Expertise
and insight
for the future

Hari Adhikari
Shurakshya Kharel

# Building a Raspberry Pi Car Safety System with Facial Recognition

Metropolia University of Applied Sciences

Bachelor of Engineering

Information technology

Bachelor’s Thesis

17 January 2019

Metropolia
University of Applied Sciences

| | |
|---|---|
| Author<br>Title<br><br>Number of Pages<br>Date | Hari Adhikari and Shurakshya Kharel<br>Building a Raspberry Pi Car Safety System with Facial Recognition<br>54 pages<br>17 January 2019 |
| Degree | Bachelor of Engineering |
| Degree Programme | Information Technology |
| Professional Major | Software Engineering |
| Instructors | Janne Salonen, Principal Lecturer |

The final year project was about developing a prototype of a car security system with cognitive services, Internet of Things (IOT) and facial recognition Api. The aim of the application was to enhance safety of the car drivers and security from the possible thefts. In addition, the application also focused on making cars personalized and creating the safe and smooth journey for the users.

The project was carried out to build smart solution with safety-first mindset. Continuous monitoring of car drivers and analyzing the response every possible minute is the key factor to ensure the car is driven in well manner. This was achieved with the integration of IOT devices and raspberry PI where backend server is continuously monitoring with the start of car. The application was developed on JavaScript based programming language. The scope of the project is limited to independent prototype of working system which is not integrated with car.

In conclusion, a functional working system was created using facial recognition and image analysis with Amazon Web Services, Open CV and different IOT devices such as PIR Motion Sensor and PI Camera Module. A robust application was built to collect data from the system and execute program on the basis of response in order to provide secure and risk-free drive experience to the user.

Hence, the face detection and emotion recognition-based system was successful to collect data to avoid possible thefts and road-accidents. Integration on car for such inbuilt-features would be huge future development.

| | |
|---|---|
| Keywords | Facial recognition, Internet of Things, Cloud Technologies, Raspberry Pi |

## Contents

## List of Abbreviations

| | |
|---|---|
| AI | Artificial Intelligence |
| API | Application Programming Interface |
| AWS | Amazon Web Services |
| CLI | Command Line Interface |
| DBMS | Database Management System |
| FBI | Federal Bureau of Investigation |
| GPIO | General Purpose Input Output |
| IOT | Internet of Things |
| JPEG | Joint Photographic Experts Group |
| M2M | Machine to Machine |
| NOOBS | New Out of the Box Software |
| NPM | Node Package Manager |
| OS | Operating System |
| PCA | Principal Component Analysis |
| PIR | Passive Infrared |
| RDBMS | Relational Database Management System |
| RDS | Relational Database Service |
| RFID | Radio Frequency Identification |

| | |
|---|---|
| SD | Secure Digital |
| SDK | Software Development Kit |
| SES | Simple Email Service |
| SMS | Short Message Service |
| SNS | Simple Notification Service |
| SQL | Structure Query Language |
| S3 | Simple Storage Service |
| UI | User Interface |
| UID | Unique Identifier |

# 1 Introduction

Facial recognition and sentiment analysis are being important aspect of research throughout the world. In this era of advancement and modernization, several technologies have enticed the world with Internet of things (IOT) based solutions integrated with machine learning and Artificial Intelligence (AI). The benefits of these AI and IOT based solutions are limitless. Likewise, image processing and video analysis have broad scope in the modern digital world. Live streaming video content analysis, image processing, object detection and facial recognition, facial identity security systems, website content analysis, content validation and sentiment analysis are some of the examples that world would benefit with this technology.

At present, every device and appliance are increasingly focused to ensure quality service and safety for the human needs. Although safety and quality service with smooth user experience is being a priority in most of the industries, the statistics of car driver's safety and auto theft indicates otherwise. According to Statistics Finland, statistics on road traffic accidents, 517 road traffic accidents were recorded between January and February, 2019 in Finland claiming 27 lives and 690 persons injured [1]. Moreover, U.S Department of Justice, Federal Bureau of Investigation (FBI) has stated that 773,139 vehicles were stolen in 2017 which lead to $6 billion amount. The vehicle theft was reported to police on every 40.9 seconds. [2.] Whether the reason is lack of proper security and monitoring systems for vehicle theft or sleep deprived and fatigue for road accidents, it is required to minimize the casualties. Hence, the proposed thesis will focus on developing an IOT based solution utilizing facial recognition and detection to overcome the short comings of present systems.

Thus, the aim of the final year project was to build a prototype system which would enhance the car and driver safety with face identification and image analysis for drowsiness. The project can be used as an independent identification and security system on cars or any other vehicles. The sentiment analysis of the car drivers and its visual representation will also provide a way for personalization of the cars. The implementation was limited to the creation of functional raspberry Pi device. Integration of system on car as inbuilt features can be the future prospect and area of research as well.

# 2 Theoretical Background

This chapter gives the detailed review of the technologies used in the project and elucidates suitability for using the technologies. It also analyses and compares several related technologies and their aspects for the project.

The chapter focuses mainly on elaboration of the technical aspects, starting from Facial recognition and its method and classification. The section further explores the libraries and cloud technologies which uses various methods of facial recognition to provide accurate data of facial detection, emotion analysis and classification. Moreover, the chapter illustrates the communication between the device and techniques of IOT as well.

The latter part of the section explores the available stacks and collates their compatibility with the project requirements. It also provides the reasons for choices of the environments and solutions included in the project.

## 2.1 Face Recognition

Face recognition is a process to uniquely identify or verify a person, face or image by mapping individual facial features through various methods. Facial recognition, also known as Face recognition is non-intrusive method which is widely used in biometrics and pattern recognition [3, 835]. It has wide scope on areas like access controls, Identification systems and surveillances. It works broadly under two applications: Face Verification and Face Identification. One to one match between the query face image and template face image is performed in Face Verification whereas one to many matches between the query face image and template image and finding similarity between them is done in Face Identification method. [4, 1-3.]

Face recognition is a step-wise procedure comprising of several individual stages working together to make a fully automated system, that is, consisting of both Face verification and identification methods. The steps involved in face recognition are shown in figure 1.

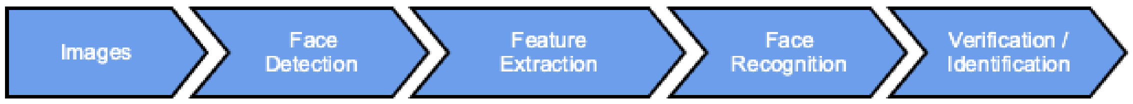


Figure 1. Face Recognition processing flow. Reproduced from Dhawan et al. [3, 837]

As shown in Figure 1, the initial step is capturing face either being a face or an image. When the face is captured, the algorithm focused on front of the human face for detecting presence of any face is used for face detection whereas in image detection, image is matched bit by bit and stored in database. The feature extraction step is to define a set of data which will represent meaningful information for classification and analysis. It reduces the amount of data without losing the important data attributes. After the feature extraction, the identity and similarity of query face with a set of data is matched with template images stored in database. [3, 836; 5, 228-229.] The final stage is the application for using it either as verification or identification.

Face recognition is progressing subject, transforming and progressing constantly. It has various approaches and techniques. The sub-section explains various techniques used for face recognition.

### 2.1.1 EigenFace

The method mainly depends on the statistics set to describe face features. The method uses principal component analysis (PCA) to represent the image with the variation of facial features like expression and lighting. [6, 90.] The purpose is to store the main components which grabs the most variance and minimize the complexity to two dimensions [3, 838]. With the given image, first the eigenvectors and eigenvalues are taken out to calculate Eigenfaces. A face then can be represented by a linear combination of these eigenfaces and can be reconstructed from these low dimension, that is, two dimensions.

### 2.1.2 Graph Matching

The algorithm is used for identifying object or object class in an image, based on the constructed rectangular image graph of a face using a set of Gabor filters. In this system, faces are represented with the collection of node making a rectangular graph. [7, 775.] The node of the graph is labeled with set of complex Gabor wavelet coefficients, known as jet [8, 2-3]. The magnitude of jets is used for comparisons and recognition. In other terms, straight forward matching of graphs is done for face recognition. First, jets are used to determine accurate location of nodes so that the magnitude would be accurate for similar face patterns. Secondly, the graphs are used for referring specific facial landmarks with nodes also called fiducial points. Lastly, the jets are combined to get graphical representation of the face. The jets forming an image graph is shown in figure below.

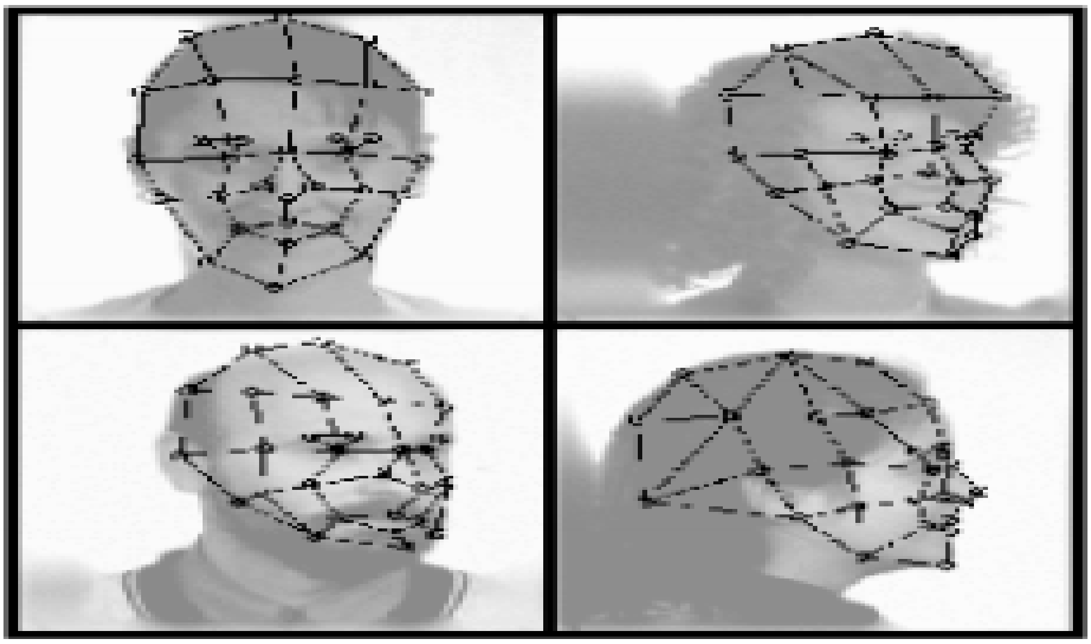


Figure 2. Image graph of the different faces. Reprinted from Wiskott et al. [8, 9]

As seen in figure 2, different fiducial points on the object represents the datasets and collectively form an image graph for different objects. The grids have more nodes on the face and the fiducial points is found accurately for the objects. However, the one with beard has slight difference on the nodes and the chin nodes is not found accurately.

### 2.1.3 Template Matching

Template matching is a process of face recognition where facial features or whole face is used as a template and two-dimensional array of intensity values of a test image is compared using a correlation or Euclidean distance with the template [6, 92]. Multiple templates from different viewpoints can be used to represent the single face. Several set of distinctive templates from a single viewpoint can also represent a single face. The process involves steps from defining a template covering needed regions like eyes, nose and mouth to feature extraction of those regions with skin color detection and edge template extraction. [9, 1253-1254.] An instance on which the template is defined focusing on regions around eyes, nose and mouth is shown in figure 3.

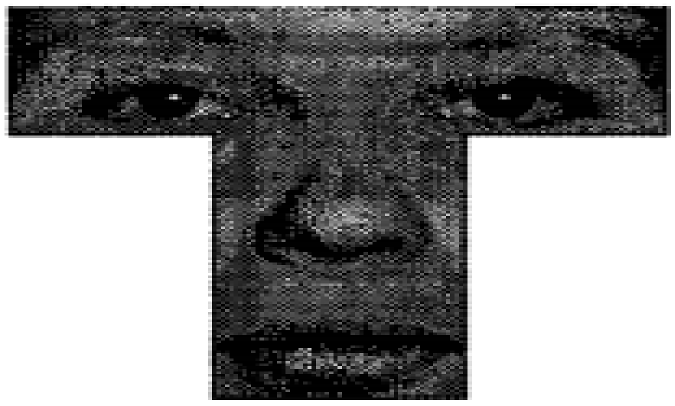


Figure 3. Face template with color. Reprinted from Karungaru et al. [9, 1254]

Figure 3 presents a T shaped template formation from the face template where eyes, nose and mouth are focused and the distance between them. The template is extracted from original image making it color template.

To have more accurate data, template matching is done with genetic algorithm. Since a template produced from the image can differ from the one in the database due to size, the use of genetic algorithm helps to adjust the size of template continuously until the fitness of algorithm reaches beyond the set threshold. [9, 1253-1254.]

### 2.1.4 Neural Networks

Neural networks are framed after the human brain and are set of algorithms, designed to identify patterns. They recognize patterns in numerical form and are contained in vectors. They are used for pattern recognition and helps in clustering and classifying the input data. The advantage of using neural network for face detection is due to its non-linearity in the network. A single layer adaptive network which had a separate network for each face or individual called WISARD was the first artificial neural network used for face recognition. [6, 91.]

The algorithm is used with various other techniques to achieve better result. There are methods which combines tools like PCA with neural networks to create a hybrid classifier for face recognition. Neural network is combined with feature extraction for face detection and classification. First, the input image is preprocessed with feature extraction. The step helps in decreasing time and provides better quality of image. Then the features are extracted using Gabor filters similar to the method in Graph Matching. For face identification, characteristics of face such as space between eyes, shape of zygotic bone which are about 80 in numbers are taken as parameters for forming face specification [10]. The parameters are called feature vector. Lastly, the images formed by feature vector are classified using Neural network. [11, 32.]

Neural network can cluster and classify data. Classification can be done on labeled datasets. However, when there are unlabeled data sets, it clusters them with respect to similarities. Basically, neural networks are provided with various data to train them with positive training data and negative training data. [12, 24-26.] For instance, the training data consists of several face and non-face images for face detection. When the neural network detects faces in non-face images which are errors, then those are classified and added to negative training examples so that neural network identifies accurately for the similar input data.

Hence, the research shows that several techniques and methods are used to identify and verify face. Although, the accuracy of the results is promising, yet the methods do not provide 100% accuracy. More than one or two methods are needed and combined to minimize the failure rate.

## 2.2 Internet of Things

Internet of Things or IOT is a system where devices are connected to network and the data generated by those devices are shared and transferred without any or with least human interaction. Kevin Ashton first presented the concept of IOT with the radio-frequency identification (RFID) technology. [13,41.] IOT is an environment of connected embedded systems sharing information over a network. Thus, it is also known as machine-to-machine (M2M) communication. [14,1.] The development of internet, micro computing, mini hardware manufacturing along with M2M communication contributed the ecosystem to grow rapidly [15, 20197]. Any device which can be connected to internet and can be controlled is an IOT device. Every device in IOT has unique identifiers (UIDs). A device or system in IOT can gather data, share or manage it. At present, IOT ecosystem is a sensor network of million devices communicating and sharing data with each other.

An IOT ecosystem must have internet enabled devices in it. Basically, those devices utilize sensors or processors to gather, share or manage data acquired from the embedded environment. The devices transfer data with connected IOT gateway or any other device for analyzing it in cloud or local machine. IOT system communicate through standards and protocols. An IOT system consists of a three-level architecture: devices, gate ways and data systems as shown in figure 4.

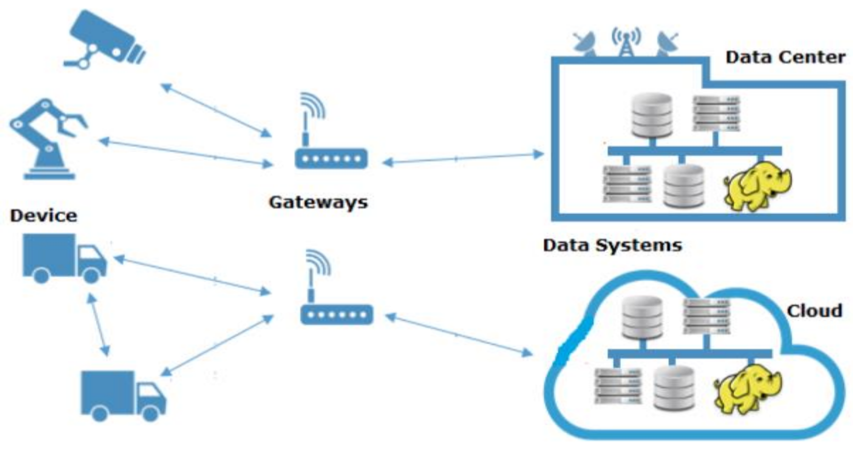


Figure 4. Three level of IOT Architecture. Reprinted from Perwej et al [15, 20197]

Figure 4 presents three different level in IOT architecture and inter-connection within these levels where data proceeds. A device transfers its data via gateway node and gateway node sends the received data to the cloud or data center where the data is analyzed or integrated. The first level is where the devices communicate with gateway or within themselves. The use case when devices share data with another device mostly requires instant data sharing. Furthermore, the devices also send data to gateway node, which is called “fog” layer as gateways are ubiquitous and secured. Gateways have two functions. Either they route data received from devices to the suitable data centers or they examine or cluster data and send to the core system or answer to the devices. Since gateways are advance computing device, the communication protocol between gateway and data systems is determined on the basis of data traffic patterns and security requirements. Finally, when the data is received by data systems, it is integrated to applications and also analyzed. The data flow is bi directional, that is, data systems can also send data to the device via gateway if any modification or change is required in accordance to the system. [13, 42-43; 15, 20197.]

## 2.3 Cloud Computing

Cloud computing is a computing standard in which multiple systems are connected in private or public networks in order to deliver dynamically scalable infrastructure for application, data and storage. The concept of cloud computing is based on an elemental principal of “reusability of IT capabilities”. [16, 6.] Cloud computing model consists of computing power, software, storage services and platforms which are provided to external customers via the internet with respect to their demand [17, 1]. Cloud computing is service oriented, and the cost depends on the use-basis, making it as a popular solution to deliver in-expensive and convenient mode for externalized Information Technology (IT) resources.

A cloud has capabilities like elasticity and self-provisioning of resources, that is, cloud consumer can use the resource on the basis of their needs and can remove it and the resource goes back to into other pool of resources with themselves through browser or some portal [18, 343]. Such model helps to deduce the cost of idle resource. The cloud is a distributed environment that accumulates resources so that they perform together which helps to manage workload in the environment [16, 9].

Clouds has different version depending on the needs. Cloud has several deployment models and delivery services. The deployment models and delivery services are explained in sub-section.

### 2.3.1 Cloud Deployment Models

Cloud deployment models are classified on the basis of accessibility of resources.

Public Cloud

Public cloud is where the resources are provided as a service from a third-party service provider over the internet with pay-per-usage fee [17, 3]. It allows user to scale their resources without the need of hardware.

Private Cloud

Private cloud is where the resources are deployed inside a firewall and managed by the consumer organization [17, 3]. The software and hardware infrastructure and access to it is managed by consumer. Usually, the resources are not shared outside the cloud user organization in this deployment model.

Community Cloud

In this cloud model, a group of organization with common specific need use and control the resources. The members of community have access to the data and software in cloud. [18, 344.]

Hybrid Cloud

The combination of two or more private, public or community cloud that interoperates forms hybrid cloud [17, 3; 18, 344].

### 2.3.2 Cloud Delivery Services

Based on the type of capabilities, cloud delivery services are broadly categorized under three services.

Infrastructure as a service

Infrastructure as a service or IaaS, offers computational infrastructure over the internet [17, 2]. It includes basic storage and computing capabilities like servers, network tools, storages, data center and so on. Cloud user buys the infrastructure as resources rather than the equipment itself and manages his/her own software or applications in the provided infrastructure.

Platform as a service

Platform as a service, also called PaaS is a cloud delivery model where infrastructure combined with software resources, middleware and deployment tools required for application development is provided over the internet [16, 7]. In other terms, platforms required to design, develop, build and test applications are delivered. In this service model, user administers the applications running in environment, not the operating systems, hardware or network where they are running.

Software as a service

In this delivery service, software that is managed and delivered remotely by cloud service provider is offered to user with pay per use model [18, 344]. SaaS application is built on top of both IaaS and PaaS. Several end users use single instance of the service running on cloud. Basically, the SaaS user does not have control and knowledge on the underlying structure.

At present, there are several cloud service providers which are offering a wide range of these services and delivery models. Amazon Web Services, Microsoft Azure, Google Cloud and IBM cloud are some of the major cloud service providers. In this project, we are going to use cloud services since it is convenient to manage and provides the needed technical requirements.

## 2.4 Technical Specification

The section includes the technology used to build the system and outlines their functions and role in the project.

### 2.4.1 AWS Services

Amazon Web Services (AWS) is highly reliable cloud service platform provided by Amazon. The platform offers compute powers, storage services, content delivery, hybrid cloud, networking and many more services with PaaS cloud computing model. AWS provides services from all over the world through dozens of data centers spread with Available Zones across the world. Being the cloud service providing pay-as-you-go model, AWS provides lower capital expenditure and high value with high performance and scalable services. Services like AWS Rekognition for facial detection and analysis, AWS Simple Email Services (SES) for sending email alerts, AWS Simple Notification Service (SNS) for sending Short Message Service (SMS), AWS Simple Storage Service (S3) for storing images in the form of objects and AWS Relational Database Service (RDS) for relational database management systems are used in the project to build car safety system along with other technology integration.

AWS Rekognition is a service available in Amazon AI suite for facial analysis from images or videos with deep learning technology [19, 158-159]. The service analyses images based on similarity score which is a statistical measure to find the match between two different faces. Face comparison, face detection, celebrity detection, object or scene detection, unsafe content detection, text in images are the key features available in Amazon Rekognition. [20, 1-2.] The service is used in the project to detect face and calculate emotions through images.

AWS SES is email sending service provided by Amazon to help send notifications on email [21, 1]. The service is reliable and cost effective regardless of the size of the businesses. The service is used in the project to send email alerts on theft and drowsiness detection.

AWS SNS is fully managed distributed publish-subscribe messaging service provided by Amazon [22, 1]. The service helps to push notification to mobile app directly and send notification by SMS text messages to the client. Thus, the service is used in the project to send SMS message when alert is detected.

AWS S3 is an IaaS solution provided by AWS for storing data as an object [19, 3-4]. The service is highly scalable, provides low latency and unlimited storage capacity. The project used the service to store images of drivers.

### 2.4.2 Raspberry PI

Raspberry Pi is a portable minicomputer which runs Linux and provides sets of General Purpose Input Output (GPIO) pins allowing developers and engineers control the electronic components exploring the world of IOT. The device was developed in the laboratory of University of Cambridge which was later released in 2012 by Raspberry Pi Foundation [23, 410].

The device performs in open-source eco system and supports Raspbian as its official supported operating system which is open source itself and runs suite of open source software [24]. GPIO pin connectors in the device helps to communicate with sensors, embedded systems and motors thereby allowing engineers build IOT based platforms. The platform is flexible enough to apply utilities and to experiment almost all electronic components. The device has been used worldwide to power the home automation system, control quadcopters, smart home systems, remote control and so on. The device costs as low as $50 which is affordable by everyone and so is its peripherals which is the reason it suites for any kind of projects.

As we were also building IOT based system for car safety and security, we believed that investment on Raspberry Pi is worthy. Thus, we decided to go with Raspberry Pi as we believed that the platform would fulfill all the requirements.

### 2.4.3 Google Data Studio

Google data studio is visualization and reporting tool created by Google to power data with interactive dashboard. Data from different marketing platform campaigns can be served to data studio with the help of various built-in and external data connectors. From very basic table to advance charts and with the control of filters, marketers can analyze data using google data studio. The platform allows share the dashboard within team with different access levels and permissions. The available 500+ data connectors helps anyone easily setup the data source and make the data flow up and running with few number of clicks. The data can be filtered, and metrics can be controlled with default calculation features in dashboard. The layout can be built with drag-and-drop feature and the User Interface (UI) is very intuitive. [25.]

As the project consists of sentiment and alerts related facial data, it was important to build the graphical representation of the records. Thus, building a dashboard to stream all available data and sharing insight with the member users was one of the main part. In real world, the dashboard could be implemented within a car as a built-in feature for best user experience. The flexibility of Google Data Studio and available features are believed to fulfill the requirement of the project. Thus, we choose to provide insight of driver's sentiments and facial related data through the power of Google Data Studio.

### 2.4.4 Database

Database is an organized searchable collection of information for storing and accessing them electronically from different systems with the help of queries [26, 1]. Database consists of tables, queries, views, reports and other objects which can be accessed and maintained with the help of Database Management System (DBMS). Before the introduction of DBMS, file-processing systems were used which uses various files to store records and needs applications for extracting and adding records to the appropriate files [26, 3].

Use of database prevents data redundancy and inconsistency issues which might occur with files system. Thus, the project has made use of database for storing the information received from Application Programming Interface (APIs). There are many databases available in the market depending upon the usage requirements. Some of the type to enlists are Centralized database, Personal database, Distributed database, End-user database, SQL or Relational database, Graph database, NoSQL database and so on. The project has made use of relational database types as the data are categorized by a set of tables with rows and columns which are used to organize dataset. Relational Database Management System (RDBMS) are used to retrieve, update and manage relational database which uses Structured Query Language (SQL) or SQL like languages to communicate data from and to the database system [26, 12]. Some of the examples of relational databases management systems are SQLite, MySQL, PostgreSQL and Oracle DB.

We have used PostgreSQL Database which is open source and reliable as it has large community of developers. The database has many built in functions and features that helps skip complex configuration which makes implementation efficient and fast. Since our system will have to integrate with visualization tool like Google Data Studio, use of PostgreSQL database can be used with built-in connector feature. The database has important features like functions, triggers, user roles, stored procedure, views, temporary table, constraint and so on which are the core components to work with and more than what we need in this project. Moreover, the database performance is high enough to fit the requirement of the project.

The project uses Relational Database Service (RDS) provided by AWS to host the database instance. The service is reliable, scalable and optimized for high performance.

# 3 Methods and Materials

The section includes the architecture and overall view of the project. It explains features and functionalities of the system and methods followed to achieve the goal.

## 3.1 Application Architecture

### 3.1.1 Application Site Map

The project was built with different modern technologies and integrations with IOT devices and cloud technologies. It is comprised of many components and sub-components in terms of technologies. The system was designed with the concept of backend and frontend technologies. The implementation of program execution and logic calculations were categorized as backend while the visualization and user interaction part were put into frontend category. Different tools and technologies had been used to build a single functional system. Figure 5 illustrates the overall architecture of the system.

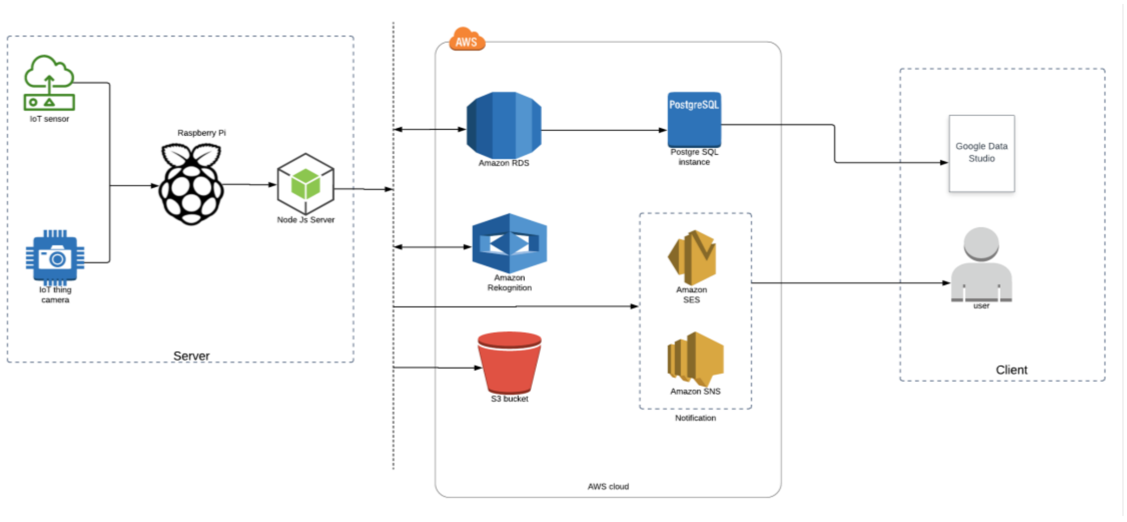


Figure 5. System architecture

As seen in figure 5, there are 3 different components building a single system. Server consists of different IOT devices integration including server for the code executions and logic handlings. Server has direct connection to the AWS services to interact with the data generated on the server. Acting as a complete backend service, cloud and server produces the result which then transfers it to client for visualization and analysis of the data.

### 3.1.2 Hardware Overview

The project was carried out using hardware devices integrated with modern stacks. Tools like Raspberry PI 3B+, PI Camera Module Version 2 and Passive Infrared (PIR) Motion Sensor were used to build the different functionalities within the system.

The project was built under raspberry pi 3B+ which is latest at the time the project was built. The specification of the version of the raspberry pi was believed to fulfill the requirements of the project. The main focus was to cover all the requirement specification or features and integrating IOT devices to achieve the goal. The server was built under the device and acted as center point for getting data and making calculations.

PIR Motion sensor was used in the project to sense human motion which outputs the digital signal to a server running in raspberry Pi. An object triggering motion emits heat in the form of Infrared radiation (IR) which results change in temperature. The change in temperature is converted into voltage which is considered as motion detected by the sensor.  The sensor is small and low-power yet being very powerful that fulfilled the need of the project. It is connected to raspberry pi through GPIO Pins. On detecting motion, the sensor outputted 5V digital signal to the raspberry Pi which was used to activate camera for capturing images and process it further. Figure 6 depicts the architecture of PIR motion sensor.

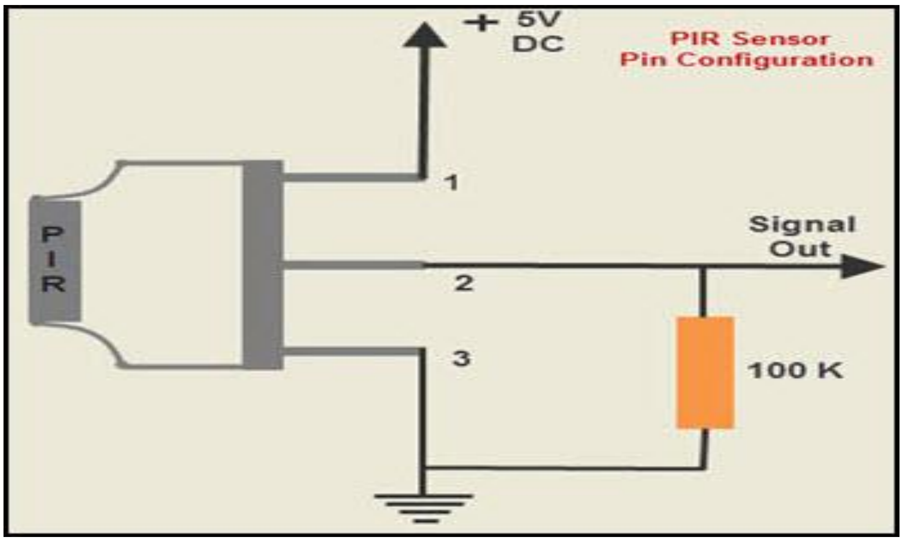


Figure 6. Motion Sensor semantic diagram. Reprinted from Tarus [27, 21]

As shown in figure 6, PIR Motion sensor consists of 3 pins for input, output and ground purpose. The setup of the sensor with raspberry Pi carried out following the official recommendation from raspberry pi. It shows that the 5 Voltage is fed through the power pin. On motion detection, 3 to 5 Voltage of output will be passed to raspberry pi through output pin.

Raspberry pi camera module v2 is official raspberry pi product with 8 Megapixel Sony image sensor featuring a fixed focus lens for capturing still videos and pictures. The product has fulfilled the need of the project providing high quality images which made detecting faces and analyzing image easy. Figure 7 illustrates the camera module V2 connected with raspberry Pi.

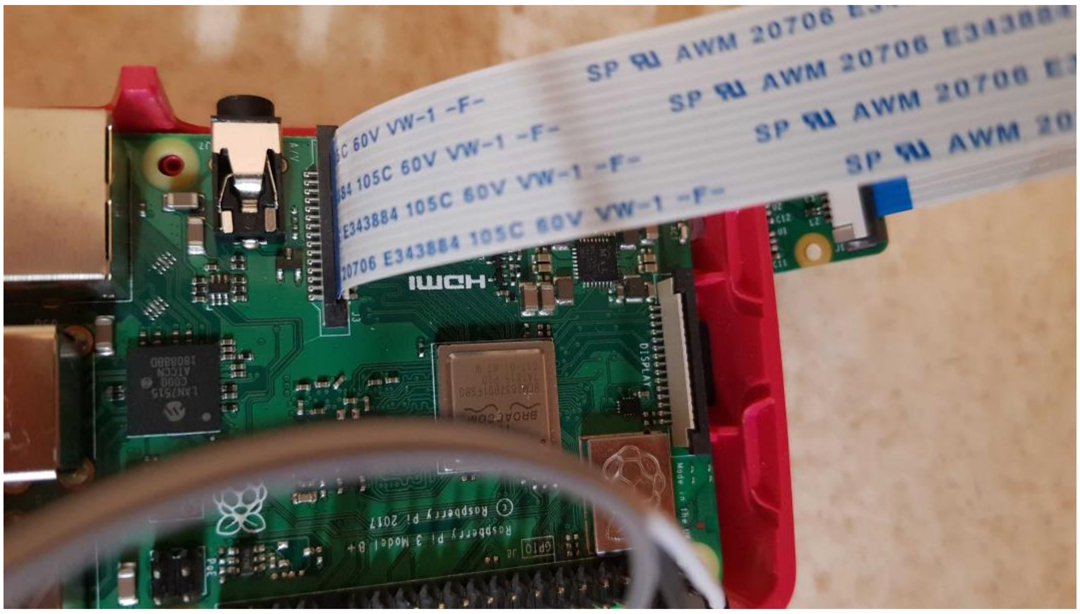


Figure 7. Camera module V2 setup with Raspberry Pi

As seen in figure 7, Camera module v2 is connected to raspberry pi for streaming videos and capturing image in allocated time frame. Raspberry Pi 3B+ has built in camera port to connect the camera module. On enabling camera, camera module can be used with command line application or can be run programmatically to capture videos and pictures.

### 3.1.3 Software Overview

The project was built using different libraries and frameworks of JavaScript programming language. The server was powered by node.js scripting language which run into raspberry pi. Different JavaScript based libraries were used to build different features and functionalities. The data coming through motion sensor triggered the camera module to capture images and thereby streaming data to the server. All the calculations and heavy lifting was carried out in node.js server. Some of the popular libraries that were used in the projects were mostly built for other programming languages. However, the one that has binding with Node.js were used in the project despite of having limitations of certain features but could fulfill the need of the project. Some of the examples are OpenCV and Tensor flow libraries.

OpenCV is a machine learning library mainly aimed for real time computer vision and comprised of many algorithms for image processing, object detection, face recognition, object identification, camera object tracking and so on. The primary interface of the library is in C++ and provides templated interfaces for C, Python and Java. [28.] The project had made use of Node.js binding of the OpenCV library named node-opencv with limited features. However, the Node.js version of library had been able to fulfill the need of the project.

The project also used tensorflow JavaScript based library to optimize the performance of face detection. Tensorflow is machine learning software library developer by Google used for processing data, numerical calculations through neural networking technology. The project had made use of FaceNet which is deep convolutional network backed by Tensorflow designed to solve face verification and face related measurements. However, the project could not benefit all the algorithms and features provided by the library because of language specific limitation. Thus, the project used Node.js implementation of FaceNet library which was limited to certain features and functionalities yet fulfilling the project requirements.

The project had utilized the cloud service technologies for most of the task executions. Image processing, notification and database services from AWS are highly popular and server less. Thus, the project was built with the integration of those services for better performance and highly reliability. AWS Rekognition provides fully managed object detection, analyze and image processing features which was very reliable when tested. The object detection API provides the people’s sentiment on real time and facial comparison from images which was very important feature of the system that we built. The project used AWS JavaScript Software Development Kit (SDK) for the interaction with the AWS APIs. Figure 8 depicts the overall program flow of the system.

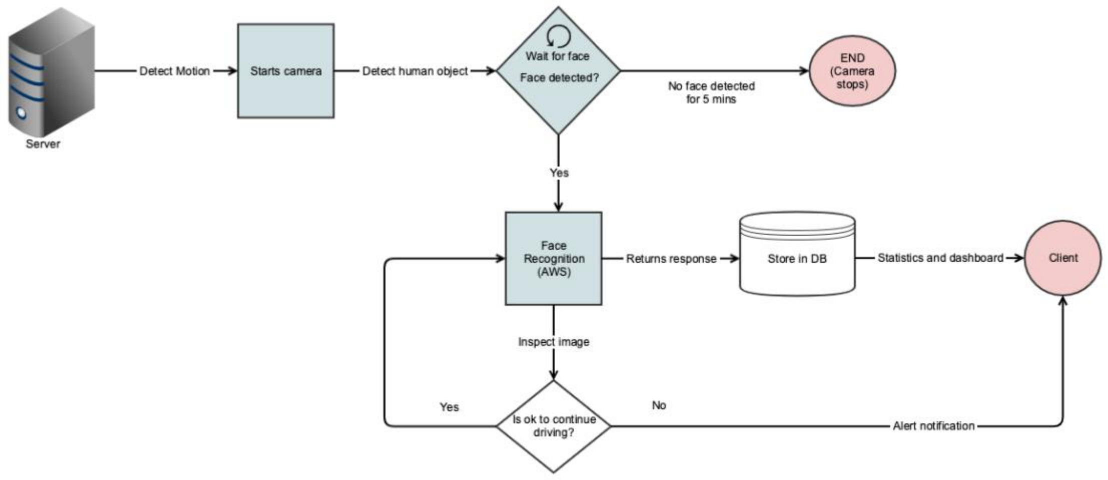


Figure 8. Program flow diagram

Figure 8 presents the program flow of the system that is plain and manageable. The server was integrated with PIR motion sensors and camera module. When the sensor gets triggered, it outputs the digital signal to the server. The server checks the status of the signal if the motion is active in status. On motion active status, the server starts camera and start capturing images. The images then will be processed by OpenCV and FaceNet based Node.js libraries to detect human faces in the image. If human face is not detected in image for 5 consecutive minutes, the system will shut down the camera automatically.

Image with human face is passed to AWS Rekognition service for getting image metadata that we use to find out drowsiness and personal sentiments. The API response contains image id, face collection id, face id of the person and emotions metadata such as happiness, smile, sad, eyes open status and so on. This information is stored in PostgreSQL database hosted in AWS along with the images in AWS S3 bucket. For each image response, the system analyses the data for drowsiness and possible theft alert. The system checks if the driver is known by comparing driver face with pre-existing face collection as soon as the system detects face for first time. If the system finds drowsiness or theft possibility, it sends alert notification to the members in face collection via email and phone message with detected image as an attachment.

## 3.2 Features and Functionalities

The project focused on features mostly concerning on driver safety and security. The section includes the main features and functionalities present in the system.

### 3.2.1 Security

One of the main focus of the project was to ensure the vehicle safety. Despite of the heavy investment on vehicle example car, the risk of being stolen has not been diminished. The project was built to minimize theft possibility with the help of deep learning technology. With the implementation of machine learning technology integrating cloud services like AWS Rekognition, the system could ensure that a car was on safe hand.

The system completely relied on face detection and recognition technology for security of car. When image is captured, the system triggers the request on AWS Rekognition face comparison API to compare the face if it exists on face collection. The collection contains images of the known members. The response received from the service would tell if the driver was known and was part of a member. Figure 9 shows steps implemented for security of a car.

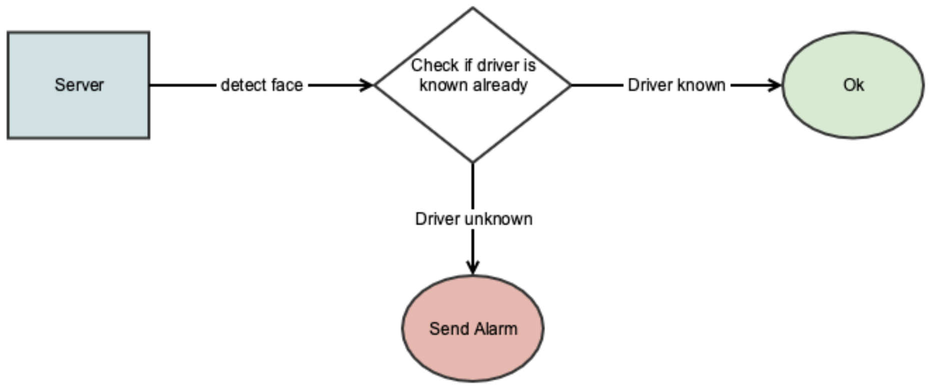


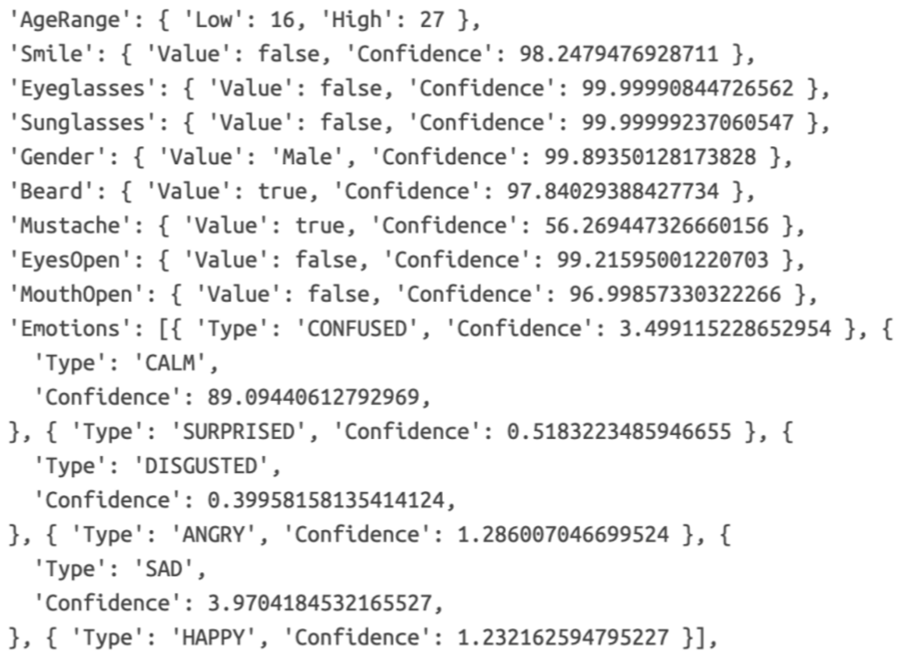
Figure 9. Car security model diagram

As presented in figure 9, car security was implemented with integration of deep learning face detection service which checked if the person sitting on driver seat was part of the member. The notification alarm system was implemented as part of system to alert the family member for taking further actions. The program continues normally if the driver is known without triggering an alert.

### 3.2.2 Safety

The most important feature implemented on the system was driver safety. Driver safety has been key concern all over the world as many accidents do happen despite of brand and condition of the vehicle. The reasons could be anything from excessive drinking to road conditions. Thus, it is very important for the driver to be in safe condition before he sits in driver seat. The condition of driver also affects the life of passengers and pedestrians. Understanding the emotions of the driver and checking if the driver is in suitable condition to drive a car was the main focus of the project.

The project mostly relied on facial recognition service named AWS Rekognition provided by Amazon. The service processes the image containing face and analyze the data and response the system with emotions and image metadata. The response data contains the properties like gender, eyes open, smile and emotions metadata such as happiness, sadness, calmness, angriness and so on which are the key source of safety calculations. Figure 10 illustrates the part of the response received from the AWS face detection API.



```
'AgeRange': { 'Low': 16, 'High': 27 },
'Smile': { 'Value': false, 'Confidence': 98.2479476928711 },
'Eyeglasses': { 'Value': false, 'Confidence': 99.99990844726562 },
'Sunglasses': { 'Value': false, 'Confidence': 99.99999237060547 },
'Gender': { 'Value': 'Male', 'Confidence': 99.89350128173828 },
'Beard': { 'Value': true, 'Confidence': 97.84029388427734 },
'Mustache': { 'Value': true, 'Confidence': 56.269447326660156 },
'EyesOpen': { 'Value': false, 'Confidence': 99.21595001220703 },
'MouthOpen': { 'Value': false, 'Confidence': 96.99857330322266 },
'Emotions': [{ 'Type': 'CONFUSED', 'Confidence': 3.499115228652954 }, {
  'Type': 'CALM',
  'Confidence': 89.09440612792969,
}, { 'Type': 'SURPRISED', 'Confidence': 0.5183223485946655 }, {
  'Type': 'DISGUSTED',
  'Confidence': 0.39958158135414124,
}, { 'Type': 'ANGRY', 'Confidence': 1.286007046699524 }, {
  'Type': 'SAD',
  'Confidence': 3.9704184532165527,
}, { 'Type': 'HAPPY', 'Confidence': 1.232162594795227 }],
```

Figure 10. Part of response received from AWS face detection API

As shown in figure 10, AWS APIs provide large number of properties on analyzed face. These metadata are highly accurate as the services are highly trained for long period. However, all of these properties are not used on the project as the scope of the project is limited to facial detection for car safety and security related purpose. The main properties used by the projects are age range for gathering age-based data which might be useful for future analyzation. Smile properties is used for finding mood of the driver. Gender based data is useful for future purpose for investigations on accidents. Eyes open properties is mostly useful for finding out drowsiness to prevent any risks while driving. Emotions are measured based on high degree out of calmness, confusion, disgust, angriness and happiness. High degree of percentage is taken into account while measuring emotions. The data are useful not only for safety point of view but also for further investigations in future.

Eyes open and close properties were checked and calculated by the system if the driver was feeling drowsy. The response was sent to server on every two seconds interval and eyes open status were counted based on eyes close value true for three consecutive times in a row. The project had limited budget which restricted the amount of request to the cloud APIs. More accurate data could be generated with real time video streaming and real- time drowsiness detection.

The system detects that the eye close for long period time is abnormal. The drowsiness feeling would occur only when the driver was drunk, or the driver was on unhealthy condition. The system had three seconds period of time to check for drowsiness effect. Figure 11 illustrates the drowsiness flow diagram implemented on the system.

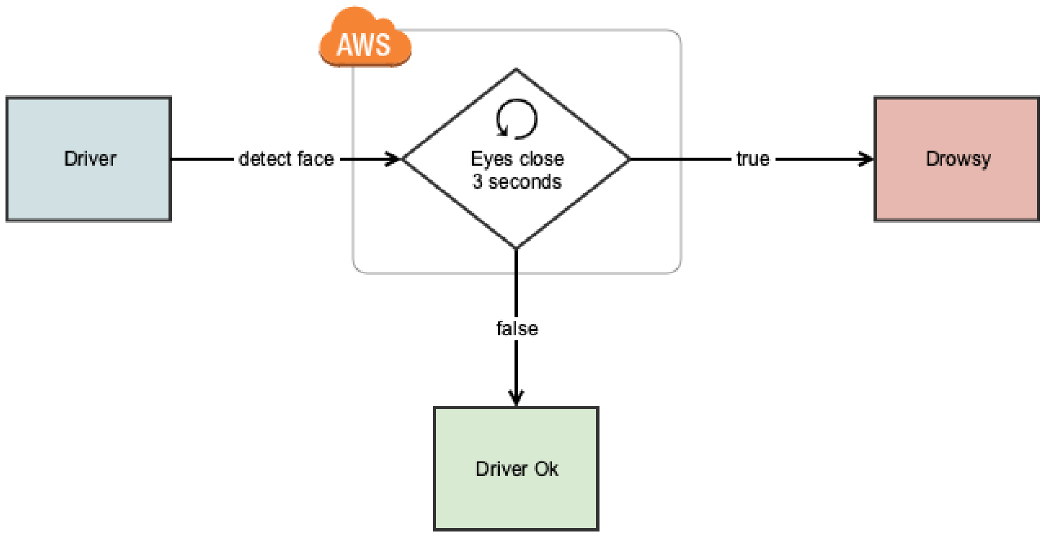


Figure 11. Drowsiness detection implementation diagram

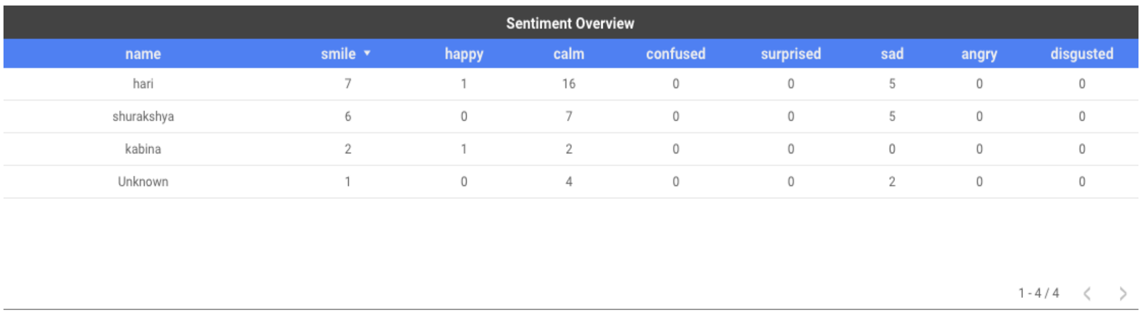
As presented on figure 11, the system detects the face of the driver based on eye close status if occurred unnaturally. The eye close property received from face recognition service was the source of sleepiness detection.

### 3.2.3 Sentiment Analysis and Visualization

The scope of the project is not limited to drowsiness and theft detection. With safety and security of the car, the project has focused on sentiment analysis of the known member while being on driver seat as well as visualization of data for future references. The sentiment analysis and measurement of driver emotions are very important factor as it provides the overall idea of driver's emotion while driving. One can assume the likeliness of any danger based on these data. For example, if the drive is too sad and angry while driving, it is more likely that the speed of the car would go high and stability of car goes beyond the neutrality.

Adding sentiment analysis data and displaying visually in car as a built-in feature would be very helpful for the drivers. Car manufacturing companies should concern about this implementation as this feature could help future transportation preventing car accidents and saving people's life. The feature is mostly focused for long journey vehicles as mood of the driver can easily varies during the long hours of journey. Displaying correct data could help driver take some action within and help prevent future accidents. For example, taking 30 mins break during high level of anger or possible drowsiness could help regain energy to continue the journey safely. Thus, real time dashboard inbuilt within a car is one of the future aim of the project.

Building sentiment analysis dashboard and representing one’s emotions visually is one of the important feature of the project. The project consists of one visual sentiment dashboard where family members could check the data from past. The data were grouped based on member name and id. So, user can view all emotions related data and alerts sent for drowsiness and security purpose. The data are stored even in the case of thefts and accidents. The visual representation of data is believed to provide accuracy of one’s feeling while driving as well as attention to be made for specific users. Figure 12 represents the sentiment data displayed in tabular in dashboard.



Sentiment Overview

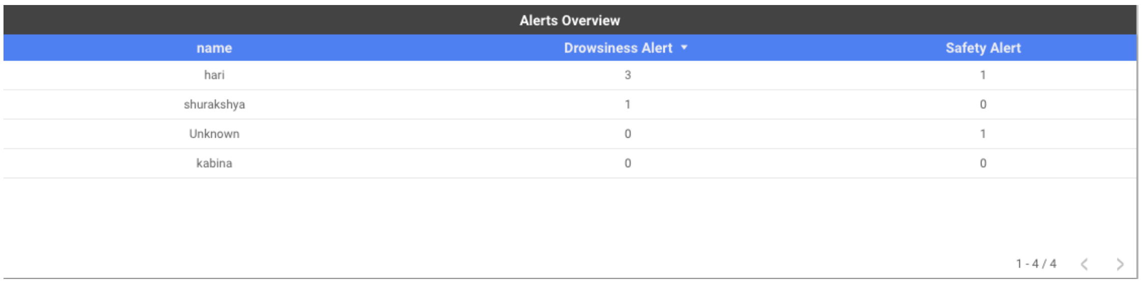
| name | smile ▾ | happy | calm | confused | surprised | sad | angry | disgusted |
|---|---|---|---|---|---|---|---|---|
| hari | 7 | 1 | 16 | 0 | 0 | 5 | 0 | 0 |
| shurakshya | 6 | 0 | 7 | 0 | 0 | 5 | 0 | 0 |
| kabina | 2 | 1 | 2 | 0 | 0 | 0 | 0 | 0 |
| Unknown | 1 | 0 | 4 | 0 | 0 | 2 | 0 | 0 |

1 - 4 / 4

Figure 12. Tabular representation of user sentiments

As illustrated in figure 12, a dashboard had been built to accumulate all sentiments from every user on driver seat. The data is grouped by user name. On theft detection, the name is filled with unknown tag. However, the sentiment will still be tracked and presented in the dashboard. The dashboard can be filtered by name and date.

Alerts are shown in different table in the dashboard. Alerts usually represent problems as they get triggered only on abnormal stages. More the alert, more attention need to be provided to the user. For example, high number of drowsiness alert for a user means that the user is driving heavily while being in drowsiness condition. It also means that the user should rest more before drive or control the speed of car while driving to prevent risk. Figure 13 illustrates the alerts in tabular format.



Alerts Overview

| name | Drowsiness Alert ▾ | Safety Alert |
|---|---|---|
| hari | 3 | 1 |
| shurakshya | 1 | 0 |
| Unknown | 0 | 1 |
| kabina | 0 | 0 |

1 - 4 / 4

Figure 13. Tabular representation of Alerts

Figure 13 shows alerts that are grouped based on driver's name. Drowsiness and safety alerts are counted when they get triggered and accumulated by user face id and user name.

The dashboard also contains more visual representation of sentiments and alerts. The visual representation makes easier to grasp the trend of user sentiments. The dashboard consists of per user based sentiment analysis charts or graphical representation which provides clear picture about how things are going in terms of safety and security. Figure 14 illustrates the visual representation of user sentiments.

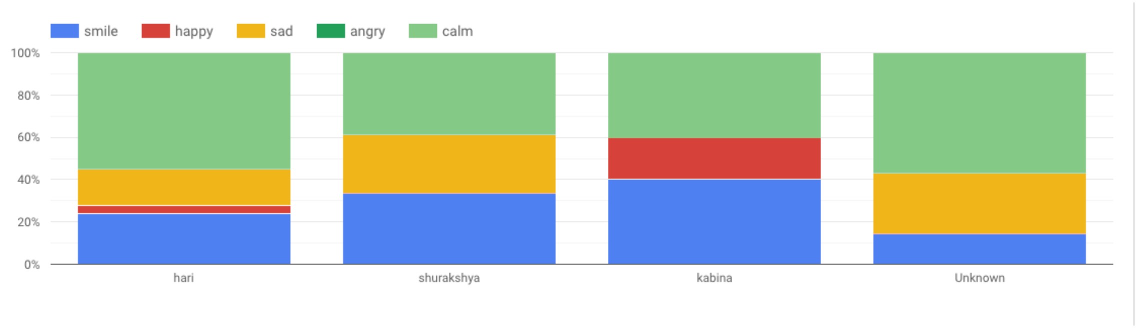


Figure 14. Visual representation of driver's sentiment

As figure 14 illustrates, it is easier to understand the graphical representation of emotions. The chart provides clear picture on how happy or sad the user was while driving. The data is very helpful in order to prevent future risks as one's emotion is directly related to the risks while driving.

The visual representation of the dashboard is something a member of the driver group can take benefit from. The data easily shows the reliable driver and the driver who should be put away from the driver seat. The data also represents if the driver should be provided with other passengers as life of other passengers depends upon him/her.

The data is not only valuable for small driving family member but also large corporates on transportation industries. Implementation of recognition technology and finding out the sentiments of drivers and conditions while driving changes way of living. The implementation should be tested well and put alive as a built-in feature in any vehicle mostly on public transportation.

## 3.3 Implementation

The section includes setup of different components and their implementations for building the system. This section is divided into 5 different phases which will cover tools setup, programming language choice, actual code implementation and logics implemented for building the system.

### 3.3.1 Phase 1: Raspberry Pi and peripheral devices setup

The first phase of building the system was choosing tools to build a server where all the logics and calculations were carried out. Because of IOT based solution to be integrated with cloud platform, we choose to use Raspberry Pi and its peripherals. We used the device to connect different external devices like camera module and motion sensor. It also executes whole program to function the system we built.

Node.js would be the choice of server side scripting language. So, we decided to build a Node.js backend which would run in Raspberry pi environment. At the time of building the project, the latest version of Node.js, 11.14.0 was installed along with all the necessary libraries and Raspberry Pi 3B+ model was used which was also the latest product in Raspberry Pi range. The device is very powerful with 4 Universal Serial Bus (USB) 2.0 ports, full-sized HDMI, Camera Module port for connecting Raspberry Pi camera, 5 Voltage power input and built-in wireless Local Area Network (LAN) and Bluetooth 4.2. Secure Digital (SD) flash storage card was used as a physical storage where Raspbian was used as Operating System (OS) which was installed using New Out of the Box Software (NOOBS), an easy operating system installer containing Raspbian. Figure 15 illustrates the ports available in Raspberry Pi for external IOT device connection.

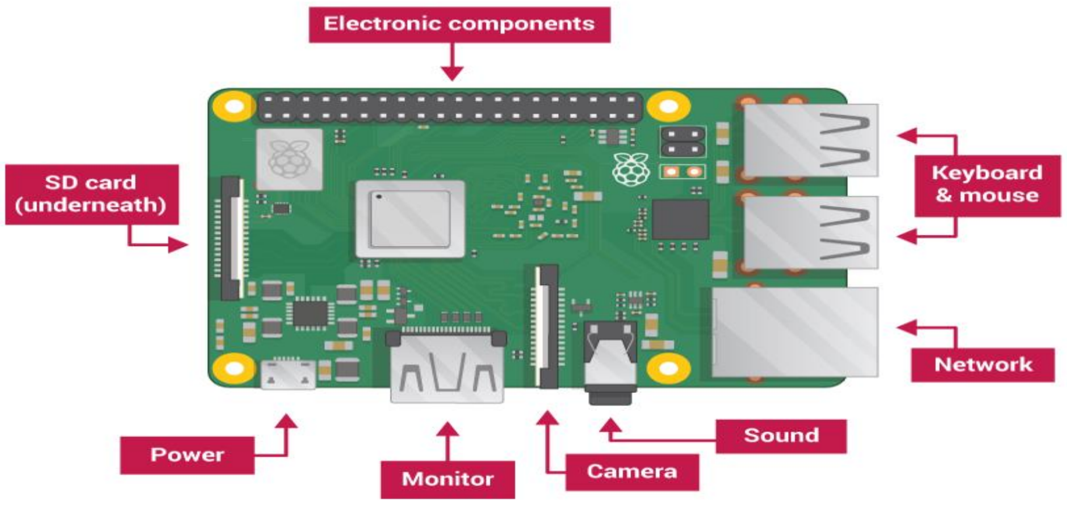


Figure 15. Raspberry Pi ports label diagram. Reprinted from Raspberry Pi projects [29]

As shown in figure 15, there are several ports in latest Raspberry Pi to connect different external devices. Out of these, sound port was not used in the project as it was not required by the project.

Raspbian is based on debian GNU/Linux and optimized for Raspberry Pi. It is official supported operating system of Raspberry Pi. [29.] Raspbian has pre-installed useful software for general purpose. Along with this installation, Pi Camera Module V2 was setup to the device through camera port for taking pictures. Camera software was enabled through Raspberry PI UI configuration and tested if the camera works. Raspberry Pi camera consists of 4 different applications named raspistill, raspivideo, raspiyuv and raspividyuv which uses the camera components to operate. Raspstill uses image encode component and raspivideo uses video encode component which are used by the system mostly.

The camera signal was tested using Node.js promised based library Pi-Camera, which is wrapper for the native raspberry Pi Camera Command Line Interface (CLI) tools for using camera module components. Version 1.2.1 of the library was used which was latest version at the time of development of the project. Listing 1 shows the Node.js camera test implementation.

```
const PiCamera = require('pi-camera');

class Camera {
  constructor() {
    this.videoCamera = new PiCamera({
      mode: 'video',
      output: `${__dirname}/video.h264`,
      width: 1920,
      height: 1080,
      timeout: 3000,
      nopreview: true,
    });
    this.photoCamera = new PiCamera({
      mode: 'photo',
      output: `${__dirname}/photo.jpg`,
      width: 640,
      height: 480,
      nopreview: true,
    });
  }

  recordVideo() {
    return this.videoCamera.record();
  }

  takePhoto() {
    return this.photoCamera.snap();
  }
}

module.exports = Camera;
```

Listing 1. Node.js subroutine to test the camera module

As shown in listing 1, a Camera class was created which contains two different methods for taking videos and capturing pictures from raspberry pi camera module. The methods were executed which stored the image and video files to the allocated folders in the device.

Out of 40 GPIO pins from electronic component of Raspberry Pi, three of the pins were used to connect PIR motion sensor in the project. However, any of the pins can be appointed as an output and input pin for a wide range of purpose. Two 5 Voltage pins, two 3.3 Voltage pins and number of ground pins are present on the board. Remaining pins are general purpose 3.3V pins. Figure 16 depicts the Raspberry Pi GPIO pin model for connecting PIR motion sensor.

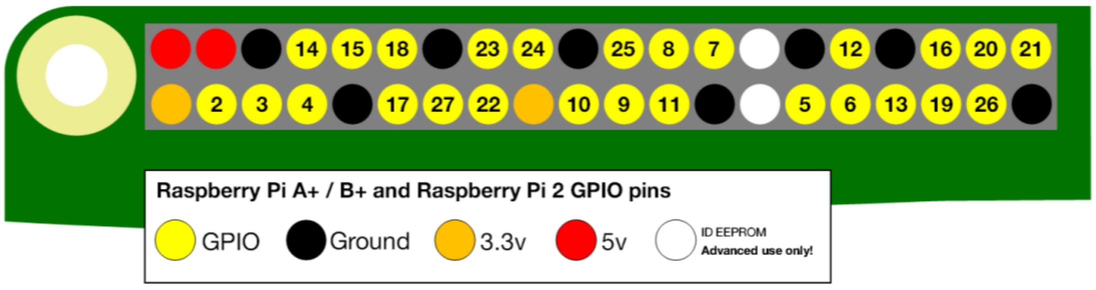


Figure 16. Raspberry PI GPIO pin model. Reprinted from Raspberry PI Documentation [30]

As illustrated on figure 16, two red marked pins are input voltage, two orange marked pins are 3.3 Voltage output pins, black marked pins are ground pins and remaining pins are general purpose pins. The GPIO pins can be controlled using number of programming languages and tools.

In the project, PIR motion sensor was connected through three GPIO pins. GPIO pins 2, 6 and 12 were used for connecting to input pin, ground pin and output pin of the PIR motion sensor respectively. The device takes 5 Voltage (5V) of input signal and outputs 3 to 5 Voltages (3-5V) of digital signal as a motion detection to the raspberry pi. The setup was well tested, and the signal was read using Node.js library called onoff. The library is built to access GPIO and detect interruption with Node.js on Linux boards like Raspberry Pi. The project used the latest version 4.1.1 of the library which was the recent version at the time of the development of the system. Listing 2 illustrates the code implementation to test the PIR motion sensor.

```
const GPIO = require('onoff').Gpio;
const pir = new GPIO(12, 'in', 'both');

pir.watch((err, value) => {
  if (value === 1) {
    console.log('Motion Active');
  } else {
    console.log('Motion Passive');
  }
});
```

Listing 2. Node.js subroutine to read the motion sensor signal

As illustrated above in listing 2, the system used onoff library to read the motion detection signal from PIR motion sensor. GPIO pin 12 was connected to input pin and third argument ‘both’ in GPIO constructor specified that both falling and rising interrupt edges need to be configured. The detection was outputted as binary value 1 and 0. 1 represents motion as active and 0 represents motion as passive.

The device and its peripherals were believed to fulfill the need of the project. The setups were tested properly before actual code implementation was begun. Figure 17 shows the hardware setup image with Raspberry Pi.

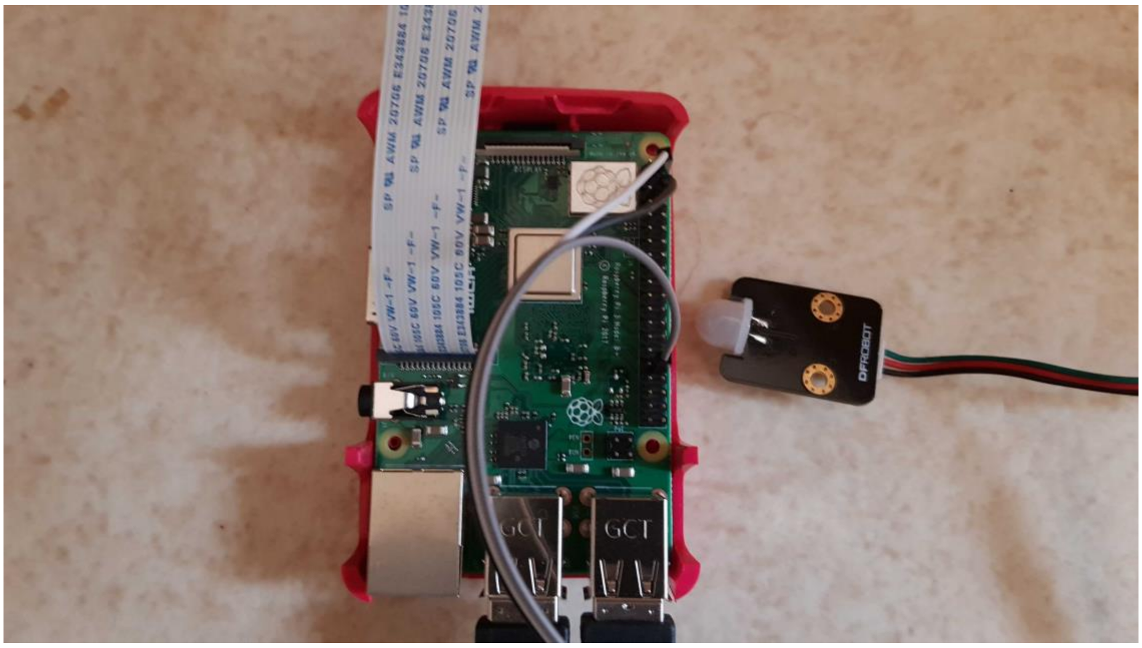


Figure 17. Raspberry Pi and peripherals final setup image

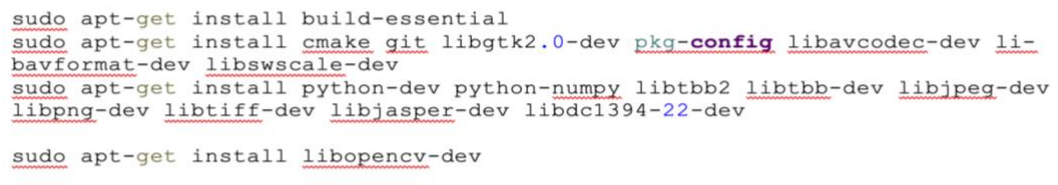
As shown in figure 17, all the peripheral devices like camera module and motion sensor were successfully connected to Raspberry Pi. The connection was however easy compared to testing and ensuring everything works.

The actual logic implementation and code writing were carried out on laptop or own machine because of efficiency and convenience. Version control and code shipment to production was carried out using Git and separate project repository was created on Github for the project. Thus, transferring code from local machine to Raspberry Pi was performed via Github repository. The first-time shipment of the code from local machine to Raspberry Pi environment would need libraries installation with Node Package Manager (NPM) script.

### 3.3.2 Phase 2: Implementing Face Detection using Camera Module

After completing device and peripheral setup, next phase of the system development is component implementation for building features. The phase explains implementation of face detection from captured images before passing to AWS Rekognition for analyzing it. As discussed earlier, we used Raspberry Pi camera module with Node.js based

OpenCV and FaceNet libraries to detect the image. Before installing these libraries, we installed all the dependencies required by OpenCV library and when installed successfully, we installed OpenCV library as shown in figure 18.



```
sudo apt-get install build-essential
sudo apt-get install cmake git libgtk2.0-dev pkg-config libavcodec-dev li-
bavformat-dev libswscale-dev
sudo apt-get install python-dev python-numpy libtbb2 libtbb-dev libjpeg-dev
libpng-dev libtiff-dev libjasper-dev libdc1394-22-dev

sudo apt-get install libopencv-dev
```

Figure 18. Scripts to install OpenCV and its dependencies

As presented in figure 18, all the dependencies required by OpenCV were installed followed by installing OpenCV itself. The latest version of the OpenCV at the time of building the system was 4.0 which was used in the project development.

The image was captured every possible second and stored in local drive. Before implementing facial detection directly from Pi Camera, we used the library to process image from local machine and stored the detected image in another folder of local machine. The processed image was stored with rectangular mark on face to identify the accuracy of the result. When tested successfully, the actual implementation on Camera Module was carried out. Listing 3 shows the code implementation to read output from Camera Module and process the image to detect human face.

```
const cv = require('opencv');
const camera = new cv.VideoCapture(0);

camera.read((err, im) => {
  if (err) {
    throw (err);
  }

  if (im.size()[0] > 0 && im.size()[1] > 0) {
    im.detectObject(cv.FACE_CASCADE, {}, (error, faces) => {
      if (error) {
        throw (error);
      }
      /* return no data if no faces */
      if (faces.length === 0) {
        console.log('no data');
      }

      for (let i = 0; i < faces.length; i++) {
        const face = faces[i];
        if (Object.prototype.hasOwnProperty.call(face, 'x')) {
          im.rectangle([face.x, face.y], [face.width, face.height]);

          /* save image to local machine if face available */
          im.save('../images/pic.jpg');
        } else {
          console.log('no data');
        }
      }
    });
  }
});
```

Listing 3. Node.js subroutine to detect human face

As illustrated in listing 3, OpenCV library was used to record image and process the image to detect human face. The stored image had rectangular box around detected face if found. The image was stored in Joint Photographic Experts Group (JPEG) format.

Node.js version of OpenCV library was providing 90% accurate data depending on photo accuracy, look of people and so on. The image object received from OpenCV response was then passed to TensorFlow backed FaceNet library for further detection to improve the accuracy and optimize the performance. The deep learning algorithm of Tensorflow provided the high accuracy response. The achievement accuracy percentage after the implementation was 99.6 and the image was ready to feed to AWS rekognition service. Listing 4 represents the FaceNet implementation using Node.js to improve the face detection performance.

```
const { Facenet } = require('facenet');

/**
 * faceCheck
 * @param {string} imagePath - path of image in local machine
 * */
const faceCheck = async (imagePath) => {
  const facenet = new Facenet();

  return new Promise(async (resolve, reject) => {
    try {
      const faceList = await facenet.align(imagePath);
      facenet.quit();
      if (faceList.length > 0) {
        resolve(imagePath);
      } else {
        resolve('no data');
      }
    } catch (e) {
      reject(e);
    }
  });
};
```

Listing 4. FaceNet implementation to detect human face

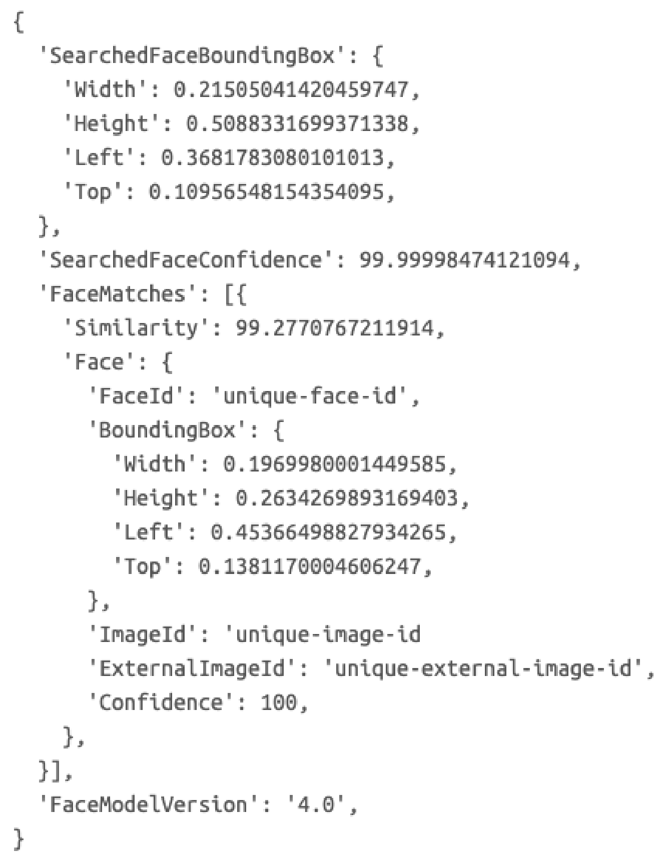
As shown in listing 4, TensorFlow backed FaceNet Node.js implementation library was used to detect face from image stored in local filesystem. The library uses deep learning algorithm to detect face and the results is passed back to the server before passing the image to AWS Rekognition service.

Hence, face detection implementation was successfully carried out with proper testing. Both live camera method and local image method was used to detect face for better performance and more accurate results. The results had to be almost 100% as AWS Rekognition would need at least one face to process otherwise the service throws error back to the system.

### 3.3.3 Phase 3: Amazon Web Services setup

The project has implemented AWS services for most of use cases. AWS provides high range of APIs in the form of CLI and SDK and setting up cloud environment was straightforward. Most of the time, understanding the need and choosing the right service was the challenging part throughout the setup. Despite of being cost efficient, it was important to setup properly to lower the cost and preventing unnecessary API requests.

AWS Rekognition service was used to analyze the image for understanding driver’s sentiments and drowsiness. The service doesn’t need much setup in UI. However, creating face collection with known member images was the first step to use the service. Thus, a face collection was created which was used to compare driver face to identify if the driver was known thereby triggering possible theft alarm. All the user images were passed to the collection which was tagged as a known member and the API would return response with face id, external image id and image id properties. So, these unique ids are the base to compare the image coming from driver seat for the first-time the system captures the image. Figure 19 shows a sample response received from AWS when a new face is added to collection.



```
{
  'SearchedFaceBoundingBox': {
    'Width': 0.21505041420459747,
    'Height': 0.5088331699371338,
    'Left': 0.3681783080101013,
    'Top': 0.10956548154354095,
  },
  'SearchedFaceConfidence': 99.99998474121094,
  'FaceMatches': [{
    'Similarity': 99.2770767211914,
    'Face': {
      'FaceId': 'unique-face-id',
      'BoundingBox': {
        'Width': 0.1969980001449585,
        'Height': 0.2634269893169403,
        'Left': 0.45366498827934265,
        'Top': 0.1381170004606247,
      },
      'ImageId': 'unique-image-id
      'ExternalImageId': 'unique-external-image-id',
      'Confidence': 100,
    },
  }],
  'FaceModelVersion': '4.0',
}
```

Figure 19. Response from AWS Rekognition on adding new face to collection

As shown in figure 19, the response payload contains very important properties along with unique image id, face id and external image id. The ‘FaceMatches’ properties contains similarity index of 99.27 percentage which acts as a parameter of understanding the fact whether the image exists or not.

When at least a human face is detected on the local machine with the help of OpenCV and TensorFlow libraries, the image is passed to AWS Rekognition service for receiving more information about face. The request payload contains collection id, “faceMatchThreshold”, “Image” and “maxFaces” properties. Listing 5 shows the request code script to compare face from existing face collection.

```
const searchFacesByImage = async (image) => {
  const params = {
    CollectionId: 'collection id',
    FaceMatchThreshold: 90,
    Image: {
      Bytes: image,
    },
    MaxFaces: 5,
  };

  return new Promise((resolve, reject) => {
    rekognition.searchFacesByImage(params, (err, data) => {
      if (err) {
        reject(err.stack);
      }
      resolve(data);
    });
  });
};
```

Listing 5. Node.js subroutine for searching face from existing face collection

As shown in listing 5, parameter payload was created with necessary properties including image as base64 encoded bytes and existing face collection id. The request is send once when the motion is detected for the first time. This limit the number of excessive request to the service which is beyond the scope of the project.

Driver sentiment analysis was performed on every possible second with the help of AWS Rekognition face detection API. Image was passed every second to get the emotions and feelings of car driver. The data received was passed to separate function to calculate drowsiness which is one of the major feature of the system. The properties of the response payload were valued based on confidence percentage. Higher the percentage, more would be the accuracy. The processed image was stored in S3 bucket with image id and uniquely identified id which could be traced in the future easily if needed. Listing 6 shows the implementation of AWS face detection API for receiving sentiment of a car driver.

```
const detectFace = async (imageBytes) => {
  const params = {
    Image: {
      Bytes: imageBytes,
    },
    Attributes: [
      'ALL',
    ],
  };

  return new Promise((resolve, reject) => {
    rekognition.detectFaces(params, (err, data) => {
      if (err) {
        reject(err);
      }
      resolve(data);
    });
  });
};
```

Listing 6. Node.js script for detecting facial details

As shown in listing 6, base64 encoded bytes of image and an array of facial attributes, possibly “ALL” or “DEFAULT” are passed as a properties of object parameter to receive facial detail from AWS Rekognition service.

The data receiving from the API as a response payload was stored in RDS PostgreSQL database instance. A database was built to store all facial metadata and possible alert notifications grouped on the basis of user image id. Five different tables were built to store the response properties from AWS APIs. The data in the tables were the source for calculating drowsiness, reading sentiments of the user and understanding driver’s emotion in desired date. Hence the creation date is tracked for each row. The data is also calculated and combined from different tables to fit into visualization tool. Figure 20 illustrates the database design to track the user sentiments and alerts.

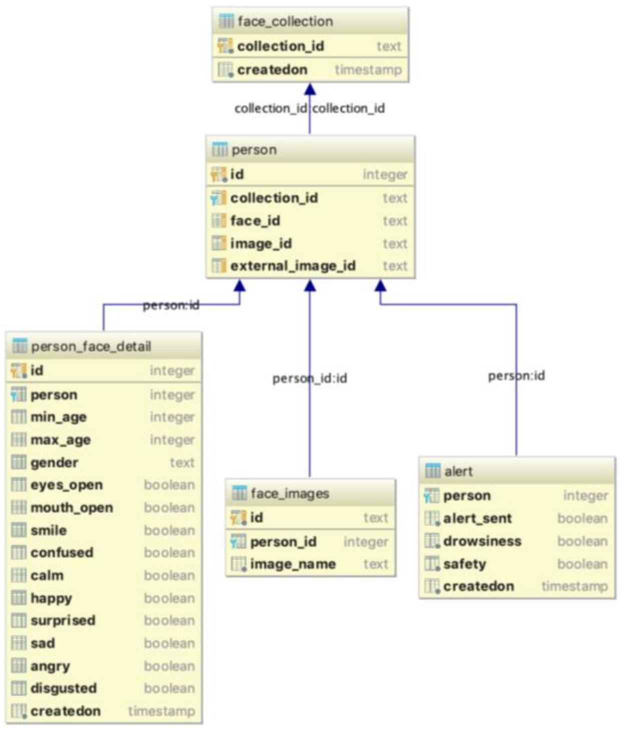


Figure 20. Schema diagram of the database

As illustrated on figure 20, five different tables had been created to store the response received from different API responses. Face collection stores collection for keeping track of known member images. Person table is created to store the known or unknown person to track image id, face id and external image id received from AWS Rekognition API. Person face detail table is used to store the individual sentiments or emotions, received from AWS face detection API. Face images table tracks the images stored in S3 and map the row with person id. Alert table stores all the alerts tracked for each driver. The data is very useful which was used to present the statistics and visual representation in visualization tool.

The implementation of alerts or notification was handled by AWS SES and SNS for sending email alerts and phone message alerts respectively. Phone number and email of members were stored for these services. However, this information can be updated depending on the use case. Sending email and SMS were carried out using AWS SDK. Listing 7 and 8 illustrates the SNS and SES implementation of phone message alert respectively.

```
/**
 * sendAlertMessage
 * @param {string} message
 * @param {string} messageSubject
 * */
const sendAlertMessage = async (message, messageSubject) => {
  const params = {
    Message: message,
    PhoneNumber: phoneNumber,
    Subject: messageSubject,
  };
  return new Promise((resolve, reject) => {
    sns.publish(params, (err, data) => {
      if (err) reject(err);
      resolve(data);
    });
  });
};
```

Listing 7. Node.js subroutine for phone message service implementation using SNS.

```
const sendSecurityAlertEmail = async (email, personName) => {
  const params = {
    Destination: {
      ToAddresses: [
        email,
      ],
    },
    Message: {
      Body: {
        Html: {
          Data: emailBodyTemplate,
          Charset: 'UTF-8',
        },
      },
      Subject: {
        Data: ‘email subject’,
        Charset: 'UTF-8',
      },
    },
    Source: email,
  };

  try {
    await new Promise((resolve, reject) => {
      ses.sendEmail(params, (err, data) => {
        if (err) {
          reject(err);
        } else {
          resolve(data);
        }
      });
    });
  } catch (e) {
    console.error(e);
  }
};
```

Listing 8. Node.js subroutine for email notification implementation using SES.

As shown in listing 7 and 8, phone message and email notification alerts were implemented using AWS SDK which triggers AWS SNS publish API to publish message in provided phone number and AWS SES ‘sendEmail’ API to send email notification to user. SNS publish API request requires parameter object containing message body, message subject and phone number where as SES send email API requires destination email address, message body, subject and email source address.

### 3.3.4 Phase 4: Implementing email and phone notifications

Implementation of phone notification and email notification using AWS was straightforward and required less effort. However, correct email contents and correct phone message content had to be chosen on right context. Email contents were created and tested on proper use case. Separate contents were created for theft alert and drowsiness alerts with image of driver attached as jpeg format at the bottom of an email. In case of theft alert, the email also contained link to add user if the driver is meant to be in member list or driver is known. This prevents triggering theft alert for same user next time, as the system knows that the user is known. The link calls an endpoint which contains a function to add user in the database as well as AWS Rekognition face collection. Figure 21 illustrates a sequence diagram for sending theft alert to user.

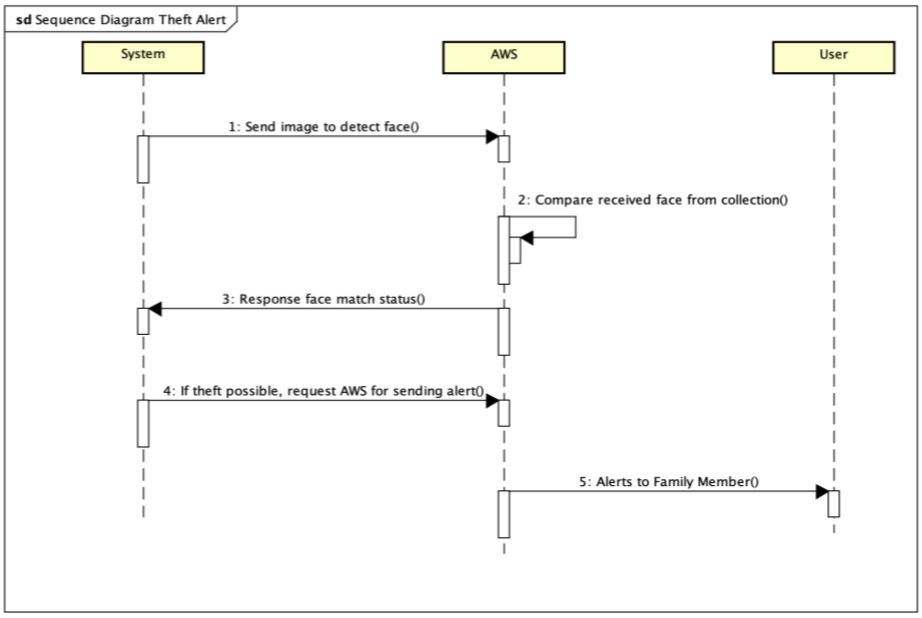
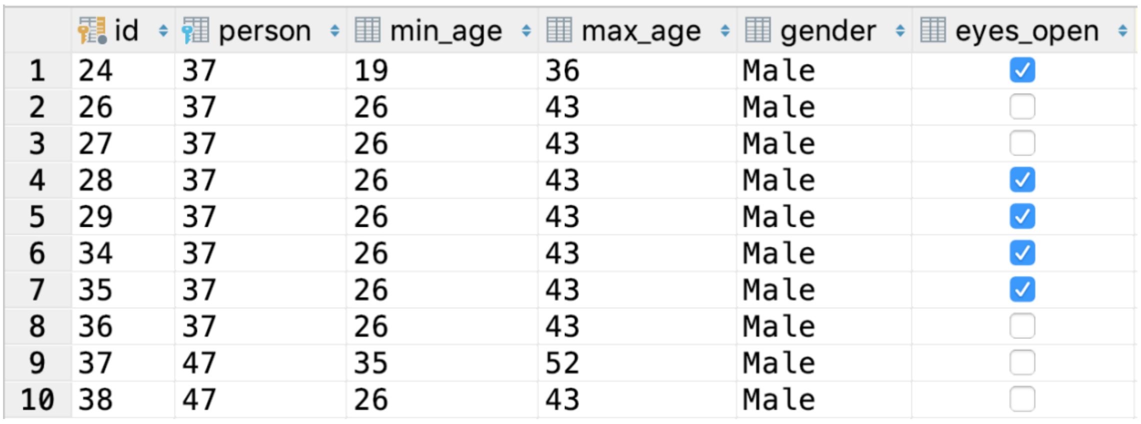

sd Sequence Diagram Theft Alert
System
AWS
User
1: Send image to detect face()
2: Compare received face from collection()
3: Response face match status()
4: If theft possible, request AWS for sending alert()
5: Alerts to Family Member()

Figure 21. Sequence diagram for theft alert to user

As illustrated in figure 21, system sends image captured to AWS which is processed and compared with existing face collection. AWS responses with status of face existence. Then based on the result received, system triggers notification by sending API request to AWS for sending email and SMS to user if needed.

System checks for drowsiness on the basis of eye close status. The captured image will be send to AWS Rekognition API for checking eye close properties. AWS uses face detection API for processing image to detect the facial expressions. The response is then passed to the system which is checked and processed to check for drowsiness condition. The system then triggers notification in order to notify driver and existing driver member to prevent accidents or any risk. Email and SMS notification is used and if needed alerts are sent multiple times if detected. Figure 22 depicts the drowsiness related data stored in table which was received from AWS detection API.



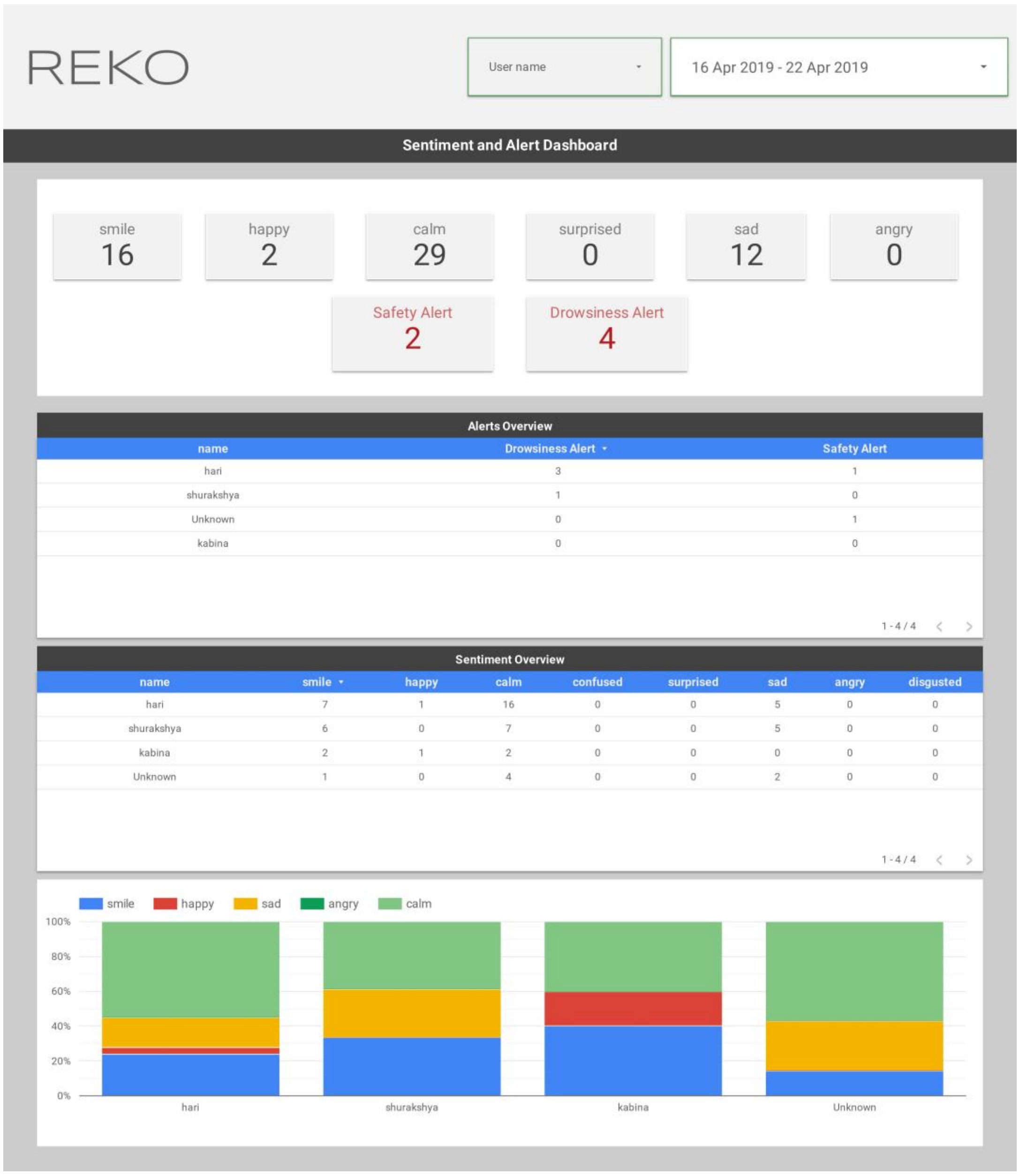
|  | id | person | min_age | max_age | gender | eyes_open |
|---|---|---|---|---|---|---|
| 1 | 24 | 37 | 19 | 36 | Male | ✓ |
| 2 | 26 | 37 | 26 | 43 | Male | ☐ |
| 3 | 27 | 37 | 26 | 43 | Male | ☐ |
| 4 | 28 | 37 | 26 | 43 | Male | ✓ |
| 5 | 29 | 37 | 26 | 43 | Male | ✓ |
| 6 | 34 | 37 | 26 | 43 | Male | ✓ |
| 7 | 35 | 37 | 26 | 43 | Male | ✓ |
| 8 | 36 | 37 | 26 | 43 | Male | ☐ |
| 9 | 37 | 47 | 35 | 52 | Male | ☐ |
| 10 | 38 | 47 | 26 | 43 | Male | ☐ |

Figure 22. Eyes open status sample stored in database table

As shown in figure 22, drowsiness related data is tracked and mapped to person which can be used to send drowsiness alert as well as to visualize in dashboards. Three consecutive ‘True’ value for ‘eyes_open’ properties in the above table means the user was feeling drowsy while driving. Hence, alerts would be sent with the driver information to prevent any possible risk.

### 3.3.5 Phase 5: Data Analysis and Visualization

Graphical representation of data is the key factor to convey message to the viewers. One of the important feature of the project is to provide visual representation of human sentiment in dashboard which is the source of investigations if any misfortune hits the vehicles. The statistics also provide an overview of individual driver which helps to know if the driver is suitable to drive car. Figure 23 shows the visual dashboard built to see sentiments and alerts related data.



Alerts Overview

| name | Drowsiness Alert | Safety Alert |
|---|---|---|
| hari | 3 | 1 |
| shurakshya | 1 | 0 |
| Unknown | 0 | 1 |
| kabina | 0 | 0 |

Sentiment Overview

| name | smile | happy | calm | confused | surprised | sad | angry | disgusted |
|---|---|---|---|---|---|---|---|---|
| hari | 7 | 1 | 16 | 0 | 0 | 5 | 0 | 0 |
| shurakshya | 6 | 0 | 7 | 0 | 0 | 5 | 0 | 0 |
| kabina | 2 | 1 | 2 | 0 | 0 | 0 | 0 | 0 |
| Unknown | 1 | 0 | 4 | 0 | 0 | 2 | 0 | 0 |

Figure 23. Sentiment dashboard for alerts and user's emotions

As shown in figure 23, the dashboard was built to view all the statistics received from face detection API. Most important parameters are focused on the dashboard. Driver’s sentiments and received alerts tables are separated. And graphical representation of user emotion is highlighted at the bottom of the report. The data is dynamic and filters on the header can be altered to update the whole data on the body section of the dashboard.

## 3.4 Testing

Facial recognition system is very powerful technology as it can identify and verify objects from images or videos. Most of the results are accurate and the data are mostly reliable. However, the technology has negative side as well failing to provide 100% accurate results which is termed as False Positives. The number of inaccurate results or a false matched against the total number of compared faces is the measurement method of False Positive Rate. When tested multiple images, the system had also dealt with this sort of results which will be explained in the section.

The test case scenario for vehicle implementation was replicated at home environment. Thus, the testing was carried out at home with all the possible coverages. At first, the program was executed to test single images followed by images of group of people. False positives were seen which was minimized with the additional layer of TensorFlow backed FaceNet library on top of OpenCV. Figure 24 shows the incorrect face detection resulting False Positives.

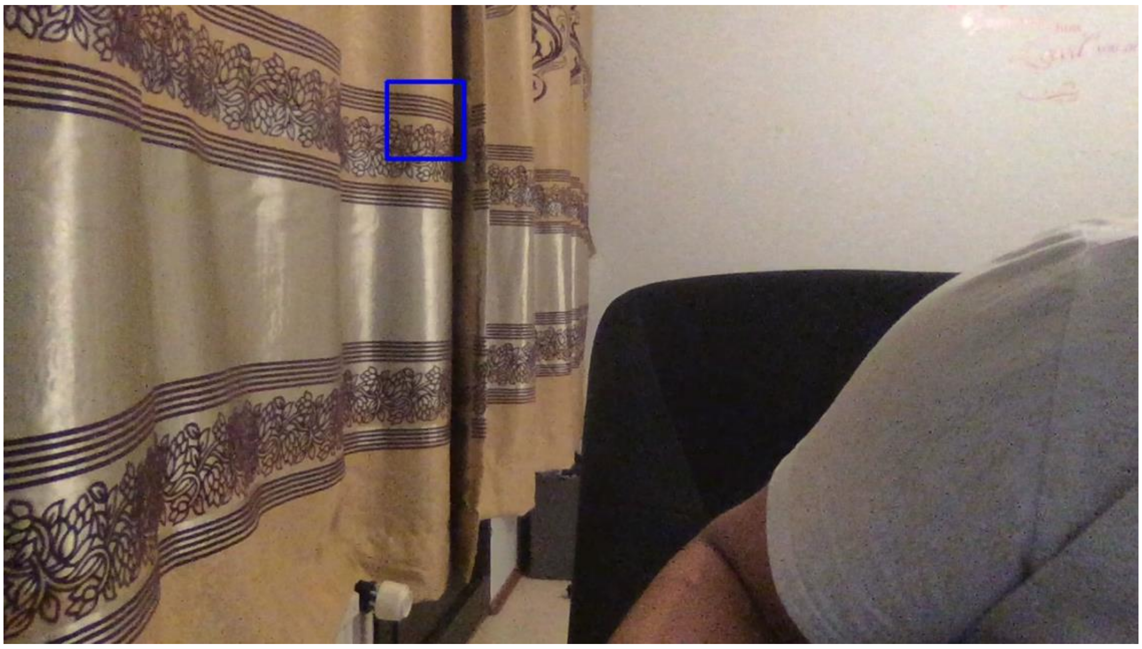
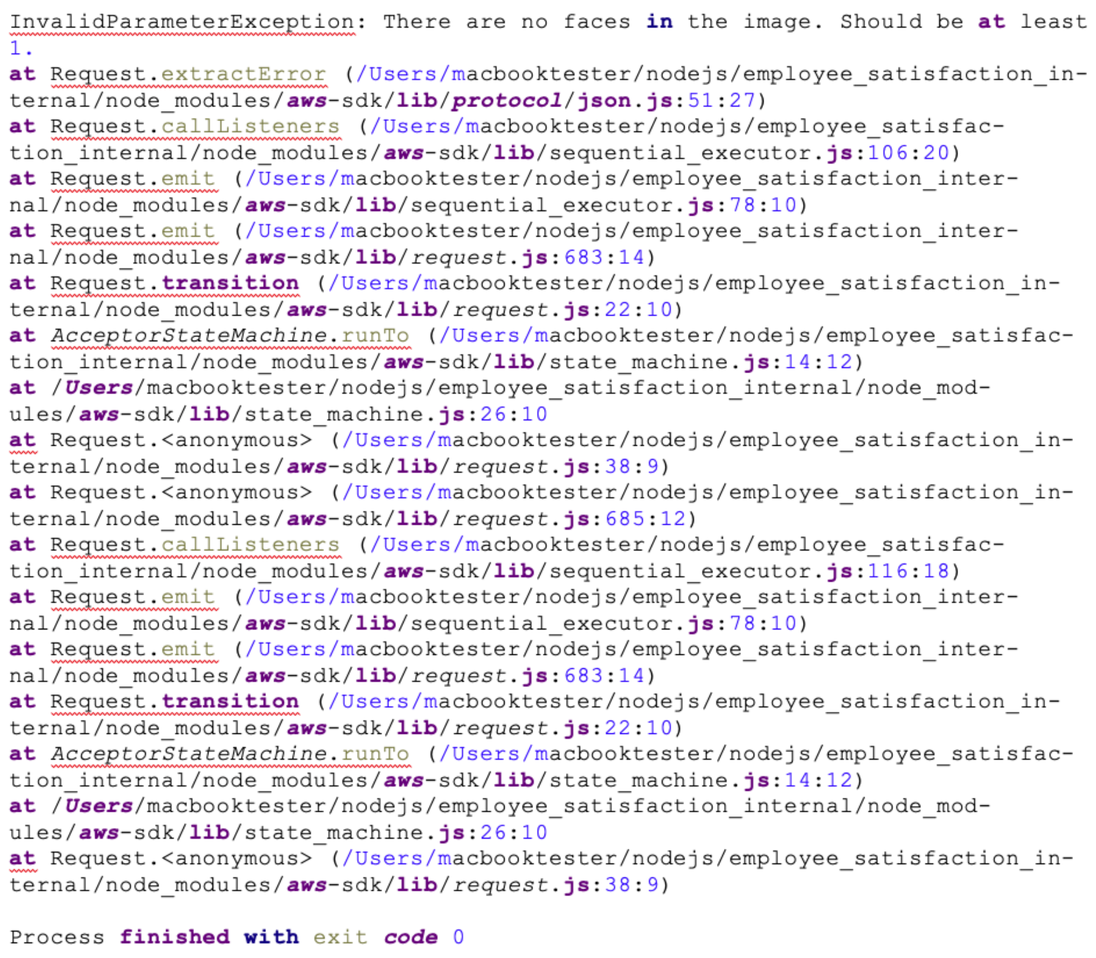


Figure 24. False face detection by OpenCV library example

Figure 24 illustrates the false positives occurrence while using facial detection. The occurrence however is very minimal.

There are several researches going on for optimizing the performances and reducing false positives to provide high accuracy results. Different algorithms are being used and training data are constantly increased with high degree of diversity to accommodate possible variations. Likewise, the system has used TensorFlow backed FaceNet layer to optimize the detection performance. As a result, more accurate images are passed to AWS to analyze image preventing error response which would break the whole execution and program flow. AWS needs image with at least one face to be able to compare faces from existing face collection. Error response from AWS when image is passed without face is presented below in figure 25.



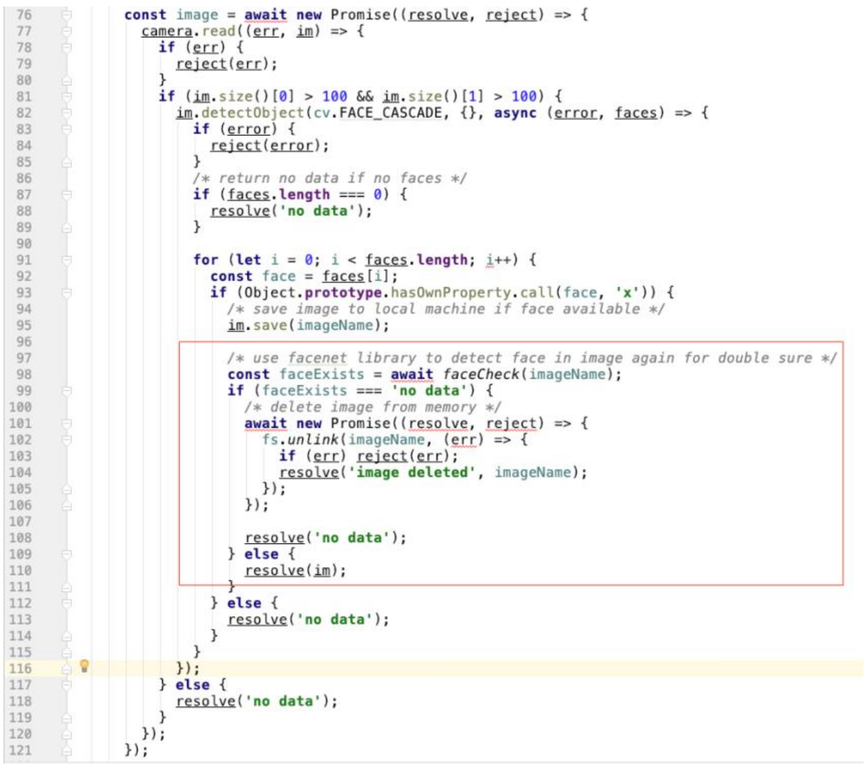
```
InvalidParameterException: There are no faces in the image. Should be at least
1.
at Request.extractError (/Users/macbooktester/nodejs/employee_satisfaction_in-
ternal/node_modules/aws-sdk/lib/protocol/json.js:51:27)
at Request.callListeners (/Users/macbooktester/nodejs/employee_satisfac-
tion_internal/node_modules/aws-sdk/lib/sequential_executor.js:106:20)
at Request.emit (/Users/macbooktester/nodejs/employee_satisfaction_inter-
nal/node_modules/aws-sdk/lib/sequential_executor.js:78:10)
at Request.emit (/Users/macbooktester/nodejs/employee_satisfaction_inter-
nal/node_modules/aws-sdk/lib/request.js:683:14)
at Request.transition (/Users/macbooktester/nodejs/employee_satisfaction_in-
ternal/node_modules/aws-sdk/lib/request.js:22:10)
at AcceptorStateMachine.runTo (/Users/macbooktester/nodejs/employee_satisfac-
tion_internal/node_modules/aws-sdk/lib/state_machine.js:14:12)
at /Users/macbooktester/nodejs/employee_satisfaction_internal/node_mod-
ules/aws-sdk/lib/state_machine.js:26:10
at Request.<anonymous> (/Users/macbooktester/nodejs/employee_satisfaction_in-
ternal/node_modules/aws-sdk/lib/request.js:38:9)
at Request.<anonymous> (/Users/macbooktester/nodejs/employee_satisfaction_in-
ternal/node_modules/aws-sdk/lib/request.js:685:12)
at Request.callListeners (/Users/macbooktester/nodejs/employee_satisfac-
tion_internal/node_modules/aws-sdk/lib/sequential_executor.js:116:18)
at Request.emit (/Users/macbooktester/nodejs/employee_satisfaction_inter-
nal/node_modules/aws-sdk/lib/sequential_executor.js:78:10)
at Request.emit (/Users/macbooktester/nodejs/employee_satisfaction_inter-
nal/node_modules/aws-sdk/lib/request.js:683:14)
at Request.transition (/Users/macbooktester/nodejs/employee_satisfaction_in-
ternal/node_modules/aws-sdk/lib/request.js:22:10)
at AcceptorStateMachine.runTo (/Users/macbooktester/nodejs/employee_satisfac-
tion_internal/node_modules/aws-sdk/lib/state_machine.js:14:12)
at /Users/macbooktester/nodejs/employee_satisfaction_internal/node_mod-
ules/aws-sdk/lib/state_machine.js:26:10
at Request.<anonymous> (/Users/macbooktester/nodejs/employee_satisfaction_in-
ternal/node_modules/aws-sdk/lib/request.js:38:9)

Process finished with exit code 0
```

Figure 25. Error response from AWS face search API without face in requested image

As shown in figure 25, AWS throws error with a message that the image passed to process must contains at least a face. The API request used to search image from collection was ‘searchFacesByImage’. This shows that the image must provide 100% accuracy face detection rate before serving it to AWS Rekognition API.

Preventing error from AWS API was done by adding one layer of additional check after OpenCV detect image from camera. FaceNet was used to check the facial presence on an image right after the result from OpenCV library. These two combination layers help provide more accurate facial detection results and errors from AWS was completely deduced which prevented the system from failure. Figure 26 illustrates the extra layer code implementation to optimize face detection performance.



```
const image = await new Promise((resolve, reject) => {
  camera.read((err, im) => {
    if (err) {
      reject(err);
    }
    if (im.size()[0] > 100 && im.size()[1] > 100) {
      im.detectObject(cv.FACE_CASCADE, {}, async (error, faces) => {
        if (error) {
          reject(error);
        }
        /* return no data if no faces */
        if (faces.length === 0) {
          resolve('no data');
        }

        for (let i = 0; i < faces.length; i++) {
          const face = faces[i];
          if (Object.prototype.hasOwnProperty.call(face, 'x')) {
            /* save image to local machine if face available */
            im.save(imageName);

            /* use facenet library to detect face in image again for double sure */
            const faceExists = await faceCheck(imageName);
            if (faceExists === 'no data') {
              /* delete image from memory */
              await new Promise((resolve, reject) => {
                fs.unlink(imageName, (err) => {
                  if (err) reject(err);
                  resolve('image deleted', imageName);
                });
              });

              resolve('no data');
            } else {
              resolve(im);
            }
          } else {
            resolve('no data');
          }
        }
      });
    } else {
      resolve('no data');
    }
  });
});
```

Figure 26. Optimization of facial detection with FaceNet library layer

As presented in Figure 26, FaceNet library was used to check the facial existence in an image. The image will be stored and passed to AWS API for further processing only if the image contains human face. Otherwise, the image will be deleted from local file system and further execution will be cancelled.

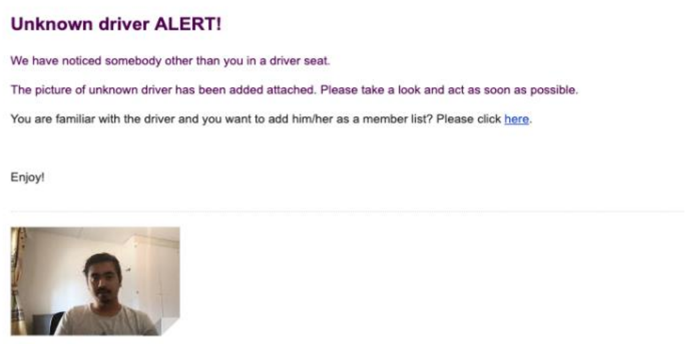
The image detection was carried out without adding any faces in face collection for generating theft alert. The system triggered email notification alert with an attachment of image and SMS notification with a link of image to be able to download. Listing 9 shows the result from AWS face search API when unknown face is passed.

```
{
  SearchedFaceBoundingBox: {
    Width: 0.21161779761314392,
    Height: 0.5229141712188721,
    Left: 0.4211474359035492,
    Top: 0.11873599141836166
  },
  SearchedFaceConfidence: 99.99998474121094,
  FaceMatches: [], // No Matching Face Found
  FaceModelVersion: '4.0'
}
```

Listing 9. AWS face search API response when unknown image passed

As shown in listing 9, the response object received from AWS API contains ‘FaceMatches’ with empty array. This value is read by the system to generate Email and SMS notification.

Email content is built with plain Hypertext Markup Language (HTML) with JPEG attachment of user. Figure 27 shows the example image received as email alert in theft case scenario.



**Unknown driver ALERT!**

We have noticed somebody other than you in a driver seat.

The picture of unknown driver has been added attached. Please take a look and act as soon as possible.

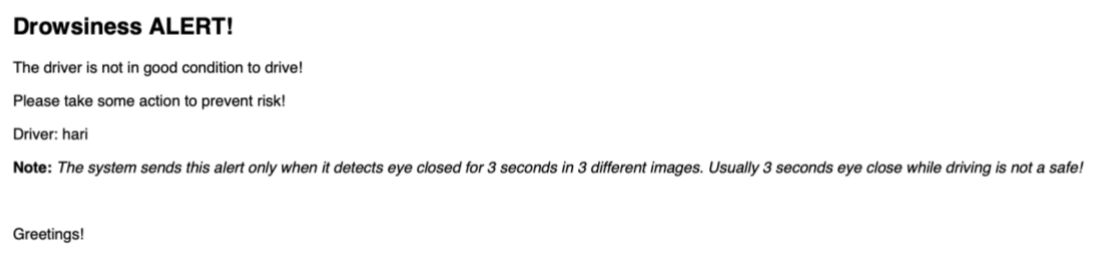
You are familiar with the driver and you want to add him/her as a member list? Please click here.

Enjoy!

Figure 27. Email alert response example when theft is detected

As shown in figure 27, the driver member receives email right away as soon as theft possibility is checked by the system. This ensures the car is on safe hand and owner can easily take further action based on image. Further detail of the driver is stored in database which is shown in dashboard.

Drowsiness was also tested with Amazon face detect API based data. The system sent three different images with eye closed face within three seconds. The alarm was triggered if API response eye close status true for three images. This request was triggered each second to ensure the performance. AWS face detection API result was 100% accurate on our test cases. Example email alerts of drowsiness is shown in figure 28.


**Drowsiness ALERT!**

The driver is not in good condition to drive!

Please take some action to prevent risk!

Driver: hari

**Note:** *The system sends this alert only when it detects eye closed for 3 seconds in 3 different images. Usually 3 seconds eye close while driving is not a safe!*

Greetings!

Figure 28. drowsiness alert email example based on AWS API response

As presented in figure 28, email notification is sent as soon as the system detects sign of drowsiness. The alert also contains driver name as well so that responsible authority can take an action as soon as possible.

# 4 Discussion

## 4.1 Results

A Raspberry Pi based car safety and security system was built with the integration of Deep Learning technology and cloud services. The potential of facial recognition technology and its scope in various use cases were explored. Face recognition libraries and other cloud services were utilized to demonstrate significance of face detection system in vehicles to minimize road causalities. The implementation was tested to evaluate accuracy of the face recognition technologies and Api while working with images and videos.

The goal of the project was to build a Raspberry Pi car safety system with facial recognition technology. Sentiment analysis and drowsiness detection were the supposed key feature of the system to ensure safety and security of driver. The application utilized AWS Rekognition on top of other face recognition libraries to analyze images for face detection and sentiment analysis. However, the results were not completely accurate due to drawbacks in used face recognition libraries and machine learning stack. Thus, combination of two libraries to minimize the flaws was fruitful for generating accurate results. Finally, a working prototype of future car safety system with Raspberry Pi was built.

## 4.2 Challenges

Throughout the final year project, several challenges were encountered. As face recognition technology is advance and provides huge number of algorithms to process images and videos, the major challenge was to decide the appropriate technology and procedure. The effect of false positives observed while implementing OpenCV library required additional application of another library, FaceNet increasing implementation time and extra tests. Likewise, error occurrence due to non-face images in request payload while calling AWS recognition face search API, needed application logic to scrutinize request payload.

Moreover, different environments for development and testing that is local environment and Raspberry Pi environment respectively needed different initial setups and pre-installed libraries for the execution of program. Since the project used cloud services which charge on pay-per-use model, the number of requests were limited for cost minimization until the completion of project. Although several obstacles were faced during the implementation phase, the final product includes all the features and application logic with sufficient testing.

## 4.3 Future Development

The device built was prototype and minimum viable product of the concept. Integration of device in car as in-built feature would require further developments regarding compatibility of the device and system. Features such as alerts are provided via email and SMS in prototype. Automation of alarm or speaker for alerts would improve the quality of system as well instant alarm will make the system reliable. In the current implementation, a separate dashboard is required to view the statistics and data from the past. Real-time dashboard view in car and sentiment analysis will provide smooth experience to the user.

Furthermore, data collected from recognition and sentiment analysis can be utilized to automate sound and music media for the personalization of car. AI technologies provided by cloud like text to speech or vocal bot for providing suggestion based on real time sentiment analysis can be included in the system. Implementation of furthers features like automated notification to local authority and health institute in case of unusual incidents or causalities will improve the system. Face recognition implementation can be further improved to detect not only the image of person but objects such as blood or crash to improve current notification system. At last, non-functional requirements like data protection and security to avoid leakage of personal data, scalability and availability of the device as well as optimization for the response time should be considered for the future implementation and development.

# 5 Conclusions

The purpose of the final year project was to build a security system utilizing technologies like face recognition, cloud services and IOT. The system was to be used for drowsiness detection and face identification for safety of both user and car. With respect to given goals, a Raspberry Pi car safety and security system was successfully developed using Camera Module and PIR motion sensors along with integration of facial recognition technology like OpenCV and AWS Rekognition. Drowsiness detection, face identification and sentiment analysis were implemented using the face recognition libraries, Cloud Api and visualization tool. Node.js binding of TensorFlow backed FaceNet library was used to reduce the false positives and increase the accuracy of the results.

The built system successfully identifies face and detects drowsiness and sends alerts on the basis of obtained data from used technologies. In addition, the system was successful to collect and visualize data.

The technology is expected to be integrated as built-in feature in future cars and possibly all vehicles. To prepare the system for implementation in car, further development should incorporate advance notifications system and automation of alarms and speaker and integration of real-time visualization dashboard.

## References

1 Statistics Finland. Statistics on road traffic accidents [online]. Helsinki, Finland: Transport and Tourism; 19 March 2019.
URL: https://www.stat.fi/til/ton/2019/02/ton_2019_02_2019-03-19_en.pdf
Accessed 20 March 2019.

2 Federal Bureau of Investigation, U.S. Department of Justice. Uniform Crime Reports [online]. Unites States; 2018.
URL: https://www.iii.org/fact-statistic/facts-statistics-auto-theft
Accessed 20 February 2019.

3 Himanshu, Dhawan S., Khurana N. A Review of Face Recognition. International Journal of Research in Engineering and Applied Sciences 2012;2(1):835-846.

4 Stan Z. Li, Anil K. Jain. Handbook of Face Recognition. New York, United States: Springer; 2005.

5 Dawa T., Vijayalakshmi N. A Comparative Review on Different Methods of Face Recognition. Oriental Journal of Computer Science and Technology 2017;10(1):228-231.

6 Tolba A.S., El-Baz A.H., El-Hardy A.A. Face Recognition: A literature Review. International Journal of Signal Processing 2014;2(2):88-103.

7 Wiskott L., Fellous J.M., Kruger N., Von Der Malsburg C. Face Recognition by Elastic Bunch Graph Matching. IEEE Transaction on Pattern Analysis and Machine Intelligence 1997;19(7):775-779.

8 Wiskott L., Fellous J.M., Kruger N., Von Der Malsburg C. Face Recognition by Elastic Bunch Graph Matching. In: Jain L.C.et al. editor. Intelligent Biometric Techniques in Fingerprint and Face Recognition. Florida, United States: CRC Press; 1999. p.1-23.

9 Karungaru S., Fukumi M., Akamatsu N. Face Recognition using Genetic Algorithm based Template Matching. International Symposium on Communications and Information Technologies [Online] 2004; 2():1252-1257.
URL: https://ieeexplore-ieee-org.ezproxy.metropolia.fi/stamp/stamp.jsp?tp=&arnumber=1413920
Accessed 15 February 2019.

10 Stojilkovic J., Face Recognition using Neural Network [online]. Faculty of organizational Science. Beograd, Serbia: University of Belgrade.
URL:
http://neuroph.sourceforge.net/tutorials/FaceRecognition/FaceRecognitionUsingNeuralNetwork.html
Accessed 20 April 2019.

11 Hemlatha G., Sumathi C.P. A Study of Techniques for Facial Detection and Expression Classification. International Journal of Computer Science and Engineering Survey 2014;5(2):27-37.

12 Rowley H.A., Baluja S., Kanade T. Neural Network-Based Face Detection. IEEE Transactions on Pattern Analysis and Machine Intelligence 1998;20(1):23-38.

13 Gokhale P., Bhat O., Bhat S. Introduction to IOT. International Advanced Research Journal in Science, Engineering and Technology 2018;5(1):41-44. DOI: 10.17148/IARJSET.2018.517

14 GSM Association. Understanding the Internet of Things (IOT) [online]. London, United Kingdome: GSMA Head Office; July 2014.
URL: https://www.gsma.com/iot/wp-content/uploads/2014/08/cl_iot_wp_07_14.pdf
Accessed 3 March 2019.

15 Perwej Y., Omer K.M., Sheta E.O., Harb A.M.H., Adrees M.S. The Future of Internet of Things (IOT) and its Empowering Technology. International Journal of Engineering Science and Computing 2019;9(3):20912-20203.

16 Hurwitz J., Kaufman M., Halper F. Cloud Services for Dummies. New Jersey, United States: John Wiley & Sons, Inc.; 2012.

17 Grace L. Basics about Cloud Computing. Software Engineering Institute [online]. Pennsylvania, United States: Carnegie Mellon University; September 2010.
URL:
https://resources.sei.cmu.edu/asset_files/WhitePaper/2010_019_001_28877.pdf
Accessed 20 February 2019.

18 Jagirdar S., Venkata K., Reddy S., Qyser A.M. Cloud Computing Basics. International Journal of Advanced Research in Computer and Communication Engineering 2012;1(5):343-347.

19 Jackovich J., Rizards R. Machine Learning with AWS. Birmingham, United Kingdom: Packt Publishing; 2018.

20 Amazon Web Services. Amazon Rekognition Developer Guide [online]. Washington, United States: Amazon Web Services, Inc.; 2019.
URL: https://docs.aws.amazon.com/rekognition/latest/dg/rekognition-dg.pdf#what-is
Accessed on 20 February 2019.

21 Amazon Web Services. Amazon Simple Email Service Developer Guide [online]. Washington, United States: Amazon Web Services, Inc.; 2019.
URL : https://docs.aws.amazon.com/ses/latest/DeveloperGuide/ses-dg.pdf
Accessed on 20 February 2019.

22 Amazon Web Services. Amazon Simple Notification Service Developer Guide [online]. Washington, United States: Amazon Web Services, Inc.; 2019.
URL: https://docs.aws.amazon.com/sns/latest/dg/sns-dg.pdf
Accessed on 20 February 2019.

23 Anwaar W., Shah A.M. Energy Efficient Computing: A comparison of Raspberry PI with Modern Devices. International Journal of Computer and Information Technology 2015;4(2):410-413.

24 Raspberry PI Documentation. Raspbian [online]. Raspberry PI Foundation; 2017.
URL: https://www.raspberrypi.org/documentation/raspbian/
Accessed on 25 February 2019.

25 Google Data Studio. Data Studio Community Visualizations Developer Preview [online]. Google LLC; March 2019.
URL: https://developers.google.com/datastudio/visualization/developer-preview/
Accessed on 20 March 2019.

26 Silberschatz A., Korth H.F., Sudarshan S. Database System Concepts. New York, United States: Mc Graw Hill; 2011.

27 Tarus Z. PIR Sensor-based Security System. Helsinki, Finland: Helsinki Metropolia University of Applied Sciences; 2017.

28 OpenCV Documentation. Introduction [online]. OpenCV org.; April 2019.
URL: https://docs.opencv.org/4.1.0/d1/dfb/intro.html
Accessed on 15 April 2019.

29 Raspberry PI Projects. Setting up your Raspberry Pi [online]. Raspberry PI Foundation; 2017.
URL: https://projects.raspberrypi.org/en/projects/raspberry-pi-setting-up/4
Accessed on 20 February 2019.

30 Raspberry PI Documentation. GPIO [online]. Raspberry PI Foundation; 2017.
URL: https://www.raspberrypi.org/documentation/usage/gpio/
Accessed on 20 February 2019.