Date: 01.02.2019

Research Project (Thesis) MOD3-03

Feature Development Kit [FDK] – Plugin Development and Continuous Integration Maintenance

Submitted by:

Ahmad Hassan Mirza [7104716]

Embedded Systems for Mechatronics, FH Dortmund

[ahmad.mirza001@stud.fh-dortmund.de](mailto:ahmad.mirza001@stud.fh-dortmund.de)

Supervisor:

Prof. Dr. Jörg Thiem

# Abstract

According to National Institute on Deafness and Other communication Disorders(NIDCD) 7.7 percent of children between the ages of 3 to 17 in the US alone have had a disorder related to voice, speech, language or swallowing with in the duration of 12 months of the report being published[1]. The idea behind project AVATAR is to develop a standalone logopedics system which can be utilized for speech therapy for children suffering from these speech impediments. For the first part of this project the objective is to extract visual information about the shape and movement of the lips and the jaw that is associated with the pronunciation of the word and attempt to infer the word that has been said regardless of any accent or mispronunciation present. This could then be used as a base and stepping stone for further development towards a complete system for speech therapy.

With the advancements in mobile devices and their wide availability and popularity, android platform has been chosen for the development of the prototype application. The application developed employs several image processing techniques to extract the relevant information which is then passed to a remote computer for further processing that is required to infer the words or sentence said using only visual data (lip reading).

The scope of the project involves research and evaluation of the useable image processing methods, development of the application using the results from phase-1, performance optimizations to achieve a near real-time processing and evaluation and analysis of the results.

**Keywords:** Image Processing, face detection, facial landmarks detection, speech therapy, logopedics, OpenCV, dLib.

# Table of Contents

[Abstract i](#_Toc6433178)

[Table of Figures iv](#_Toc6433180)

[Abbreviations v](#_Toc6433181)

[1. Introduction 1](#_Toc6433182)

[1.1. Motivation 2](#_Toc6433183)

[1.2. Goals 3](#_Toc6433184)

[2. Existing System 4](#_Toc6433185)

[2.1. Software Sharing 4](#_Toc6433186)

[2.2. Shared Build Process 6](#_Toc6433187)

[2.2.1. Basic Software Configuration 6](#_Toc6433188)

[2.2.2. Registration of Files 6](#_Toc6433189)

[2.2.3. Basic Software Configuration Toolkit 6](#_Toc6433190)

[2.2.4. Motor management Diesel Gasoline Build 6](#_Toc6433191)

[3. Foundation 7](#_Toc6433192)

[3.1. Requirements 7](#_Toc6433193)

[4. Concept 8](#_Toc6433194)

[4.1. Components 9](#_Toc6433195)

[4.2. Feature Development Kit 11](#_Toc6433196)

[4.3. Feature Store Platform 12](#_Toc6433197)

[4.4. Virtual Machine Environment 14](#_Toc6433198)

[4.5. Continuous Integration 16](#_Toc6433199)

[4.5.1. Machine to Machine Mapping 17](#_Toc6433200)

[4.6. FDK Plugins 18](#_Toc6433201)

[4.7. State Machine 20](#_Toc6433202)

[4.8. Sequence 21](#_Toc6433203)

[4.9. Configuration and Maintenance 22](#_Toc6433204)

# Table of Figures

[Figure 1.1: Interfaces in the AUTOSAR Layered SW Architecture [2] 1](#_Toc6433692)

[Figure 2.1: Activities involved in shared software configuration 5](#_Toc6433693)

[Figure 4.1: Activites involved in shared software configuration by FDK 8](#_Toc6433694)

[Figure 4.2: Overall component diagram 9](#_Toc6433695)

[Figure 4.3: FDK concept 11](#_Toc6433696)

[Figure 4.4: Feature store platform 13](#_Toc6433697)

[Figure 4.5: FSP usecase diagram 13](#_Toc6433698)

[Figure 4.6: Steps for shared build process 15](#_Toc6433699)

[Figure 4.7: Concept of VM in FDK 15](#_Toc6433700)

[Figure 4.8: Concept of VM for multiple customers in FDK 16](#_Toc6433701)

[Figure 4.9: Components for upload and download in FSP 17](#_Toc6433702)

[Figure 4.10: Block diagram of FDK plugins 18](#_Toc6433703)

[Figure 4.11: State machine diagram of FDK process 20](#_Toc6433704)

[Figure 4.12: Sequence diagram of FDK process 22](#_Toc6433705)

[Figure 5.1: FDK plugins by Bosch 23](#_Toc6433706)

[Figure 5.2: Types of FDK plugins 23](#_Toc6433707)

[Figure 5.3: Block diagram for adapter\_fdk\_TXCAN 24](#_Toc6433708)

[Figure 5.4: Block diagram for CAN\_Config\_Check 26](#_Toc6433709)

[Figure 5.5: Block diagram for DSM\_Check 28](#_Toc6433710)

[Figure 5.6: Block diagram for EEP\_Check 29](#_Toc6433711)

[Figure 6.1: Text file output for negative test of CAN\_Config\_Check 32](#_Toc6433712)

[Figure 6.2: Text file output for positive test of CAN\_Config\_Check 33](#_Toc6433713)

[Figure 6.3: Text file output for negative test of DSM\_Check 33](#_Toc6433714)

[Figure 6.4: Text file output for positive test of DSM\_Check 34](#_Toc6433715)

[Figure 6.5: Text file output for negative test of EEP\_Check 34](#_Toc6433716)

[Figure 6.6: Text file output for positive test of EEP\_Check 34](#_Toc6433717)

# Abbreviations

API – Application Programming Interface

JNI – Java Native Interface

NDK – Native Development Kit

IDE – Integrated Development Environment

UI – User Interface

ROI – Region of Interest

XML – eXtensible Markup Language

ROS – Robot Operating System

# Introduction

According to the findings of National Health Interview Survey [²] nearly 8 percent of the children aged 3-17 in the US had a communication disorder with in the past 12 months of the study, and among these children only 55% received any intervention service during the time period of 12 months. These statistics show that the problem of speech impediment is very prevalent in children and young people while almost half of the people suffering from this problem receive any help.

The motivation behind this project is to develop a standalone system which can be used by laymen without any extra help to address certain speech impediments, and the most important part is that the system should be easily and widely available for everyone.

In order to achieve this objective, a smartphone application on the android platform has been chosen as the medium for the first phase of the project. The reason behind this is that in recent years smart phone industry has seen a large boom and by 2020 there are expected to be 3.5 billion smartphone users worldwide [3], apart from this the advancement in technology has made the smart phones very capable and tasks that required specialized equipment before can be performed by a simple application running on the smart phone. The introduction of wearable tech like smart watches and other health monitors that can be connected to a smartphone via Bluetooth and the data analysed by a paired app have also made diagnostics and health monitoring very easy and does not require highly skilled medical personnel to perform these tasks as in the past.

The first part of the project deals with the development of the android application. This part has further been divided into four phases for better management and implementation. *Phase-1* was research and performance evaluation of the methods which could be used to develop the application. *Phase-2* was the development of the application to the point that it could use the on-board camera to capture live images and perform image processing tasks to extract useful information. *Phase-3* was to establish real-time communication between the mobile application and a remote PC to transfer the extracted data to a PC to perform the resource extensive processing that is required to infer the words from lip reading. *Phase-4* was implementation of the Matlab scripts running on the remote PC using Hidden Markov Model and its training using data set developed using the android application. In the future the tasks done in phase-3 and 4 would be moved to be performed on the device itself or on the cloud to enable portability.

For image processing OpenCV’s android SDK is used in conjunction with dlib c++ library to extract facial landmarks. The details of the implementation are provided in the following sections.

# Theory

* 1. Literature Review

Digital Image Processing is the collection of techniques used to manipulate images to extract some results that might be of interest out of the image for example object detection, classification etc. Image processing itself can be further sub-divided into sub.-classes lie image analysis, image restoration, image enhancement etc. In the past decades the usage of these digital image processing techniques in production applications has increased tenfold. The applications of image processing range from medical technology, automated driving to space technology.

The following sections provide a brief overview of the state of art of the techniques that have been used or could be used in this project.

* 1. Image Processing Techniques – Theory

In the scope of this project object detection and classification techniques have been used to extract facial features from the images captured from the device camera which are then further used for image analysis using machine learning techniques:

* + 1. Face Detection

Face detection is the first step towards all facial analysis algorithms which in turn play a huge role in today’s technologies that interact with human’s; these facial analysis algorithms can range from facial recognition, authentication/security applications, lip reading and expression classification. Face detection is an easy task for human beings but for machines considering all the variable could prove to be a difficult task. The scale, orientation, pose and the lighting conditions of the image are among the many variables that need to be accounted for in order to perform a successful facial detection. There are multiple ways that have been a topic of research for performing face detection. They can be divided into four main methods: (i) Feature-based (ii) Appearance based (iii) Knowledge based (iv) Template matching. Out of these the more popular ones are feature based and knowledge based. For the first technique face detection can be done by using self-defined features in conjunction with HAAR cascade machine learning algorithm which is the techniques used in OpenCV’s face detection algorithm. The other technique is to use neural networks and train them on a huge dataset of faces, which are readily available now, one of the most influential face databases is the FERET or COLOR FERET database [5]. Tensor Flow is an open source platform for machine learning which is commonly used for research and applications involving deep neural networks. Tensor flow allows users to create dataflow graphs and the training API extracts features automatically from the training dataset which allows for detection and classification of objects of interest.

In 2001 P.Viola and M.Jones published a paper describing what is now known as Viola-Jones (V-J) Object detection framework. V-J algorithm is one of the first real time object detection algorithms and provides very high detection rate and very less percentage of false positives.

The algorithm has some limitations related to the pose of the face but the classifiers can be trained to recognize slight tilts in the face. V-J detector utilizes Haar-like features that are a scalar product between the image and some Haar-like templates. Figure XX shows the Haar-like templates used in the V-J algorithm and also employed by OpenCV. The Haar features signify that almost all the human faces share similar visual outlooks for example the bridge of the nose is lighter than the eyes which can be defined by the Haar template figure XX(b-1) and the eyes region is darker than the upper cheeks and which is defined by inversion of Haar template of fig. (XX-a-1). This way the regions of interest i.e. different features on a face can be detected using the gradients in pixel intensities. The V-J algorithm uses a 24x24 window as the base window size to start the evaluation of these features which gives rise to up to 160,000+ features to be calculated. The steps put forward by the algorithm brings this immense calculation down to a smaller size. The steps involved after selection of appropriate Haar-Feature templates in V-J algorithm are [7]:

* Image Integral and feature extraction
* AdaBoost Learning
* Cascading Classifiers

The first step of the V-J algorithm is to obtain an integral image from the original image. The integral image provides a quick and efficient way of calculating the sum of pixel values in a subset of a pixel grid which contain the pixels located above and to the left of x, y inclusive (with x, y being the coordinates of the pixel of interest) [8]. This can be represented mathematically by the following equation:

Where ii(x, y) is the integral image and i(x, y) the original image. The computation of integral image can be done in one iteration over the entire image. The advantage of computing an integral image is that the sum of any rectangle with in the image can be calculated very quickly and efficiently by the following equation.

The second part of the algorithm is application of AdaBoost to remove unnecessary features. For example the same Haar-feature template that applies to the eyes and the bridge of the nose can also apply to other parts of the face and using AdaBoost algorithm a best threshold can redefined which classifies the faces while selecting the features with minimum error rate so that the features that most accurately classify the faces and non-face parts of the image are selected. V-J algorithm combines a small number of these features to form an effective classifier, by employing a weak learning algorithm which determines the optimal threshold classification function so that a minimum number of misclassifications occur. All the images used in the training the classifiers are given an equal weight and every feature is applied on all the training images. After each iteration of classification weights assigned to misclassified images is increased and the iteration of classification is repeated. After each iteration new error rates and weights are calculated and assigned until the required accuracy is achieved. After the application of AdaBoost algorithm is successfully completed one feature is selected out of the 160,000+ features for each iteration.

The classifiers obtained from the AdaBoost stage are not strong enough to be used individually for object detection but the V-J algorithm suggests the work around to this by using the classifiers obtained in a cascade. Each stage of classifiers checks the given stage as to whether it definitely does not contain a face or maybe it contains a face. As soon as a sub window is classified as not containing a face it is discarded. The sub- window classified as maybe is then passed to the next stage in the cascade. It can be deduced that the more stages a sub-window passes the higher is the probability of it \_containing a face. All the stages in this cascade are obtained by training classifiers by using AdaBoost algorithm and then adjusting the threshold to minimize false negatives. OpenCV already provides a cascade classifier for face, eyes and other features and also provides a way to train custom classifiers for other objects that a user might want to detect [9].

The other method to perform face or object detection is to use neural networks. A lot of research is available on this technique and some trade-offs have been listed between using template based detection and neural networks. HA.Rowley et.al show in their paper that although neural network based object detection can enhance the results of the face detection system but it causes an increase in the execution time [10]. Although this paper was published in 1998 and with the evolution of neural networks e.g. tensor flow, this trade-off might not exist at this time and a new evaluation need to be performed. The system described in this paper is a two stage detection system. A filter is applied on a 20x20 pixel region of the image which generates an output from 1 to -1 based on the presence of a face in the image, in case the faces or the image is larger than the filter size the image is reduced in each repetition and the filter is applied on every point of the image on each iteration. In the second stage overlapping detections are merged, resulting in removal of false detections in neighbouring areas of the true faces, to further reduce the number of false positives, multiple networks can be used and their outputs are arbitrated to produce a final output.

M.Wu et. al. developed a system for face detection and feature localisation based on case structured classifier, template matching, feature space analysis method and AdaBoost. They also incorporated human face super resolution algorithm for helping in detection of faces in low-resolution images. The system performs image pre-processing with histogram equalization, the pre-processed image is fed to the face detector, which is based on the V-J’s cascade framework for robust and rapid object detection. The detector returns the number of faces and the locations of all the faces in the image. A resolution threshold of 64\*48 has been defined, if the faces are larger than this threshold value then they are passed on to the organ detection system otherwise they employ the face super-resolution algorithm to get a higher resolution image. Before this algorithm is applied a rough location of eyes needs to be detected, if a non-human face is detected in this step then that result could be discarded making the algorithm more robust. For organ location (eye location), this part is also interesting for our application for landmarks detection, the team used 8-eye-pair template as per the minimum relevant match theory. The team report an accuracy of 95.8% for eye detection, 89.8% for nose, 87.3% for mouth and 91.7% for chin detection [11].

Anamika Singh et.al. Proposed a system for face and eyes detection using Sobel edge detection and morphological post processing steps for extracting eyes, by analysing the eye regions and the shape of the region i.e. the distance from the left and right of the face and given the fact that the eyes are generally in a straight line on the face. They report a 92.5% success rate for eyes extraction from the IMM frontal face database [12].

Another algorithm for face detection and landmark localization is presented by Prathap Nair et.al [13]. The proposed algorithm detects and segments 3-d face meshes and localizes landmarks based on Point Distribution Model which removes any dependency on prior knowledge of texture, pose or orientation. They perform face detection by analysing the transformation between model points and points of interest in the sample image based on the deviation of the parameters from the mean model, afterwards landmark localization is performed on the extracted face based on the transformation that gives the minimum deviation of the model from the mean shape. They reported a 99.6% accuracy for face detection.

* + 1. Facial Landmarks detection

Landmarks detection in dlib and google.

* + 1. *Lip Reading Techniques*

Overview of pre-existing lip reading techniques.

* 1. Hidden Markov Model

The Hidden Markov Model (HMM) is a statistical model for a system with unobservable or hidden states. In simpler versions of Markov models for example the Markov Chain, the state is visible, and the only parameters to be considered are state transition probabilities. In the HMM since the states are not visible, instead the output dependent on the state is accessible. So each state has a probability distribution for possible outputs.

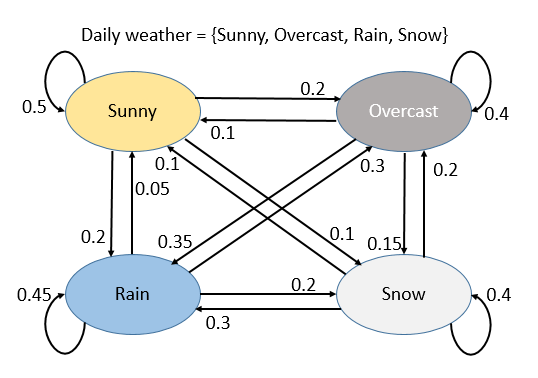
* + 1. *Markov Chains*

Markov Chains are used to describe types of random processes, giving information about the probabilities of sequences of random variables called states, which can take up values from a set (chain) for example the weather forecast for the day. Different states are then defined for the chain for example in case of weather the states could be {Sunny, Overcast, Rain, and Snow}. For a random process to be a Markov Chain it has to fulfil the property called the **Markov Assumption** which is defined as: the probability of being in a state j depends only on the previous state, and not on what happened before, or simply put only the present weighs in on predicting the future, while the past does not matter [4]. Mathematically this can be stated as:

Table xx below shows the essential components for a Markov chain:

|  |  |
| --- | --- |
| **Component** | **Explanation** |
| S={q1,q2… qn} | A set of ‘n’ states |
| P= | A transition probability matrix, containing probabilities of moving from state i to state j. |
| v=(v1,v2…vn) | Initial probability distribution vector. |

Fig. xx shows a state diagram with the transition probabilities, the states are shown as the nodes in the graph. The transition probabilities shown in the figure can also be shown in a matrix as:



In the stochastic probability matrix P each element (i,j) represents the transition probability. The deductions that can be made from this matrix are:

* Pk: each entry will represent the probability to arrive from i to j in k steps.
* If probability of being at a state at time t=0 are given by a vector ‘v’. The probability of being at state x at time t can be calculated as Pkv, the required result will be denoted by the xth entry in the matrix.
  + 1. *Hidden Markov Model*

As opposed to the simple Markov Model, in an HMM, Markov chains cannot be observed directly. In contrast the outputs generated from each stage are visible to the observer. Taking the example of a speech recognition system, the parts of the speech are non-observable events the words and their positions in a sentence are characterized as observable events and also can be the inputs of such a system. Table xx describes the components required to specify an HMM.

|  |  |
| --- | --- |
| **Component** | **Explanation** |
| S={q1,q2… qn} | A set of ‘n’ states |
| P= | A transition probability matrix, containing probabilities of moving from state i to state j. |
| V=(v1,v2…vn) | Initial probability distribution vector. |
| O=(o1,o2 … oT) | A sequence of T observations |
| B=bi(ot) | A sequence of observation likelihoods, they represent the probability of an observation ot being generated from a state i |

The Markov assumption as stated in the previous section also applies to HMM, but a 2nd assumption also comes into play which is termed as Output Independence and states that the probability of an output observation oi depends only on the state that produced the observation qi  and not on any other state or observation [5,6]. This assumption is mathematically described as:

* 1. Robot Operating System

ROS short for Robot Operating System is a collection of tools and libraries which is mainly designed to allow for writing robot software [30]. It provides the developers with low-level device control, drivers, package management and an infrastructure for message passing between processes. ROS systems are based on Publisher-Subscriber model. Active ROS processes are represented in a graph architecture and divided into different nodes. Each node can do its independent processing and can further communicate with other nodes. This communication is achieved by passing ROS messages. A node can be configured to be a publisher or/and a subscriber. In this scenario a publishing node does some processing and publishes some relevant information on a specific topic. This information could range from sensor data to images ( as in our case). Any other node which subscribes to this same topic name will have access to this data as soon as it is published by the first node.

**2.4.1 ROS-Master :**

A ROS-Master node must always run for the ROS based systems to work. This master node provides naming and registration services to all the other nodes in the system and manages publishers and subscribers to different topics.

**2.4.2 ROS Nodes:**

A node in a ROS system is any process that performs some computations.

A ROS eco system is made of multiple nodes. The advantage of using nodes is break down of code into simpler componenets. Added isolation so that if a fault occurs in one node, other nodes usually stay uneffected by it e.t.c. Node are combined together into a graph and communicate with each other by publishing and subscribing to topics or services.

**2.4.3 ROS Topics:**

Topics are named buses which are utilized by nodes to exchange information via ROS messages. Node in themselves are not aware of which other nodes they are commmunicating with instead they subscribe to the Topic containing the data that is relevant to them and publish the generated data to another relevant Topic if needed. Topics are generally used for unidirectional communication. ROS supports both TCP/IP based and UDP based message transport TCPROS is the default protocol used for communication.

For this project, the idea was to distribute the system between image processing tasks and machine learning (ML) tasks. The image capture and processing is done on the android device. Once the ROI has been extracted it should be transfered to a remote server running the ML module of the project which can then infer from the captured image-sequence the word or phrase that has been said.

For implementing this architecture several alternatives are available. For example the ML component (MLC) could be run on a cloud server (for example on Amazon Web Services) exposing the interface of the MLC via APIs. The android device can then transfer the images to the MLC via API calls and once the processing has been done on the cloud server the resulting string can be passed back to the android device via another API in json format.

Another option for implementation which does not require AWS, and in addition API development, was to use ROS. ROS provides the same architecture but on a simpler and lower scale. The application will again be divided into two parts Image-Processing Component which is to be run on the android device, and MLC to be run on a remote PC which must be on the same network as the android device running ROS-master node. For this architecture the Android application must also act as a ROS-node and once the ROI has been extracted it publishes the images to a ROS-Topic. The MLC running on the remote PC subscribes to this same topic. As soon as the images are available on the topic they are picked up by the MLC and the processing is performed. The results can either be shown on the PC or again published to another Topic to which the android device is subscribed to, so that the results can be shown on the device.

# Application Requirements

The requirement analysis for the application was done to come up with what needed to be done and to manage the development lifecyle of the application. The figure above shows a UML representation of the developed requirements. As can be seen from the requirements model, the developent has been divided into four phases to allow for easier development and clear goals. Each phase and the requirements contained in it are described in the following sections.

**3.1 Phase -1:**

*Analysis and comparison of facial and landmarks detection techniques:*

Phase-1 encompasses study of relevant image processing libraries and techniques which could be used for the application. A look into the timing and performance parameters of these techniques is to be performed so that the best approach could be decided for the development of the appliation in the later stages. The requirements corresponding to this stage are listed below:

*Req-1.1:* Determine best algorithms for face and facial landmarks detection, identifing parameter is timing performance.

*Req-1.1(a):* Application should run on android devices.

*Req-1.2:* Application should provide execution times for the relevant processes for analysis.

*Req-1.3:* Application should be able to detect faces in still images.

*Req-1.3(a):* User should be able to take an image using the device camera and use it for processing with in the application.

*Req-1.3(b):* User should be able to load an existing image from the gallery and use it in the application.

*Req-1.3(c):* OpenCV sdk for andoid should be used for face detection.

*Req-1.4:* Application should be able to detect facial landmarks in the given image.

*Req-1.4(a):* Utilize dLib for facial landmarks detection.

*Req-1.4(b):* Utilize Google’s Mobile Vision API for facial landmarks detection.

**3.2 Phase-2:**

*Real-Time Facial Landmarks Detection on Live Camera Stream:*

In phase-1 once the best approach has been identified, phase-2 is to extend the existing implementation from its application of still images to live feed from the on-board device camera. And to make the application as close to real time as possible given the hardware constraints of the target device.

*Req-2.1:* Andoroid application should be able to perform facial landmarks detection on live image stream from the device camera.

*Req-2.1(a):* App should utilize OpenCV and dLib in conjuction for facial landmarks detection

*Req-2.2:* Application should run close to or greated than 10FPS.

*Req-2.3:* App should be able to extract Lips ROI.

*Req-2.3(a):* Cartesian coordinates of the lips detection should be available for further processing in later stages.

**3.3 Phase-3:**

*Implementation of wireless data transfer to a remote computer:*

Phase-3 was to incorporate distributed processing capability in the application. Such that the image processing is to be performed on the android device but for further processing, the extracted images and information will be transfered to a remote computer running the Machine Learning Component of the application. This componenet will be responsible for processing the transfered images and deducing the said word or phrase from the given information.

*Req-3.1:* App should be able to transfer information over a wireless connection to a remote computer for further processing.

*Req-3.1(a):* Remote PC and the android device should be able to communicate over a TCP/IP link

*Req-3.1(b):* Use Robot Operating System for establishing data transfer capabilities.

*Req-3.2:* App should transfer the cartesian coordinates of the lips to the remote node.

*Req-3.3:* App should transfer the images of the Face/Lips ROI tot he remote computer.

**3.4 Phase-4:**

*Implementation of Machine Learning Component:*

Phase-4 is the final phase which brings together the work done in the previous parts and ties them together to produce the whole application. In this phase the Image processing results done on the device should be processed further on the computer. The chosen ML algorithm should then infer from the data the said word or phrase.

*Req-4.1:* App and PC should work as a system to analyse the video stream, extract facial landmarks and lips ROI and perform lip reading to infer the said word/phrase.

*Req-4.2:* A ROS node on PC should be able to fetch and save the incomming data from the mobile device.

*Req-4.3:* PC Node shall use TensorFlow to analyse the incomming images and perform lip reading.

*Req-4.3(a):* Training of TensorFlow model should be perfomed on a varied and a large dataset for better results.

*Req-4.4:* System shall be trained on a set of pre decided words and phrases.

# Architecture Design

As defined by the requirements the system is designed as a distributed system with two main parts.

* Android Application
* Machine Learning Module

The android application is to act as the interface to the user. It has been designed to utilize the onboard device camera of the android device to capture images/videos. No recording or processing is performed unless the user initiates the process from the Start/Stop button.

Once the process is started by the user indicated by the press of the Start button the algorithm starts to process the video feed from the camera frame by frame. The application is developed using OpenCV framework. *CameraBridgeViewBase* class of the OpenCV is used to control the camera operation. This class implements the behavior for interactions between android camera and the OpenCV library, including but not limited to the processing of the frames, enabling and disabling of the camera sensor and drawing of the image on the screen. Apart from this CascadeClassifier class from the OpenCV objdetect package is used for detecting faces in the frames. Although all the faces present in the frame are detected the further implementation on MLC is limited to processing a single face in the frame for now.

Dlib is used for detection facial landmarks. The dLib shared libraries created by Ttzutalin are imported in to the android application and the interface defined by jni functions are used to create the structure of our application.

Fig.XX shows the modules of the Android Application. Following sections give some details into the modules and their contents:

**4.0.1 AndroidManifest.xml**

AndroidManifest.xml file is a must have file for each android project and it contains information about the application’s packagename and components contained therein among other things. This information is shared with other tools and applications in the andoird eco system like the operating system, build-tools and Google play which is used to publish apps to end users. Some of the important information that is contained with in this file is the application’s package name as mentioned earlier which is used to infer a unique app identifier for the application, The names of all the componenets in the application for example activities, services e.t.c, device configurations and permissions that the app can have by default are also contained in this file like for our App, permission is granted to access the camera to capture video feed and to read/write to the storage by the following code snippet in Fig.XX

**4.0.2 Java Component**

The folder named java contains all the source code files associated with the project. The project in its current state has only two activities, and is divided further into four classes containing the code used for performing various functions, which are described in a latter section.

**4.0.3 JniLibs**

Android provides the capability to build or use pre-built native libraries in the android application. In order to allow Gradle to package prebuilt native libraries along with the APK the .so files are placed in the JniLibs folder corresponding to their respective hardware archtiecture.

**4.0.4 dlib**

This folder containes the prebuilt shared libraries and the Jni-Interface for the dlib library imported as a sub module in the project. The API of this library is used with the source code of our application to build the required logic.

**4.0.5 openCVLibrary343**

OpenCV sdk version 3.4.3 is imported into the project as a sub module as well and is later used for performing facedetection and handling camera related tasks

* 1. **Application Flow**

Fig.XX shows the flow of the application. Once the application starts MainActivity is started by the Android system which in turn starts the AvatarCameraActivity via Android Intent. The onCreat() method of this class initiates the required libraries as well as the device camera. In case the processImage flag is set to one (user input), the current frame is converted in to a bitmap and sent to the OpenCV face detector to extract the face ROI, if a face is not detected in the current frame the system moves on to the next frame and does the process again if a face is found in the frame the input frame is croped to the faceROI and sent to the dlib module for detecting landmarks the results are returned and shown on the device screen in a picture-in-picture mode. The results are also packaged into a ros message (details in a later section) and published to the defined ros topic.

* 1. **Component Diagram**

The application is divided into twelve components communicating with each other via different interfaces ranging from functions calls to ROS topics. The combination of all these componenets makes up the application work as a unit. The fig.XX below shows the information flow between the components as well as their associated interfaces.

* 1. **Class Diagram**

The developed system mainly consists of five classes, at the beginning an extra FaceDetection class was also implemented but in a later revision the code to extract face was moved from this class to AvatarCameraActivity class. As described earlier the application contains OpenCV android sdk version 3.4.3 and dlib prebuild shared libraries and the ROS package. The logic implemented by the code uses the API of these packages to perform its functions. There are further two dependencies on pre trained models for both OpenCV’s face detection algorithm and dlib’s landmark detection. In Fig.XX they are shown as artifacts:

* lbpcascade\_frontalface.xml
* shape\_predictor\_68\_face\_landmarks.dat

The classes and their functions are explained in the sections below:

**4.3.1 MainActivity**

The funcionality of the MainActivity in this instance is not much except for calling the AvatarCameraActivity via Android intent. MainActivity is also needed in the android architecture as this is the first activity that is started by the operating system when the application is started.

**4.3.2 AvatarCamerActivity**

This Activity is the backbone of the whole application. It handles the starting up and stoping of the camera, detection of faces and landmarks and also publishing of the landmarks data over the ROS network.

For handling the frames captured from the onboard camera the following OpenCV function is used :

public Mat onCameraFrame(CvCameraViewFrame inputFrame)

This is a pre built openCV method and is invoked the frames need to be processed and delivered on screen. The function returnes a modified frame in matrix form ready to be displayed on the screen. This function periodically calls the faceDetection method and then the landmarksDetection methods to extract the required information.

The landmarks data is then sent tot the Sender Class to be published over the ROS network to the set Topic.

**4.3.3 DetectLandmarks**

In the context of application’s architecture this class has been placed in the MvUtils package as this class is a helping class and provides and interface to the underlying dlib package and its java native interface. The class has two constructor methods, one provides the option to initilize the object with the application’s context object and a bitmap image. This constructor is provided if the implementation for landmarks detection on a single image is needed, and the constructor was developed in the previous version of the app. The second constructor only takes the context object to initilize the class’s object.

Context [31] in Android system provides an interface to global information related to the application’s environmnet e.g. app’s resources and classes and other operations like passing and receiving intents for launching activities.. The implementation of this class is provided by the Android system. The constructor checks if the shape\_model file for dlib is already present on the device storage or not, in case it is not present the model is downloaded from an external link and saved. For this operation permission to access the internet and read/write permission to the storage is needed. It has already been set in the AndroidManifest file but might have to be set in the device settings.

There are several different methods implemented in the class for detecting faces using OpenCV as well as dlib seperatly, the method being activley used in the current implementation is

public Bitmap detFacesFromBitmap(Bitmap b)

The method takes in a bitmap image and identifies the face in the image and then detects landmarks on the detected face. As per the dlib documentation landmark points labeled between 49 to 68 correspond to lips on the face. For out application these points are the ones in focus so the method only extracts these points and adds them to an array-list named landmarkPoints. This function only return the bitmap with an overlay of the detected landmark points. The interface to get landmarkPoints is provided by the method:

public ArrayList getLandmarkPoints()

An important part defined by this class is the location where the dlibe shape model should be in the device and is defined in the code by the following line:

new File(Environment.*getExternalStoragePublicDirectory* (Environment.*DIRECTORY\_DOWNLOADS*).getPath() + File.*separator* + *modelFileName*);

As can be seen in the definition the shape model is checked in the External storage in the directory defined by the DIRECTORY\_DOWNLOADS constant in the Environment class of android system and is defined as “Download”

**4.3.4 Sender**

The sender class is a ROS class for publishing data to topics on ROS network. The class has a simple constructor that does not take any external parameters and intilizes the names of the topics on which the data is to be published. The topic is set to “image\_transport”. The publisher object for compressed images is defined as:

Publisher<sensor\_msgs.CompressedImage> imagePublisher;

Sensor\_msgs/CompressedImage.msg [32] is a message type defined by the ROS system that deals with transferring different formats of compressed images, in a typical ros system this is used to handle data from on-board cameras on a robot. The message body is defined to contain a Header, fornat of the image in string format, currently supported formats are jpeg and png, a data buffer of type uint8[] to contain the image data.

**4.3.5 Reciever**

The reciever class is implemented to recieve a string message of ros message type std\_msgs.String. As per the current implementation the remote module does not publish any Strings back but as per the architecture the result infered by the ML component running on the remote computer will publish the results on a ROS topic and the Receiver module shall pick it up to be displayed on the device to the user.

* 1. **Sequence Diagram**

Fig.XX above shows the UML sequence diagram. This sequence diagram illustrates the flow of the process on the device side only. The application starts in the MainActivity from which the AvatarCameraActivity is initilized. The initilization process includes init of both the OpenCV and dLib libraries as well as the defined ROS nodes. During this initilization process, user is prompted to enter the ip address of the ROS master node as well.

Once all the initlizations have been run properly the on device camera is started by the OpenCVCameraView class and the frames are drawn on the screen. Further process from this point onwards only takes place upon the request of the user which is indicated by a button press on the screen. Once this processing starts it continues to run in a loop which can only be stoped by a user interrupt also indicated by the press of the “Stop” button.

Inside the processing loop, the frames are captured by the OpenCVCameraViewFrame and transferred to the AvatarCameraActivity, which is the main activity class for handling all the image processing tasks and routing the outputs. AvatarCameraActivity calls the faceDetection() function passing the inputFrame image matrix as the parameter, under the hood of this function is the OpenCV face detection algorithm. This method detects faces in the input frame and returns a cropped image of the face back to the AvatarCameraActivity.

The image containing the FaceROI is then passed to detectLandmarks function which passes the image to a n object of the DetectLandmarks class. An image with an overlay of 64 landmark points, their coordinates and the coordinates of the landmarks associated with the lips is returned from theis method call:

landmarkDetector.detFacesFromBitmap(croppedFaceImage);

Once the landmark points have been recieved the next step in the process is to perform the lip reading part using the ML componenet. To accomplish this the data is transferred over the ROS network to the topic named “image\_transport“

The ML component running on the remote PC takes over from there

***TODO: Sequence diagram for the ML componenet***

# Application Development and Challenges

**5.1 UI development**

The android system draws the user interface from xml files placed in the resources/layout directory. XML is the abbreviation for eXtensible Markup Language and is defined by www consortium’s XML specs and several others, this markup language is meant to be both human and machine readable.

For version1 of the project the goal was to compare several different image processing techniques and document their performance parameters to determine the best approach for implementing the application described in this document. Fig.XX shows the first design of the application UI

In the current version of the application there was no need for all of the UI elements so the UI was revised with the MainActivity only including a button to start the main process while the AvatarCamera activity now has the Layout as seen in fig.XX. The UI layout is quite simple with The Parent block being a FrameLayout containing three further child elements.

JavaCameraView fills the entire screen and the frames from the camera are drawn on this view by the OpenCV module.

An ImageView is embedded as a child element of this. By default this view is hidden. When the image processing starts the frames with the processed image with the landmarks drawn over the original face is displayed on this ImageView and it acts as the Picture-in-Picture.

A button is provided inside a LinearLayout and gives the user control over starting and stopping the detection process.

**5.2 Performance improvements**

With the normal implementation at the time of application’s development the average frame rate without any image processing was 23-25 FPS, and with the undelaying image processing algorithms running the frame rate dropped to 3-4 FPS. This frame rate is not enough for performing lip reading or speech processing. Improving the frame rate was the major challenge in the development process of the application. Following steps give a brief overview of the steps taken to improve the frame rate.

* Convert the Input frame from rgb to gray-scale.
* Face-detection skips some frames instead of running on every frame.
* Lower frame resolution (optimized for good results and performance).

RGB vs Gray-Scale image processing information here

Face detection is performed using OpenCV in order to crop the input frame to the FaceROI, removing the unnecessay portions of the image hence making the part of landmark detection faster. Assuming the usage of the application it is highly unlikly that the position of the face will displace too much during the usage. Keeping this assumption in mind the face detection process does not need to be executed on every frame. To achieve best results some setting were tried out to achieve a balance between performance and accuracy. Setting the rate of face detection to once per 10 frames brought the FPS while detection process up to 7, decreasing it to once per 20 frames brought it up to 9-11 FPS and the accuracy remained intact. Decreasing it further beyond this point had a negative affect on the accuracy as the position of face was too easily lost during the process. The setting to set the frame rate is done by the following line of code in AvatarCameraActivity:

private int frameSkipCount=20;

The associated public get and set functions for the private variable frameSkipCount are also provided with in the scope of the class.

The resolution by default for OpenCV’s camera handle is set to 960x720, and the resoltion for the testing device the Huawei Media Pad M5 Pro is 2k 2560x1600. These resolutions are very large and images of this size take a huge amout of processing power to go thorough image processing algorithms. To make the performance of the application better different resolutions were tried and 800x800 resolution produced the best result. Following code snippets set the frame resolution in the AvatarCameraActivity class:

private int MAX\_WIDTH = 800;

private int MAX\_HEIGHT = 800;

mOpenCvCameraView.setMaxFrameSize(MAX\_WIDTH,MAX\_HEIGHT);

The private constants define the resolution to be used by the OpenCV’s CameraBridgeViewBase and the method call to setMaxFrameSize sets the parameters.

# Results

Description of results here

# Conclusion

Conclusions here.

# Documentation

* 1. **Developer Documentation**

Developer’s documentation

* 1. **End-User Documentation**

End User Documentation

# Java Doc

# Appendix

# References