the network with the generated data, which consists of 30830 entries, and then activating the network for every reasonable input. 30% of the entries were held back for validation, the entries held back for validation were selected randomly. The output was logged as ranges of input that resulted in a certain output, this was done for two of the three different network configurations as Network 1 can not utilize *important location* directly.

6.2.4.1 Network 2

Network 2 has the inputs of current *important location*, *day of week*, and time of day. Its output is the predicted destination *important location*.

Network 2 Error report	
Training Error	Validation Error
6.8%	6.57%

Table 6.10: This table shows the training and validation error for Network 2, training and validation error is explained in section 6.2.3

The following is an example of the Monday mapping, *Day of Week*: 1, of going from L_1. The ranges are shown as the start each new output, *important location*, and continue until a new output is outputted. For example *important location* 2 was the output between 00:00 and 12:39 in table 6.11.

L_1			
Day of Week	Hour	Minute	Important location
1	0	0	L_2
1	12	40	L_{13}
1	12	59	L_{20}
1	15	46	L_9

Table 6.11: This table shows the Monday, *Day of Week* is 1, output from the *Network 2* when the inputted *important location* is set to L_1. Each row shows the start of the time interval which produces the output, *important location*, and produces the same output until the start of the next interval.

Compared with the table 5.1, it can be seen that the most common destination taken Monday morning, *important location* 2, is indeed predicted around that time. A more interesting example is Saturday and Sunday as there are a greater variance in destinations based on time. Here follows the output for Saturday *important location* 1:

L_1			
Day of Week	Hour	Minute	*Important location*
6	0	0	L_9
6	2	32	L_{10}
6	4	46	L_{13}
6	7	45	L_{14}
6	8	1	L_{20}
6	9	41	L_9
6	14	0	L_1

Table 6.12: This table shows the Saturday, *Day of Week* is 6, output from the *Network 2* when the inputted *important location* is set to L_1. Each row shows the start of the time interval which produces the output, *important location*, and produces the same output until the start of the next interval.

Compared to the table 5.2, it can be seen that although L_9, L_{20} and L_1 are garbage results, L_{10}, L_{13} and L_{20} have been categorized in reasonable time intervals. The garbage result are probably a result of not having any training data at those time intervals.

6.2.4.2 Network 3

The following is the Monday mapping, *day of the week*: 1, of going from L_2 having been at L_1. The ranges are shown as the start each new output, *important location*, and continue until a new output is outputted. For example *important location* 10 was the output between 00:00 and 04:51.

Network 3 Error report	
Training Error	Validation Error
6.3%	6.17%

Table 6.13: This table shows the training and validation error for Network 3, training and validation error is explained in section 6.2.3

The following is the Monday mapping, *day of the week*: 1, of going from L_2 having been at L_1. The ranges are shown as the start each new output, *important location*, and continue until a new output is outputted. For example *important location* 10 was the output between 00:00 and 04:51.

$L_1 \rightarrow L_2$			
Day of Week	Hour	Minute	Important location
1	0	0	L_{10}
1	4	52	L_2
1	6	37	L_{20}
1	6	43	L_9
1	13	55	L_{14}
1	22	34	L_1

Table 6.14: This table shows the Monday, *Day of Week* is 1 , output from the Network 3 when the current *important location* is set to L_2 and the previous *important location* is set to L_1. Each row shows the start of the time interval which produces the output, *important location*, and produces the same output until the start of the next interval.

Comparing the resulting input output mapping to the expected patterns, showed that the network had utterly failed to capture even the most prominent of patterns from the generated dataset. For example, in the generated data: a Monday starts in 98.94% of cases with a trip from L_1 to L_2, and those cases are followed by a trip to L_4 or L_5 in 100% of cases, see table 5.1. But when looking at the input output mapping of the neural network, Monday leaving L_2 with L_1 as a previous location, ranging over all time inputs none of them resulted in either L_4 or L_5. What is stranger still is that they are mapped to *important locations* that have never been accessed from L_2.

6.2.4.3 PredictorG

PredictorG is the algorithm that was to serve as a baseline to which the networks could be compared to, it is described in section 5.1. The binary features were limited to *day of the week* and to make the data more readable it was also set as a required match. To clarify, this network has the inputs of current *important location*, *day of the week* and *time of day.* Its output is the predicted destination *important location.* The following is the Monday mapping, *day of week*: 1, of going from L_1. The ranges are shown as the start each new output, *important location*, and continue until a new output is outputted

L_1			
Day of Week	Hours	Minute	Cluster
1	0	0	L_{-1}
1	5	57	L_2
1	10	8	L_{-1}
1	11	32	L_9
1	11	39	L_{10}
1	11	41	L_9
1	12	4	L_{10}
1	12	9	L_9
1	13	30	L_{10}
1	13	39	L_9
1	13	51	L_{10}
1	14	1	L_{-1}

Table 6.15: This table shows the Monday, *Day of Week* is 1 , output from the PredictorG when the current *important location* is set to L_1. Each row shows the start of the time interval which produces the output, *important location*, and produces the same output until the start of the next interval.

The most notable difference compared with the neural network methods is that the output becomes jittery when there are two approximately equal current destination *important locations*. Also as can be seen at table 6.15, entry 2 and 11 has a *important location* labeled with -1, as there are no entries close to that input time.

PredictorG Error report	
Training Error	Validation Error
–	14%

Table 6.16: This table shows the validation error for PredictorG, Training error is not applicable as the training of PredictorG consists of just adding paths see 5.1.

6.2.5 Dialog Interaction

The dialog interaction provided a functioning interface, whose primary functionality is to let the system to acquire information from the user. But also provides basic command style request from the user to be processed. The system can:

- Ask and acquire the name of unidentified individuals in the car.
- Ask and clarify uncertainty of a poorly identified user.
- Ask and clarify uncertain location predictions.

The user can:

- Set predefined settings with exact voice commands.
- Acquire status information with exact voice commands.
- Use touch buttons for quick access to commands and answers to confirmation questions.

7 Discussion

In this chapter we will discuss some of the findings and results of the the tests, thereafter possible future work will be suggested. The chapter ends with an ethics discussion and a conclusion.

7.1 Discussion of the Results

In this section the most prominent results are discussed, starting with the culling algorithms followed by a lengthy section discussing the results of location prediction. After that follows a short mention of our thoughts on the selection of Talkamatic for this project, thereafter the section will be concluded with the discussion of Face recognition and hardware requirements.

7.1.1 Culling

We found that culling with Distance culling or Speed culling, section 4.2.3.2, provided great compression of the data, and it also made finding *points of interest* easier. In addition the culling solved the issue of having multiple successive samples in the same location, that could have lead to predicting movement to the current location. Distance culling was deemed the superior of the two culling methods as although it tends to move points slightly, it does not place points along paths traveled like the Speed culling tended to do. Both are very rudimentary using the same parameters for all datasets, returning a rough estimate rather than a complete map of all points of interest. However combined with clustering the results of the culling methods seemed quite good. An odd consequence is that the statistics produced in 6.2.1 indicate that *speed culling* would be the best method, although on visual inspection, with knowledge of where the important locations should be, indicates that *distance culling* is superior. This might indicate that *distance culling* works well on some data but fails when exposed to the more varying data of the Geo-Life dataset.

7.1.2 Location Prediction

The idea behind Network 1 was to let the neural network do the abstraction, in a sense this was the simplest approach as the neural network would do all of the work. The idea behind Network 2 was that limiting the number of possible outcomes would make it easier to pick the correct answer, in addition to not being able to make absurd predictions, like the center of the Atlantic ocean. Network 3 was essentially just providing Network 2 with more information, namely the previous location.

Network 1 was proved to be unreliable as although it had the lowest validation error, it never seemed to actually predict reasonable points. It quite often converged the points to the geographical center of the set, and when it appeared to get the direction right it either over or undershoot the distance by several kilometers. Out of the three network configurations, Network 2 performed the best as it had a lower validation error than Network 3 in the GeoLife dataset test 5.2.1 by 17.074% and did not have the troubles of Network 1. When it comes to Network 3 there is obviously something wrong as it has such a high validation error. We suspect that the network is too small, as the *important locations*'s input are represented as one hot and there are two input *important locations*. Which means that there are two times the number of clusters found plus eight, *day of week* and *time of day*, going through three hidden layers whose size are decided by Encog. Using the Generated data on Network 3 yielded, 48 input nodes going through hidden layers of 49, 8 and 14 nodes respectively. We suspect that the second layer, in this case, is too small and might be the cause for the relatively high training and validation error of Network 3 in the test, see table 6.13.

When comparing with the baseline, predictorG, only Network 2 seemed relevant as it uses the same inputs and was the network structure that preformed the best. Network 2 had a better validation error of 6.57%, see table 6.6 against that of predictorG who had 14% in the generated data test, see table 6.16, but even if this discrepancy in validation error is ignored, Network 2 would still be a better solution. Another advantage of using Network 2, over predictorG is that it does not produce the same type of noise, when encountering two almost equally relevant prediction possibilities, Instead one of the predictions dominates the time slot, see table 6.11. The negative aspects of Network 2 compared with predictorG is that in time slots where no data is available, there is still a prediction made which is often wrong with no connection to that day or time, for example c_9, c_{20} and c_1, see table 6.12. However the issue with producing odd results where there is no available data is not a big problem, as when dealing with real data, samples will cover much more of the day. Additionally if the samples do not cover some part of a day, it is unlikely that that user will be using the car during those time intervals. The jittery response of predictorG is more of a problem, as it may be experienced as random or erratic behavior by a potential user. Thus we conclude that, at least in our case, the layer of abstraction that neural networks provide, gives an advantage compared to rule based solutions, as simple rules can not in a good way capture the driving patterns and making the rules more complex will radically increase the complexity of the algorithm.

Using a general dataset is a usual technique to initiate a network. In this case, using the the Networks 2 or 3 as an example, they use a classifier structure. Where the input size is a one-to-one representation of how many *important locations* there are. The amount of *important location*s found differs greatly between each user, which the GeoLife dataset and personal datasets showed. These ranges could vary from 3 clusters for someone who recently has moved to the area or up towards 100 and the significance of each cluster is also very user dependant. Without a deeper analysis of the data, finding places such as home and work, there will be no real common patterns between users. As the identifying number of a cluster, that is to say what *important location ID* it will receive, will most likely not be attributed to the same type of location between users. Therefore we believe that a general training set would, in the current state at best give no significant improvement and at worst skew the results of the majority of predictions.

Multiuser predictions had its implementation started, but was not integrated with the system. At the time of giving up on the feature the following functionality was implemented: First it used the normal location prediction for all users. Then it looked at the upcoming appointments of the users, the appointments replaced the predicted location, if it was applicable for this travel and added a time constraint for that user. Then it used Google to query the expected time between all the users desired locations, as well as the current location of the car. After that all possible paths are iterated through with the constraint that the drivers appointment is never missed. Lastly the trips that delivered the most passengers on time, are compared according to how many additional drop-offs they managed and lastly the minimum amount of time taken. It can be seen in this explanation there are a lot of assumptions made about what is important and this is the main reason way the feature was not implemented. As the current priorities where implemented as a placeholder and finding a list of priorities that where justifiable proved harder than initially expected. In addition to this there would not have been time to conduct or construct proper tests, nor would there be time to integrate the multiuser predictions with the rest of the prototypes modules.

7.1.3 Talkamatic

Talkamatic provided an interesting platform for dialogue management. It was probably not the easiest platform to choose for a project like this, but provided a good interface, with which the confidence values used throughout the other modules could be integrated. Hopefully our feedback helped their platform to become better.

7.1.4 Face Recognition

There were two options for the implementation of the automatic profile selection. Either the face recognition would run constantly, or the face recognition would be run when users entered and left the car. There is no real benefit to have face recognition running constantly, as the internal passenger state does not change when the car is in motion. The only possible advantage might be if an initial face detection fails and then succeeds during the trip, data could be logged on the appropriate user. But assuming that the face recognition fails defeats the point of having the system recognise the user, as the user sits down in the car.

7.1.5 Hardware Requirements

On the subject of hardware, there was problems when trying to install the entirety of the client onto the Raspberry Pi 2 model B, as Talkamatic did not have an ARM build at the time of making this project. However the hardware should be able to run the application performance wise, as long as face recognition is not run continuously. Our reasoning behind this statement is that the the Talkamatic client ran without any problems on a computer that had a lower clock frequency than the Raspberry Pi 2 model B. But the face recognition took 70% of the processing power of the Raspberry Pi 2 model B when we tested it on the system, which would most lightly cause sluggish behavior, if it ran continuously alongside the Talkamatic client. In any case there should exist a NUC unit that can run the entirety of the prototype, as NUCs can be much more powerful and have the standard x86/x64 processor architecture.

7.2 Future Work

For further development it should be noted that it is easier to remove and reroute false positives, than create new points of interest in place of missing ones. As adding a point provides no context of how the point is connected to the rest of the paths.

Currently the networks are automatically constructed based on the input data given to Encog, future work should examine if a better model for deciding the network structures can be achieved. To a similar end the amount of *important location*, clusters, could be culled by importance, as *important location* with low probability are unlikely to be predicted. A combination of the two will most likely be the best approach in making a solution that is more tolerant in regards to varied training data.

Using or developing a more accurate culling algorithm could possibly improve the initialization of the networks and should be considered as a possibe significant improvement.

An actual implementation would require an integration with the cars internal CAN network, this was firstly considered a part of this project but was excluded due to time constraints.

7.3 Ethics

The large amount of private data processed by the suggested algorithms, poses a lot of privacy concerns. In addition to this is the fact that most of the data must be processed externally on a server, which evokes trust issues. It is clear that this type of solution has to be accompanied by agreements and regulations, to remain ethical.

An application like this will add a new layer of logging personal data on top of already existing tracking services. This is something that has to be emphasized if this is to be delivered as a commercial product. Having a machine provide a location suggestion based on previously traveled destinations could unravel places that are sensitive for the user. Creating a good and easy-to-use control panel so that the users can easily manage their personal data both to edit or remove logged points is essential to keep some degree of privacy.

This application is seen as a possibility to aid the driver, but depending on how the system is realized, as a commercial product, there exists possibilities that an application like this would do the opposite. Having minor bugs in the code or an non-ideal design could impair the drivers focus on the road resulting in a possible cause of harm to the car, with its occupants, or the surroundings of the car. Security must also be a much larger concern than was considered in this prototype. Having a computer, that is connected to the Internet, can be susceptible to attacks and because it connected to the cars internal systems, it could act as a conductor for hacking attacks that could either harm the occupants of the car or the car it self.

7.4 Conclusions

Not all of the planned features where implemented and some of the ones that where did not reach their full potential, but the proactive behavior desired of the IPA prototype was achieved as a combination of all the modules described in this thesis: *Automatic Profile Selection*, section 4.2.2, *Location Prediction*, section 4.2.3, and *Dialogue Manager*, section 4.2.6. The location prediction combined with the automatic profile selection makes it possible to proactively greet the user with more than just a simple hello, in a way that is both personal and performs a function. In addition, having the program asking verification questions when it has low certainty and auto resumed on high certainties, improved how intelligent the proactive behavior seamed. Summing up, this project achieved a proof of concept for a proactive IPA for a car environment.

Bibliography

Daniel Ashbrook and Thad Starner. Using gps to learn significant locations and predict movement across multiple users. *Personal and Ubiquitous Computing*, 7 (5):275–286, 2003.

AT&T. Volvo cars and at&t enter multi-year agreement to connect future models in u.s. and canada | at&t. `http://about.att.com/story/volvo_cars_and_att_enter_multi_year_agreement_to_connect_future_models_in_us_and_canada.html`, April 2014. (Accessed on 05/19/2016).

Peter N Belhumeur, João P Hespanha, and David J Kriegman. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(7):711–720, 1997.

Allan W Black. Festvox: Us slt (us female). `http://festvox.org/cmu_arctic/dbs_slt.html`, 2015. (Accessed on 05/28/2016).

CMU. Versions of decoders [cmusphinx wiki]. `http://cmusphinx.sourceforge.net/wiki/versions`, March 2012. (Accessed on 05/19/2016).

Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

EU. Protecting your data: your rights - european commission. `http://ec.europa.eu/justice/data-protection/individuals/rights/index_en.htm`, March 2016. (Accessed on 05/19/2016).

Sébastien Gambs, Marc-Olivier Killijian, and Miguel Núñez del Prado Cortez. Next place prediction using mobility markov chains. In *Proceedings of the First Workshop on Measurement, Privacy, and Mobility*, page 3. ACM, 2012.

Ben Goertzel. Opencog foundation | about. `http://opencog.org/about/`, April 2010. (Accessed on 05/19/2016).

Google. Alphago | google deepmind. `https://www.deepmind.com/alpha-go`, 2016a. (Accessed on 06/26/2016).

Google. Google calendar api | google developers. `https://developers.google.com/google-apps/calendar/`, 2016b. (Accessed on 05/20/2016).

Google. Google maps distance matrix api | google developers. `https:`

//developers.google.com/maps/documentation/distance-matrix/, 2016c. (Accessed on 05/20/2016).

Google. Getting started | google maps geocoding api | google developers. https://developers.google.com/maps/documentation/geocoding/start?hl=en_US, 2016d. (Accessed on 05/20/2016).

Johann Hauswald, Michael A Laurenzano, Yunqi Zhang, Cheng Li, Austin Rovinski, Arjun Khurana, Ronald G Dreslinski, Trevor Mudge, Vinicius Petrucci, Lingjia Tang, et al. Sirius: An open end-to-end voice and vision personal assistant and its implications for future warehouse scale computers. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 223–238. ACM, 2015.

Jeff Heaton. Encog machine learning framework. http://www.heatonresearch.com/encog/, 2016. (Accessed on 05/30/2016).

IBM. Ibm bluemix - what is bluemix. http://www.ibm.com/cloud-computing/bluemix/what-is-bluemix/, 2016a. (Accessed on 05/19/2016).

IBM. Node-red : Documentation. http://nodered.org/docs/, 2016b. (Accessed on 06/07/2016).

Intel. Mini pc: Intel® nuc. http://www.intel.com/content/www/us/en/nuc/overview.html, 2016. (Accessed on 05/19/2016).

ISO 20922:2016. Information technology – Message Queuing Telemetry Transport (MQTT) v3.1.1. Standard, International Organization for Standardization, Geneva, CH, March 2016.

Itseez. Opencv | opencv. http://opencv.org/, 1016. (Accessed on 05/19/2016).

Itseez. Android | opencv. http://opencv.org/platforms/android.html, 2016. (Accessed on 05/19/2016).

VZ Këpuska and TB Klein. A novel wake-up-word speech recognition system, wake-up-word recognition task, technology and evaluation. *Nonlinear Analysis: Theory, Methods & Applications*, 71(12):e2772–e2789, 2009.

Haowei Liu. *Face Detection and Recognition on Mobile Devices*. Elsevier, 2014.

Wesley Mathew, Ruben Raposo, and Bruno Martins. Predicting future locations with hidden markov models. In *Proceedings of the 2012 ACM conference on ubiquitous computing*, pages 911–918. ACM, 2012.

MySQL. Mysql :: Mysql 5.7 reference manual. http://dev.mysql.com/doc/refman/5.7/en/, May 2016. (Accessed on 05/19/2016).

OpenCog. Opencog framework: Opencog source code documentation. http://docs.opencog.org/opencog/index.html, May 2016. (Accessed on 05/19/2016).

PI. Power supply - raspberry pi documentation. https://www.raspberrypi.org/documentation/hardware/raspberrypi/power/README.md, 2016a. (Accessed on 05/19/2016).

PI. Raspberry pi 2 model b. `https://www.raspberrypi.org/products/raspberry-pi-2-model-b/`, 2016b. (Accessed on 05/19/2016).

Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, et al. The kaldi speech recognition toolkit. In *IEEE 2011 workshop on automatic speech recognition and understanding*, number EPFL-CONF-192584. IEEE Signal Processing Society, 2011.

ROBOTLAB. Robotlab smart useful retail and educational robots. `http://www.robotlab.com/`, 2016. (Accessed on 06/01/2016).

OpenCV Dev Team. Face recognition with opencv — opencv 2.4.13.0 documentation. `http://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_tutorial.html`, 2016. (Accessed on 05/31/2016).

Matthew Turk and Alex Pentland. Eigenfaces for recognition. *Journal of cognitive neuroscience*, 3(1):71–86, 1991.

Univeristy of Edinburgh. The festival speech synthesis system. `http://www.cstr.ed.ac.uk/projects/festival/`, 2016. (Accessed on 05/20/2016).

Yu Zheng, Quannan Li, Yukun Chen, Xing Xie, and Wei-Ying Ma. Understanding mobility based on gps data. In *Proceedings of the 10th international conference on Ubiquitous computing*, pages 312–321. ACM, 2008.

Yu Zheng, Lizhu Zhang, Xing Xie, and Wei-Ying Ma. Mining interesting locations and travel sequences from gps trajectories. In *Proceedings of the 18th international conference on World wide web*, pages 791–800. ACM, 2009.

Yu Zheng, Xing Xie, and Wei-Ying Ma. Geolife: A collaborative social networking service among user, location and trajectory. *IEEE Data Eng. Bull.*, 33(2):32–39, 2010.