ARABIC SIGN LANGUAGE TRANSLATOR

HANDS SPEAK

Amel Bourdoucen, Majda Al Rashdi, Mouna Khriji, Wala Al Ramadhani

Final Year Project – Spring 2018

B. Sc. in Computer Science

Department of Computer Science

College of Science

Sultan Qaboos University

A report submitted in partial fulfillment of the requirements for the B. Sc. in Computer Science

Supervisor: Dr. Nasser Alzeidi

Examiner: Abdelhamid Abdesselam

DECLARATION

We hereby certify that this material, which we now submit for assessment of the report in partial fulfillment of the requirements for COMP5900, is entirely our own work and completed by members of our group except for information obtained in a legitimate way from literature, company or university sources. All information from these other sources has been cited and acknowledged within the text of our work.

Signed: ~~Amel~~ Date: 2/5/18

Signed: ~~Majda~~ Date: 2/5/18

Signed: ~~Mouna~~ Date: 2/5/18

Signed: ~~Wala~~ Date: 2/5/18

ABSTRACT

Arabic Sign Language uses manual communication to deliver a meaning just like other sign languages. The signs are classified into nouns/adjectives and verbs, similar to other spoken languages. This project aims to create a platform to ease the communication struggle of the deaf people when dealing with the other members of the society that do not understand the Arabic Sign Language (ArSL).

This is achieved by designing and implementing an application that performs four main steps necessary for the translation of sign languages. Namely, Training Supported Vector Machine (*SVM*), Hand Segmentation, Hand Feature Extraction and Gesture Matching. Training *SVM* main purpose is to train the *SVM classifier*, which is used later for gesture matching. Hand segmentation is the process of extracting the hand object from the frame while hand feature extraction focuses on extracting the main characteristics of the hand. The last step, which is Gesture Matching, concerns about comparing the extracted characteristics to the trained data present in the *SVM*. The result is translated Arabic text and speech in correspondence to the sequence of gestures captured by the application. The application is developed using *OpenCV* which is an open source computer vision software library that contains many optimized computer vision algorithms that are used within the application.

The application also provides a two-way communication by converting speech to text. In addition to that, it provides some other functionalities to allow storing and sharing the translated text with others. It is envisioned that the application will cater to the needs of the deaf people and will provide an efficient solution to reduce the difficulties in communicating with the other members of the society. This will therefore boost their confidence and contribution to the society and ease their day to day experiences.

ACKNOWLEDGMENT

We would like to thank Allah for giving us the strength and the determination to work on this progress report.

We would also like to thank our mentor and supervisor Dr. Nasser Alzeidi for his guidance and patience and Dr. Abdelhamid Abdesselam for polishing and reviewing our report. In addition to that, we would like to thank our dear friends and family for their constant support and understanding during the progress of this project. We thank the deaf community center in Muscat and Mr. Yahya Al Barashdi for his guidance and help in distributing the survey. Finally, we would like to thank the Computer Science Department at Sultan Qaboos University for providing the necessary resources to complete this project.

Table of Contents

LIST OF TABLES

LIST OF FIGURES

LIST OF EQUATIONS

Chapter 1: Introduction & Background

This section discusses the motivation and goals behind creating the Arabic Sign Language Translator application. The main purpose of the application is aimed at bringing people closer by eliminating the differences in communication with the help of a portable and efficient solution in the form of an application.

1.1 Problem Statement

The main issue that the deaf community faces is the lack of the ability to communicate with the rest of the society that does not know the Arabic Sign Language (ArSL). Our application aims to translate the detected hand motion into meaningful speech. There are other sign language translators available, but they are either expensive or not portable. Our aim is to provide a service to the users through a system accessible from their mobile phones, therefore providing portability and a cheaper option. Our approach aims to overcome the challenges of fast hand motion and blurriness to deliver to the Arabic deaf community an efficient ArSL Translator application with minimum requirements.

1.2 Goals

One of the major goals to achieve in this project is contributing to the change of the lifestyle of at least one person from the deaf community. Moreover, we plan to help deaf people connect with the rest of the community and not be limited in their everyday activities due to the fact that not everyone knows the ArSL. This can be done by developing a real time application that translates the ArSL into readable text and audible sound. The application also provides a two-way communication through translating Arabic spoken language to text. The ArSL translation process is done by detecting a sequence of gestures through a camera and translating them to be displayed and heard by the user.

1.3 Motivation

Communication is an essential part in our everyday life because it is our way of connecting with others as a person would feel isolated otherwise. However, standard methods of communication aren't available to the deaf and those who are unable to speak. Hence, communication can be an obstacle for the deaf since not everyone can understand the ArSL, and this can make them feel

neglected. Their struggle has motivated us to help them improve their communication lifestyle. As a result, we aim to translate the ArSL into speech.

1.4 Significance

The project is of great importance to those who are deaf and cannot speak. Since it would significantly change their lifestyle and would make their everyday activities that require communication easier. This is done by helping them communicate with the rest of the society through real time translation or through using additional features that ease sharing of the translation on different social media platforms.

1.5 Scope

The scope of the gestures that the application translates does not include 3D signs, signs in front of the face, signs which include movement, signs that can be translated from the context of the sentence, signs that change in meaning according to the changing expressions of the face. The application only analyzes the signs that have a distinct meaning, static signs and signs that are independent of the face.

1.6 Method

The tasks are focused on developing a real time application. The project is carried out through several phases as listed below:

1. Read about the Arabic Sign Language, the frameworks which can be used for gesture recognition, the cameras which are used to detect motion and gestures.
2. Explore research papers which involve hand recognition methodology.
3. Prepare a survey and hold a meeting with the head of the deaf society of Muscat to decide on the user requirements.
4. Design the logo and interface for the mobile application.
5. Study Swift and *OpenCV* framework and how to connect *OpenCV* with Swift.
6. Install all the required development tools.
7. Choose and plan a suitable algorithm to implement in the project.
8. Implement the project in four phases: Training *SVM*, Hand Segmentation, Hand Feature Extraction and Gesture Matching.
9. Test and validate the implementation.

1.7 Overview

The project is implemented using computer vision and image processing techniques. The main software utilities that are used in the project are *OpenCV*, *Swift* and *SQLite3*. "*OpenCV* is an open source computer vision software library which contains many optimized computer vision algorithms" [1]. These algorithms are used for tracking moving objects, face detection and removal. One of the special features provided by *OpenCV* is its ability to work with real time vision applications. Most of the code is written in *C++* and then connected with *Swift* using *Objective C* to develop the mobile application. The four main parts of the project are: Training *SVM*, Hand Segmentation, Hand Feature Extraction and Gesture Matching. The algorithms used for these are discussed in the coming chapters.

In addition to *OpenCV* and *Swift*, we are using *SQLite3* to store the gestures of the ArSL in the form of indices that represents the words that will eventually be used in gesture recognition to output a translated text/audio to the user.

1.8 Report Overview

This report consists of nine main chapters:

- Chapter one introduces the project and defines the goals, significance, motivation, method and problem statement of the project.
- Chapter two discusses the concept of the ArSL and how it is used. Also, a brief summary of what has been researched is discussed. In addition to that, the algorithms used are described. Research questions and some of the related works are listed.
- Chapter three is the project management part of the report which discusses the approach taken along with the initial and final project plan.
- Chapter four lists down the requirements of the project starting with the functional requirements and ending with the non-functional requirements. In addition to the requirements, it also contains the interview and survey questions which were conducted.
- Chapter five includes all of the design parts of the project and these consist of the user interface of the mobile application, the high-level database design and the sequence diagrams of *OpenCV* interaction with Swift and the mobile application.
- Chapter six describes the implementation of the project, it contains screenshots of some parts of the code along with the explanation of it.
- Chapter seven includes the testing part of the application and the discussion of the results obtained and their validity.
- Chapter eight includes the user manual of the application.

- Chapter nine lists down the limitations of the project along with the future works. It also summarizes the project.

Chapter 2: Literature Review

This section describes the concept of ArSL and the elements that are responsible for it such as: nouns and verbs and the way sentences are formed with the use of the hands. In addition to that, the section also describes the related research works for translating gestures, the algorithm used in this project, and the implemented related systems.

2.1 Arabic Sign Language

A sign language is a notation which uses manual communication to describe a meaning and is mainly used by the members of the deaf community [2]. There are many different types of sign language, one of them is the ArSL which is used by Arab societies. The units of the ArSL consist of three recognized phases: hand shape, position and movement. The signs are divided into two basic forms: nouns/adjectives and verbs. The words/signs lack prepositions, intensifiers etc. However, these are expressed in other ways; the intensifiers the signs are repeated.

2.1.1 Brief Introduction into ArSL

ArSL was only recently recognized as a standard language and documented. This was due to the reason that the Arab world consists of many countries, and every country has its own version of the ArSL (Figure 2.1). Huge efforts have been carried to be able to standardize and spread awareness of the importance of this step towards easing the conversations in the community of hearing impaired people.

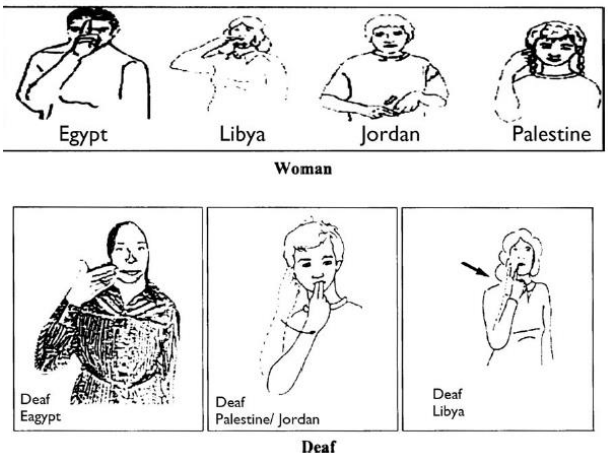


Figure 2.1 Similar word, different sign in different countries

2.1.2 Evolution

The history of Sign Languages goes back to Pedro Ponce de León (1520-1584) a Spanish man who was the first person to develop methods to teach the deaf children in 1555. Today the signs in ArSL are estimated to be more than 1350 signs [3]. ArSL letters are shown in Figure 2.2.

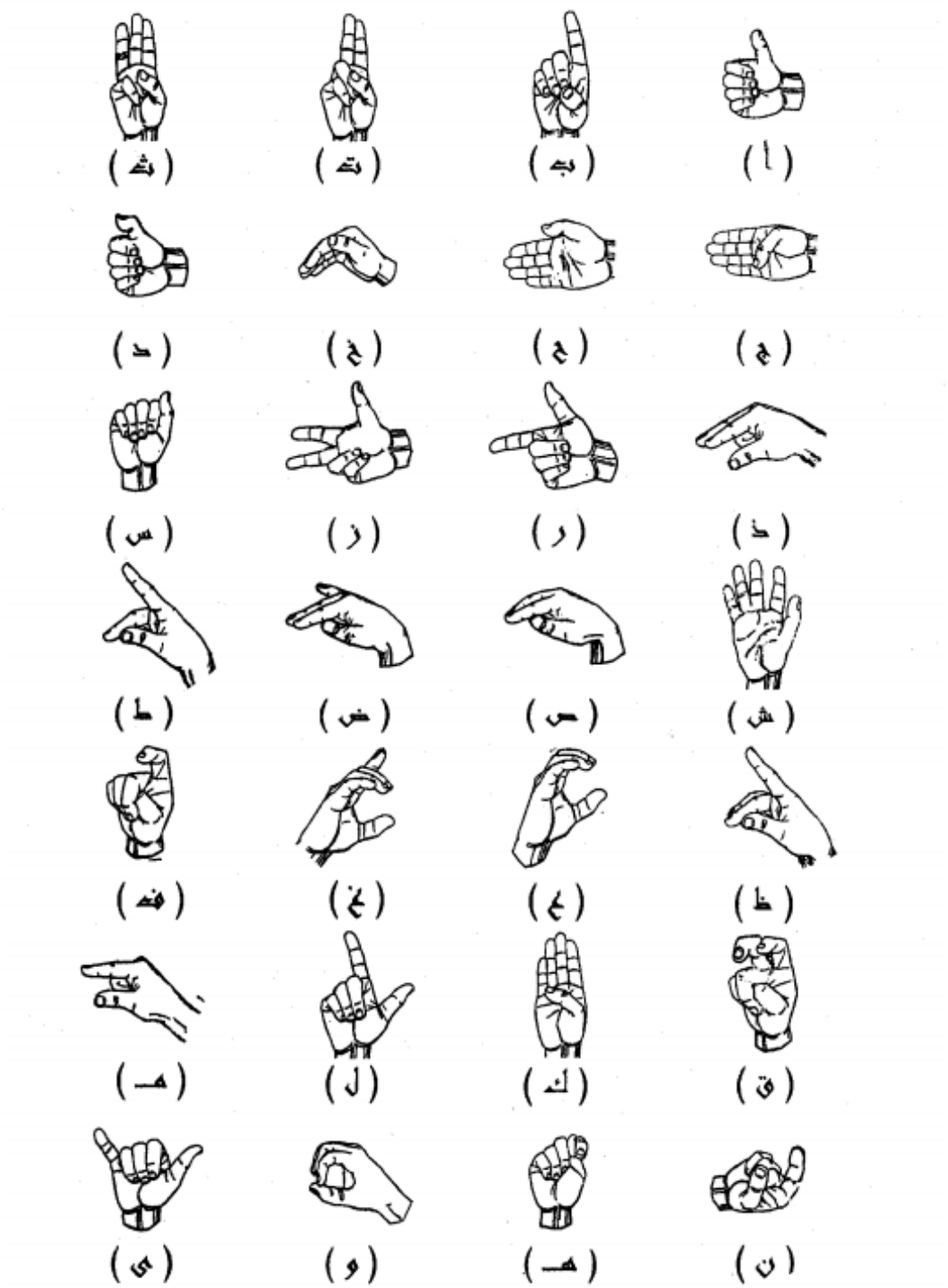


Figure 2.2 Arabic letters signs [1]

[1] Image taken from "قاموس لغة الإشارة" [3]

Sign languages have existed even with verbal languages; as people tend to use their hands to express themselves sometimes while speaking. ArSL has many similarities with other sign languages like the American Sign Language (ASL) or the British Sign Language (BSL). There are different resources that explain the origin of ArSL:

- Borrowing: From other sign languages such as American Sign Language (ASL).
- Creations: of new Signs as a result of gesture repertoire of spoken languages.
- Miming of things and actions.
- Expanding: for instance, compounding and blending as a result of merging two different words to form a third one with a new meaning.
- "Dumb" regional signs: inherited throughout the years from "mute" people.

Finger spelling is a new technique in ArSL. The purpose of it is to be able to spell nouns and verbs that have no sign correspondence.

2.1.3 Structure and Comparison

ArSL is composed of three main components: configuration of the hands (hand shape), placement and space (position of hand in correspondence to the body), and finally movements (interaction with space and direction). ArSLs also use their non-manual features like the eyebrows for example as shown in Figure 2.3.

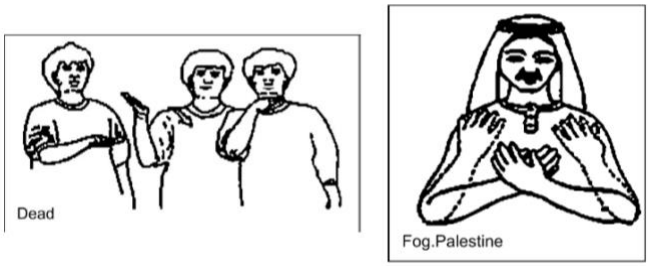


Figure 2.3 Using no manual features. (Dead: lips spread together with hand movement; fog: eyes slightly closed)

There are two classes of words in ArSL: nouns/adjectives and verbs. The language however lacks many other components of the sentence like adverbs and intensifiers. These can be delivered by other ways such as directions of signs in relation to each other, prepositions like Figure 2.4, repetition of certain signs to intensify the word.

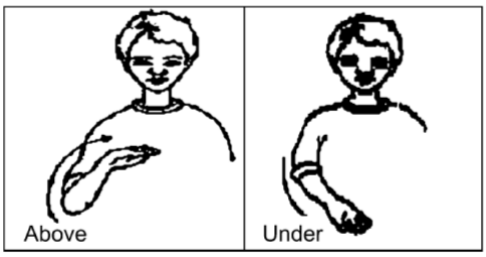


Figure 2.4 Prepositions. (Above and under: miming in)

The words whether nouns/adjectives or verbs can be represented in the following categories:

- Synosigns: which indicates two signs with the same meaning. This is not common in ArSL. This is shown in Figure 2.5.
- Antosigns: mostly complementary pairs, this is only different in movement.
- Homosigns: it is easy to understand these signs from the context.
- Compounds: two signs can mean something completely different when combined.

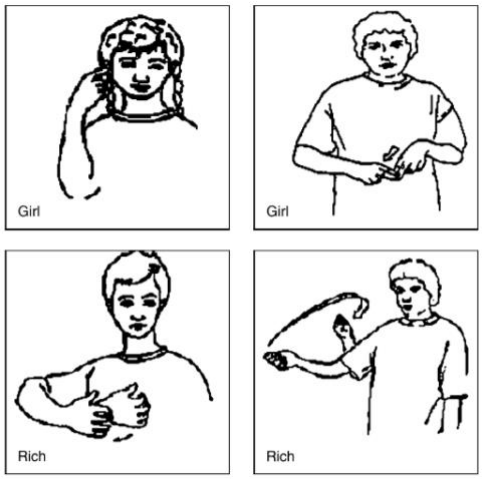


Figure 2.5 Synosigns. (Girl and Rich expressed in different signs)

ArSL in general does not follow the same order of their spoken or written counterparts. In fact, there is more focus on content signs such as nouns and verbs. For instance, a sentence is usually made up of a predicate and a subject like "he deaf" instead of "he is deaf". Tenses such as past and present tense are usually practically used in the sense that they are usually mentioned in at the beginning of the conversation and only shifted when they need to be changed to another tense. Emphasis is usually done by repetition, longer signing time or change of facial expressions. Word order can be expressed in various ways, such as (verb-subject-subject or subject-verb-object). This makes it very flexible and easy to comprehend [2].

2.2 Related Works

This section previews some of the research papers on gesture recognition, their achievements and limitations.

2.2.1 Gesture Recognition

There are three main procedures in hand gestures recognition that will be discussed: Hand Segmentation, Hand Feature Extraction and Gesture Matching.

2.2.1.1 Hand Segmentation

This phase takes as input the frame captured from a camera and segments the hand from the background using skin color. Therefore, the output of this phase consists of the hand area in white color and the background in black.

Some papers suggest converting the captured frame to *HSV* (Hue Saturation Value) color space [4], while others suggest converting it to *YCbCr* [5] [6] [7] "*YCbCr* is a color space where *Y* is defined as the luma/brightness of the color, *Cb* represents the blue component difference and *Cr* represents the red component difference" [8].

After frame conversion, skin color extraction is done by either using a predefined skin pixel values as proposed by the papers in [5] [6] [7]or by using a one-time learning square as suggested by [9] [4]. The learning square extracts skin color pixels by displaying a box and asking the user to place part of their hand within the boundaries of the box then, the maximum and minimum values of *RGB channels* are stored and these values are transformed to *HSV* color space [4]. However, the learning square has some limitations; one of them is its inability to recognize the hand in different lightings; if the user moves their hand, the lightning might change and since the learning square is done only once, the skin pixel values are unchanged, therefore, the system might not to recognize the hand. Also, the other limitation that might occur is the face being extracted if it is in the frame, thus the face is removed using *Haar-like feature* [5] [6] [7] . In addition to that, parts of the background which have similar color to the skin might also be extracted and considered as a hand, this limitation is called background color leakage. It is avoided by extracting the background and subtracting it from the original frame which contains the hand [7].

Morphological operation *opening* is applied to fill the holes and remove the noise produced after skin color extraction [5] [7]. It is a process that consists of *erosion* followed by *dilation*. However, after this process black holes within the hand are expected.

2.2.1.2 Hand Feature Extraction

As discussed in the previous section, the output is a binary image; white blobs with a black background. A contour is a list of points that represent a curve in an image [7]. The outline of the white blobs is considered as contour and it is extracted by setting a minimum threshold value as

shown in Equation 2.1. This process also helps get rid of the unwanted noise that is present in the white blobs. This solution is proposed by a research paper that was done for a gesture recognition system for human-computer extraction [7].

$$\mathbf{54} \leq \boldsymbol{Y} \leq \mathbf{163}\ \boldsymbol{and}\ \mathbf{131} \leq \boldsymbol{Cr} \leq \mathbf{157}\ \boldsymbol{and}\ \mathbf{110} \leq \boldsymbol{Cb} \leq \mathbf{135}$$

Equation 2.1 White Blobs Minimum Threshold Value

The palm area is found upon determining the *maximum inscribed radius*. Highlighting the palm area of the hand allows the calculation of the shortest distance of every point in the contour to the contour perimeter. The issue here is that the process is time consuming so the solution to this is instead of considering all the points in the contour, we only consider a specific number of points therefore limiting the *region of interest* (ROI) [7].

To eliminate the arm and other unwanted area, we further limit the ROI to the palm area only, this is done by using the radius of the *maximum inscribed circle.* After that contour extraction and *polygon approximation* is further performed on the smaller ROI. Using this new area, the *minimum enclosing radius* is found (The methods to calculate the *maximum inscribed circle* and *minimum enclosing radius* are not mentioned). *Out of roundness area* consists of everything outside the area of the maximum inscribed circle in Figure 2.6. [7]

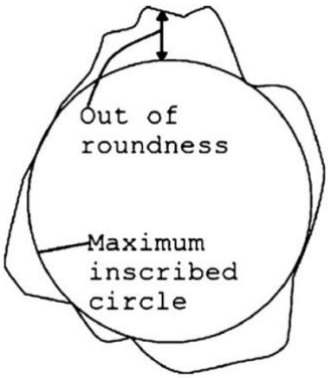


Figure 2.6 Maximum Inscribed Circle

In the hand itself there are still unwanted areas that lie between the polygon and the fingers. This issue is solved by marking a convex hull region and its convexity defects. The line around the hand is the convex hull marked from A-H, and the areas between the convex hull and the hand are the convexity defects as shown in Figure 2.7. These areas are used to differentiate between the gestures so the presence of all the areas is not necessary at all times [7]. The details of some of the calculations of the features are explained later in this report.

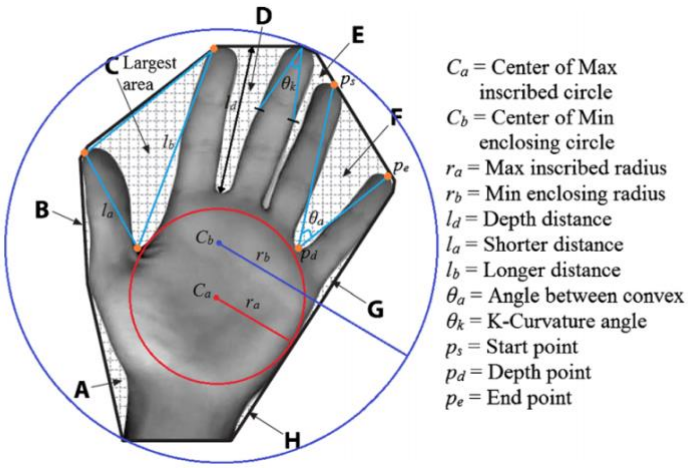


Figure 2.7 Convex Hull, Convexity Defects of Hand Shape

According to another research paper done on ArSL [5], after finding the convex hull and the convexity defects the *7 Hu moments* features (a set of features of an object) can be calculated to make the feature extraction process easier. The details of the 7 features is as follows:

- **Center of gravity:** Using the tracking movement of the hand in every frame, the velocity and angle calculations are performed to obtain the center of gravity (x_C, y_C). *M* is an instance of the moments function. *p(x, y, z)* is the density at the point *(x, y, z)* in the element of volume *dV* as shown in Equation 2.2 below:

$$M_{xy} = \int xp(x,y,z)dV$$

Equation 2.2 Moments of mass about a plane

M1.0, *M0.0* and *M0.1* are used to calculate the center of gravity (x_C, y_C) as shown in Equation 2.3:

$$x_c = \frac{M_{1.0}}{M_{0.0}}$$

$$y_c = \frac{M_{0.1}}{M_{0.0}}$$

Equation 2.3 Center of gravity

- **Velocity:** Due to the motion of the center of gravity as the hand moves, the velocity can be calculated assuming the center moves from point (X_1, Y_1) to (X_2, Y_2) in the duration from one frame to the next. As shown in Equation 2.4 below:

$$V_x = X_1 - X_2$$

$$V_y = Y_1 - Y_2$$

$$V = \sqrt{V_x^2 + V_y^2}$$

Equation 2.4 Velocity

- **Angle:** With the aid of the velocity, the angle of the hand motion can be calculated. As shown in Equation 2.5 below:

$$\theta = \tan^{-1}(\frac{Y_1 - Y_2}{X_1 - X_2})$$

Equation 2.5 Angle

Where θ is considered to be the angle and V_x and V_y are in correspondence to the velocity in *x* and *y* axis.

- **Area:** Here the area of a contour is defined as the number of pixels within the limits of a contour [5]. Which can be found using Equation 2.6:

$$\mathrm{A} = M_{0.0}$$

Equation 2.6 Area

- **Perimeter:** the perimeter is defined to be the length of the contour [5].

$$\mathrm{T} = \sum_{i=1}^{N-1} d_i = \sum_{i=1}^{N-1} |x_i - x_{i+1}|$$

Equation 2.7 Perimeter

As shown in Equation 2.7, the perimeter is the summation of the length of segments between subsequent points of a boundary.

- **Orientation:** One of the most important features to be extracted, this is calculated using Equation 2.8:

$$\theta = \frac{1}{2}\tan^{-1}(\frac{2M_{1.1}}{M_{2.0} - M_{0.2}})$$

Equation 2.8 Orientation

In another research paper, a different approach towards specifying the features to be extracted is specified. Contour features are analyzed to determine if the shape contains fingertips, which also indicates that it's indeed a hand. These contour characteristics will help determine the state of the hand/fingers if it's open or closed, hand/fingers count and hand/fingers location [7].

Two issues might occur in the stage of hand detection: false positives and motion blur. This is due to the fingers not being able to be detected at the same time so the solution to this is to add a short delay time in order to withstand the quick changes in the hand state [7].

The *Kalman filter* [10] [11] is a computational algorithm which implements a predictor-corrector type estimator to reach an optimum estimation of past, present, and future state of a linear system to minimize the estimated error covariance. *Kalman filter* is necessary to get rid of the jumpy and unstable hand locations that occur during the change in the hand state [7].

In other research papers instead of manual calculation of features, *SURF* (Speeded-up Robust Features) is used [12]. *SURF* is a local feature extraction method that computes *keypoints* and their *descriptors*, *SURF* is invariant to changes in illumination, rotation and scale. It is considered a fast feature detector which is mainly used in real-time processing. In *SURF* the *keypoints* are detected using *Hessian Matrix* [13] and the *descriptors* are computed by taking into consideration the surrounding region of each *keypoint*. This region is computed using *Haar wavelet*.

2.2.1.3 Gesture Matching

The papers that used local *descriptors* of *SURF*, used a type of classifier to recognize the gestures [14] [15]. Followed a *SVM* classifier, this is a supervised learning approach that requires a training dataset to learn and be able to recognize new data. A bag-of-visual word (*BoW*) model is an approach used to quantize *descriptors* into 'visual words', decreasing their sum and making them more robust to changes. The reason of using *BoW* is that the number of *descriptors* extracted from all images is enormous and of different sizes. In addition, searching for the nearest neighbor descriptor of all images consumes a long time, the *BoW* methods solves these problems. The *descriptors* formed by *BoW* are the training dataset of *SVM* classifier. In order to form *BoW*, three stages are required:

1. **Feature descriptors**

   Descriptors of the database images are computed using *SURF* feature extraction.

2. **Building vocabulary**

   The *descriptors* extracted are clustered using *K-means* algorithms, where *K* is the number of bags being chosen to use. The clustering results in a vocabulary of *K* clusters.

3. **Construct bag of words**

   For each local descriptor of images, the nearest visual word from the vocabulary to the descriptor is found, using a *Flann Based matcher*. Then compute the *BoW descriptors* of images, which is a normalized histogram of vocabulary frequencies of images.

The *BoW descriptors* obtained from the three stages are then used to train the *SVM* classifier as the *descriptors* labels are provided too.

The following defines some of the terms used:

1. **Bag of words / features:**

   A model that is commonly used in computer vision, where it stores the occurrences of each word (feature) of all images for training the classifier.

2. **Vocabulary building:**

   A list of all unique features is obtained, then are clustered using *K-means* algorithm.

3. **Flann matcher (Fast Library for Approximate Nearest Neighbors):**

   Contains a collection of optimized algorithms for nearest neighbor matching. This method is very fast and suits real-time processing and large dataset matching.

2.2.2 Accuracy and Constraints

Table 2.1 Summary of accuracies and constraints of research papers

Application	Word Recognition Accuracy Percentage	Sample Space	Constraints
Dynamic Hand Recognition of Arabic Sign Language using Hand Motion Trajectory Features [5]	85.67%	1260 feature vectors so 63 feature gestures for each gesture using 8 signers.	High similarity between gestures, usage of two hands and interference with the face.
A Mobile Application of American Sign Language Translation via Image Processing Algorithm [9]	97.13%	A set of 16 Arabic Sign Language gestures.	Limited dataset.
Hand Tracking and Gesture Recognition for Human-Computer Interaction [4]	98%	Each user performed a set of 40 gestures. In total 1600 gestures.	Similar skin background limitations and several hands intersecting in same time and space.
Hand Detection and Feature Extraction for Static Thai Sign Language Recognition [6]	77.78%	72 input frames with a total of 42 gestures.	Takes as input only static images.
Hand Tracking and Gesture Recognition System for human-computer interaction using low-cost hardware [7]	86.66%	10 users with each user performing a set of 30 hand gestures	Data is not provided.

2.3 Algorithms and Procedures

From a collection of many algorithms and procedures proposed by many research papers that are explained in the section above, we have chosen several of them with regards to the level of efficiency and effectivity. The algorithms adapted by our project are explained in this section in the form of four phases: Training *SVM,* Hand Segmentation, Hand Feature Extraction and Gesture Matching. Each phase is divided into sub phases.

2.3.1 Training SVM

This phase is performed only once, and its purpose is to train the *SVM classifier*, which is used later for gesture matching. The phase of constructing the training data consists of three main phases: *SURF* Feature Extraction, Vocabulary Building, bag-of-words construction. This data will then be the used to train the *SVM*.

2.3.1.1 SURF Feature Extraction

This phase takes as input a set of binary images of different gestures, each image has a fixed size of 300 by 300 pixels. Then the *keypoints* of each image are extracted using *SURF*. These *keypoints* are then used to extract the *descriptors* which are then stored unorderly in a Bag of Words (*BoW*).

2.3.1.2 Vocabulary Building

The *descriptors* in the *BoW* created in the previous step are clustered using *Kmeans* clustering technique [16]. This step is done to obtain the unique features of all images which is called vocabulary building, *BoW* vocabulary is then set.

2.3.1.3 Bag-of-Words creation

BoW which is a normalized histogram of vocabulary frequencies, is constructed from *SURF keypoints* extracted from the dataset of images. The *BoW* extractor is used to extract the *keypoints' descriptors* and match them to the constructed vocabulary using *FlannBased* matcher. This is done to obtain the occurrence of each vocabulary. The resulted *BoW* is the trained dataset. At the same time, the labels of each descriptor obtained from the image are stored, this will ensure mapping the vocabulary to a label (gesture).

The data and their labels are then used to train the *SVM* classifier (*OpenCV's train_auto* function is used to select optimal parameters).

2.3.2 Hand Segmentation

The main goal of this phase is to get a binary image with the hand region as white and the rest as black. The input is an image frame which is retrieved from the system's camera and then the result of this phase is passed to the next phase for hand feature extraction. Listed below is the detailed descriptions of these steps.

2.3.2.1 Frame retrieval

The image frames are retrieved from the device's camera at default value of 30 frames per second [17].

2.3.2.2 Convert frame to YCbCr

Some of the previous works on skin color extraction have converted the frame to *HSV* while others converted it to *YCbCr*. Here, the frame is converted to *YCbCr* digital video color space because the transformation and the separation of the color is efficient in comparison with *HSV* color space. In addition, the process of converting an image to *HSV* is time consuming. Moreover, it was found that *YCbCr* has a better accuracy than *HSV* for representing the skin color region.

2.3.2.3 Background Subtraction

One of the limitations in previous papers is background color leakage. This is avoided by extracting the background and subtracting it from the original frame which contains the hand. The absolute difference in *YCbCr* color space of the extracted background and the frame with the hand is calculated and then split into 3 channels (*Y*, *Cb*, *Cr*). Each channel is processed differently by thresholding each channel with its own minimum and maximum threshold and then the noise is removed using morphological operation *opening*. After that, the three channels are combined together with a simple addition operation. After the trial of different morphological operations, applying morphological operation *closing* twice is found most suitable to remove the remaining noise. Finally, we mask it with the original frame and the output would be a *YCbCr* image which contains only the hand.

2.3.2.4 Face Removal

The other limitation of the papers discussed before is the extraction of the face in skin color extraction phase. Since the face is of the same skin tone as the hand. We must first detect the face using *Haar Feature-based Cascade Classifiers* [18]. After detecting the face, the face region is filled with a black ellipse, to prevent it from being extracted in the skin color extraction phase.

2.3.2.5 Skin Color Extraction

The threshold values for skin color extraction in [7] is based on the values suggested by D. Chai ($\mathbf{54} \leq \boldsymbol{Y} \leq \mathbf{163}\ \boldsymbol{and}\ \mathbf{131} \leq \boldsymbol{Cr} \leq \mathbf{157}\ \boldsymbol{and}\ \mathbf{110} \leq \boldsymbol{Cb} \leq \mathbf{135}$) [19]. *OpenCV* library provides a function which performs thresholding operations to detect objects based on threshold values.

2.3.2.6 Morphology and Gaussian Filtering

Open morphological operation is used to efficiently remove the noise. It consists of *erosion* followed by *dilation*. Later, *Gaussian filtering* is applied to smooth the noise around the edges of the hand.

2.3.3 Hand Feature Extraction and Gesture Matching

In hand feature extraction we have decided to use *SURF* instead of *hu moments* due to the reason that *SURF* is a more accurate method for feature extraction and it's faster in terms of performance. In this stage the obtained binary image from the previous step is used to extract *SURF keypoints*. Then, the *descriptors* are computed using *BoW* extractor which in turn uses *FlannBased matcher* to estimate the nearest neighbor vocabulary.

The extracted *descriptor* of type *BoW* is then fed to the *SVM* predict function to classify it and return the corresponding label. This label is the index used to search the database for the text translation of the gesture.

2.4 Research Questions

During the course of this project, we need to address the following questions:

1. Which framework is best suited for capturing and detecting the hand movement?
2. What are the different algorithms used for hand detection and tracking?
3. What is the most suitable color space to use in the hand detection phase?
4. How to extract the skin tone color? And which method is the most suitable?
5. What is the most accurate threshold to remove the contour with the unwanted noise?
6. What are the characteristics needed to distinguish the detected hand and the trained data set of Arabic signs?

2.5 Similar Systems

There are very few systems which translate the sign language. However, none are used in mobile applications, these systems include:

1**. KinTrans** [20]:

It is a software which translates in real time the sign language into voice and voice into text. It is built mostly for business use rather than personal use since it is not portable and requires special hardware equipment such as a Kinect sensor which is used to capture the gestures. The Kinect sensor needs to be connected to a monitor. Figure 2.8 below shows how the Kinect camera works. This system is different from what we are developing since our system can be used in any portable device or even a desktop with a normal webcam and so our project does not require any special sensor or equipment.

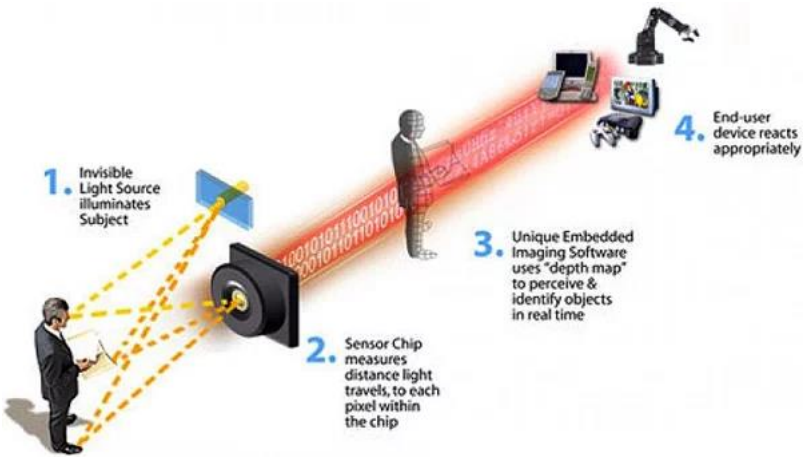


Figure 2.8 How Kinect camera works [2]

2. **MotionSavvy UNI** [21]:

"It is a real time translation technology that converts sign language to grammatically correct spoken language". It uses a special tablet called UNI which is built solely for the purpose of translating the sign language to audible voice and vice versa. Figure 2.9 below shows how UNI works However, so far, they are only translating the American Sign Language unlike our application, which would be translating the Arabic Sign Language without the usage of any sort of special tablet since our system can work on multiple platforms including mobiles, tablets and desktops.

[2] Image taken from https://www.jameco.com/jameco/workshop/howitworks/xboxkinect.html

Figure 2.9 How UNI works[3]

2.6 Conclusion

In this chapter, we have researched the ArSL to understand the syntax and dictionary of the language and to get an overview of it. The next step was to study the Gesture Recognition process through implementations of different systems to learn the different algorithms used for this. Research questions were addressed to be answered during the project phases. Lastly, the related works were discussed to view the significant differences in systems.

[3] Image taken from https://www.fastcompany.com/3053285/this-gadget-reads-sign-language-and-then-translates-it-into-speech-to-give-the-deaf-a-voice

Chapter 3: Project Management

In this chapter the project management approach will be discussed. The ArSL Translator application was planned by dividing the tasks between the team members and setting a specific time interval for each task to be finished.

3.1 Approach

The approach taken to manage the ArSL Translator is the waterfall approach which is based on a linear sequential series of tasks and distinct goals to be achieved at each level.

3.2 Initial Project Plan

Figure 3.1 shows the division of tasks and the duration for each task using the Gantt chart. Table 3.1 shows the tasks and their corresponding dates.

Table 3.1 Project Plan

VIEW 1

	Task Name	Duration	Start	Finish	Predecessor	Resource Names
1	**Background**	**27 days**	**Mon 1/8/18**	**Wed 2/21/18**		**Majda,Amel,Mouna,Wala**
2	Read about sign language	4 days	Thu 2/22/18	Wed 2/28/18		Majda,Mouna
3	Research about motion cameras	9 days	Sun 1/14/18	Thu 1/25/18		Amel,Wala
4	Read about framework used for gesture recognition	8 days	Mon 1/22/18	Thu 2/1/18		Amel,Wala
5	Read research papers about hand recogntion methodology	10 days	Mon 2/5/18	Tue 2/20/18		Amel,Majda,Mouna,Wala
6	Meeting with the head of the data society in musct	1 day	Thu 2/22/18	Thu 2/22/18	1	Majda,Mouna
7	Prepare a survey	1 day	Wed 2/21/18	Wed 2/21/18		Mouna
8	**Requirement and Design**	**4 days**	**Fri 2/23/18**	**Wed 2/28/18**		
9	User Requirements	2 days	Fri 2/23/18	Sun 2/25/18		Amel,Mouna
10	System Requirements	2 days	Sat 2/24/18	Mon 2/26/18		Majda,Wala
11	Interface Design	3 days	Sun 2/25/18	Tue 2/27/18		Amel,Majda
12	Requirements are well specified	0 days	Wed 2/28/18	Wed 2/28/18		
13	**Development**	**35 days**	**Tue 2/20/18**	**Thu 4/19/18**		
14	Background about swift and connecting it to opencv	3 days	Thu 3/1/18	Tue 3/6/18	2,3,4,5	Majda,Wala
15	Install and Register the required development tools.	3 days	Thu 3/1/18	Tue 3/6/18	2,3,4,5	Amel,Majda,Mouna,Wala
16	Implementation	30 days	Wed 3/7/18	Thu 4/26/18	14,15	Amel,Majda,Mouna,Wala
17	Progress Report Completed	0 days	Tue 3/13/18	Tue 3/13/18		
18	Verification	2 days	Mon 4/30/18	Tue 5/1/18	16	Amel,Mouna
19	Validation	2 days	Mon 4/30/18	Tue 5/1/18	16	Majda,Wala
20	Implementation is completed	0 days	Thu 4/19/18	Thu 4/19/18		
21	**Testing**	**13 days**	**Sun 4/22/18**	**Thu 5/10/18**		
22	Unit Testing	2 days	Thu 4/19/18	Mon 4/23/18	20	Amel
23	System Testing	2 days	Thu 4/19/18	Mon 4/23/18	20	Mouna,Wala
24	Acceptance Testing	2 days	Thu 4/19/18	Mon 4/23/18	20	Majda
25	Poster Submission	0 days	Sun 4/29/18	Sun 4/29/18		
26	Project Testing Done	0 days	Mon 4/23/18	Mon 4/23/18		
27	Final Report Submission	0 days	Thu 5/3/18	Thu 5/3/18		
28	Presentation	0 days	Thu 5/10/18	Thu 5/10/18		Amel,Majda,Mouna,Wala
29	Project Product Submission	0 days	Thu 5/10/18	Thu 5/10/18		

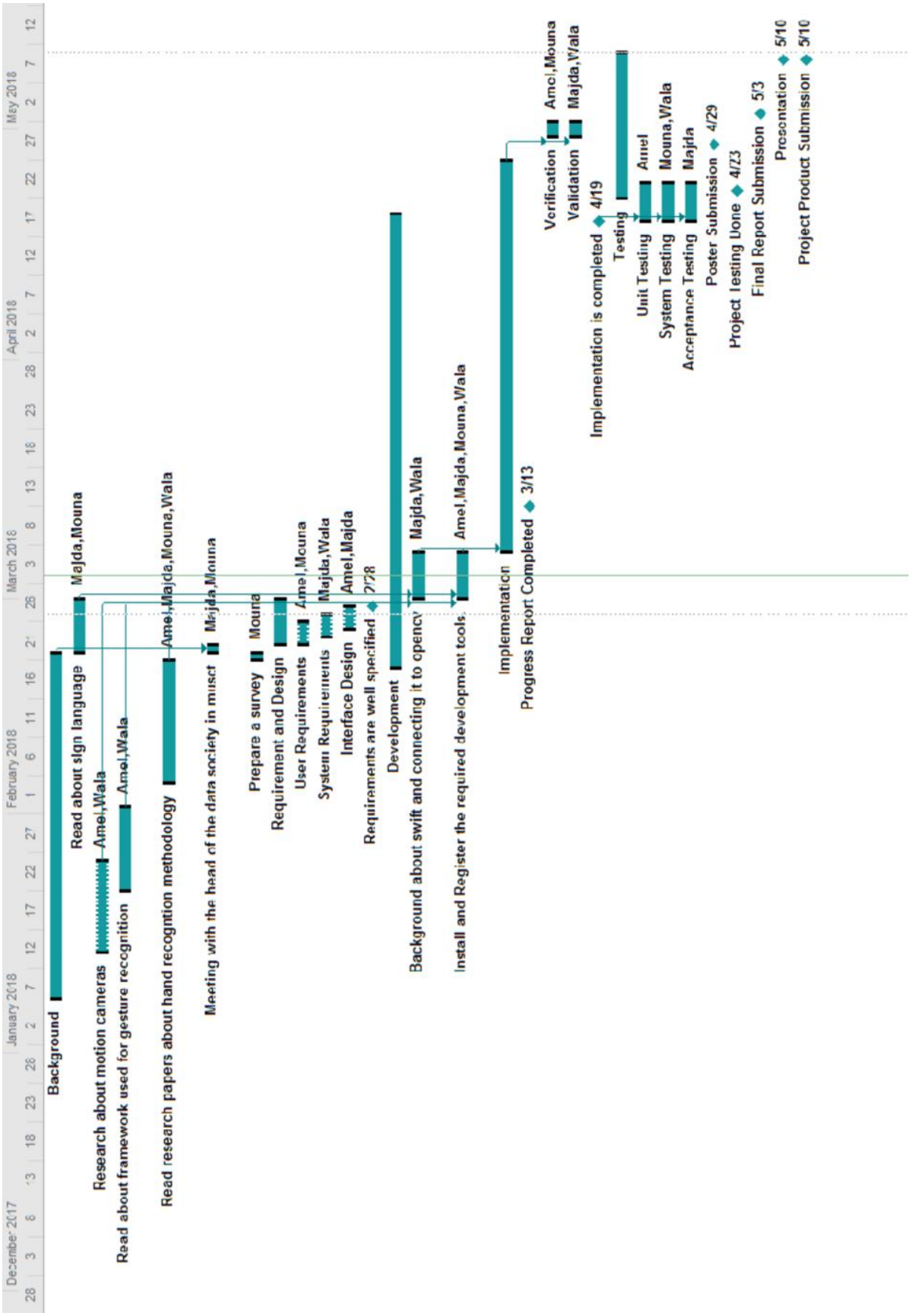


Figure 3.1 Gantt Chart

3.3 Problems and Changes to the plan

During the course of the project, there were slight modifications in the progress of the implementation. Initially, the algorithm used for feature extraction was using *hu moments* and *SURF*. Eventually, this was changed to training *SVM* to improve the accuracy of gesture recognition. As a result of this step, that was both risky and beneficial to the project, the testing part of the project was delayed.

3.4 Final Project Plan

The previous project plan was modified for the dates of the implementation and testing phases and the new project plan is shown in Table 3.2 and Figure 3.2.

Table 3.2 Final Project Plan

	Task Name	Duration	Start	Finish	Predecessors	Resource Names
1	**Background**	**27 days**	**Mon 1/8/18**	**Wed 2/21/18**		**Majda,Amel,Mouna,Wala**
2	Read about sign language	4 days	Thu 2/22/18	Wed 2/28/18		Majda,Mouna
3	Researched about motion cameras	9 days	Sun 1/14/18	Thu 1/25/18		Amel,Wala
4	Read about framework used for gesture recognition	8 days	Mon 1/22/18	Thu 2/1/18		Amel,Wala
5	Read research papers about hand recogntion methodology.	10 days	Mon 2/5/18	Tue 2/20/18		Amel,Majda,Mouna,Wala
6	Meeting with the head of the data society in musct	1 day	Thu 2/22/18	Thu 2/22/18	1	Majda,Mouna
7	Prepared a survey	1 day	Wed 2/21/18	Wed 2/21/18		Mouna
8	**Requirement and Design**	**4 days**	**Fri 2/23/18**	**Wed 2/28/18**		
9	User Requirements	2 days	Fri 2/23/18	Sun 2/25/18		Amel,Mouna
10	System Requirement	2 days	Sat 2/24/18	Mon 2/26/18		Majda,Wala
11	Interface Design	3 days	Sun 2/25/18	Tue 2/27/18		Amel,Majda
12	Requirements are well specified	1 day	Wed 2/28/18	Wed 2/28/18		
13	**Development**	**35 days**	**Tue 2/20/18**	**Thu 4/19/18**		
14	Background about swift and connecting it to opencv	3 days	Thu 3/1/18	Tue 3/6/18	2,3,4,5	Majda,Wala
15	Install and Register the required development tools.	3 days	Thu 3/1/18	Tue 3/6/18	2,3,4,5	Amel,Majda,Mouna,Wala
16	Implementation	29 days	Wed 3/7/18	Wed 4/25/18	14,15	Amel,Majda,Mouna,Wala
17	Progress Report Completed	1 day	Tue 3/13/18	Tue 3/13/18		
18	Verification	3 days	Tue 4/24/18	Thu 4/26/18		Amel,Mouna
19	Validation	3 days	Tue 4/24/18	Thu 4/26/18		Majda,Wala
20	Implementation is completed	0 days	Thu 4/26/18	Thu 4/26/18		
21	**Testing**	**13 days**	**Sun 4/22/18**	**Thu 5/10/18**		
22	Unit Testing	2 days	Thu 4/19/18	Mon 4/23/18		Amel
23	System Testing	2 days	Thu 4/19/18	Mon 4/23/18		Mouna,Wala
24	Acceptance Testing	18 days	Thu 4/26/18	Sat 5/26/18	20	Majda
25	Poster Submission	1 day	Sun 4/29/18	Sun 4/29/18		
26	Project Testing Done	0 days				
27	Final Report Submission	1 day	Thu 5/3/18	Thu 5/3/18		
28	Presentation	1 day	Thu 5/10/18	Thu 5/10/18		Amel,Majda,Mouna,Wala
29	Project Product Submission	1 day	Thu 5/10/18	Thu 5/10/18		

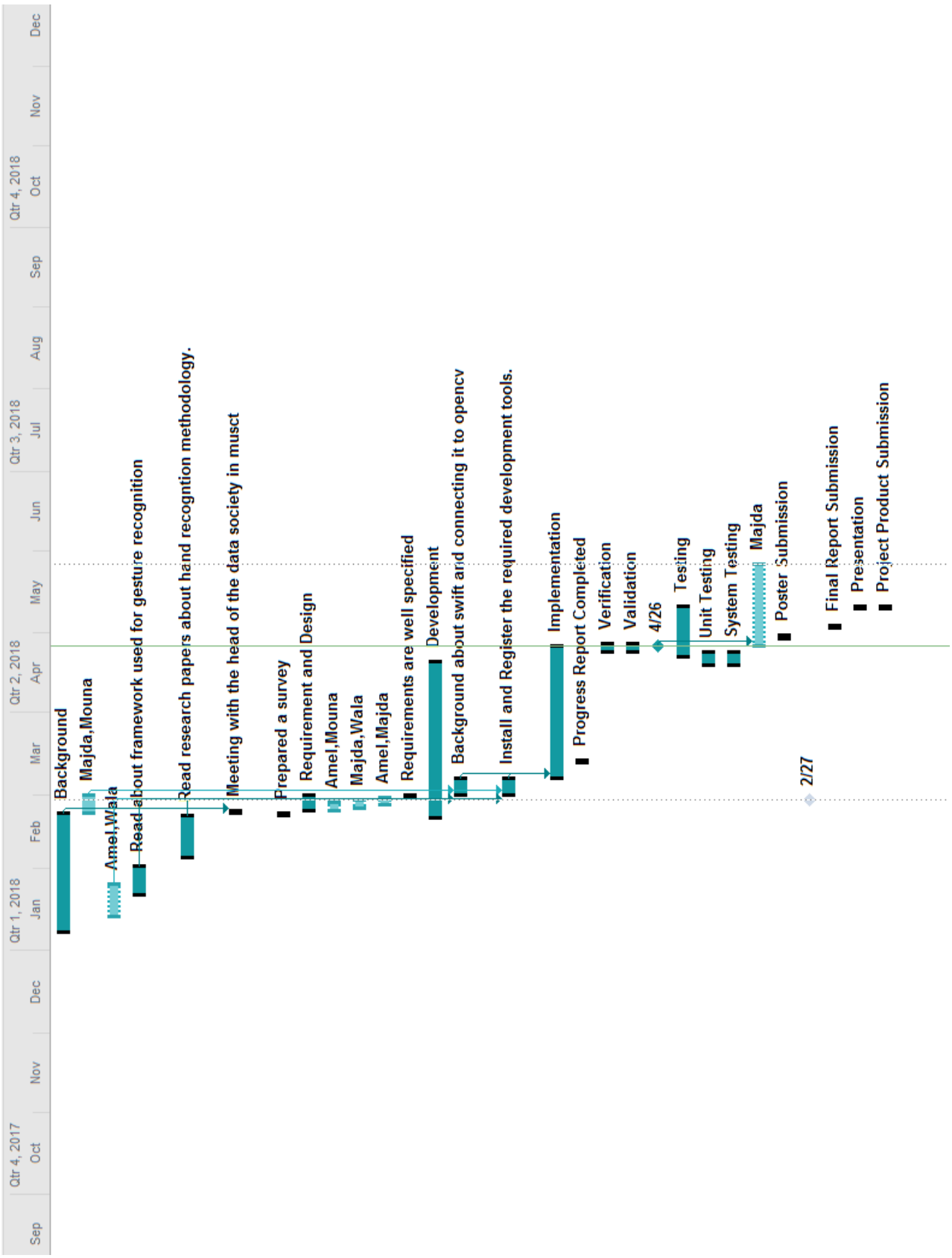


Figure 3.2 Final Gantt Chart

3.5 Conclusion

Tasks were divided equally between members and specific time spans were given for each task accordingly. The method used to manage the project was the waterfall method, this has helped maintain a sequential order while attempting the tasks.

Chapter 4: Requirements and Analysis

In this chapter the requirements are discussed in much more detail listing down the functional and non-functional requirements of our project. In order to achieve this, we have reached the deaf community to correctly identify their requirements. An interview has been conducted with a translator and below are some of questions asked.

Interview questions:

1. What are the challenges deaf people face in real life communication?
2. Does the sign language use 2D or 3D motion?
3. How fast do deaf people signal gestures when communicating with each other?
4. Are there special words that will differ in meaning with or without facial expression?

We have also designed and conducted an online survey targeted to the deaf people. Below are some sample questions from the survey and Appendix I shows the entire the survey.

Survey questions:

1. Specify the difficulty level of communicating with people who don't understand Arabic sign language.
2. What sign language do you use?
3. Do you have a personal translator?
4. Do you use an application to help you communicate with people who don't understand ArSL?
5. If there was an app to ease your communication would you use it?
6. What features would you like the app to have (e.g. save translated text/speech, record a video etc.)?

4.1 Functional Requirements

The survey was directed at 100 people in the deaf center in Muscat and the results of the survey that is included in Appendix II shows the following:

- The majority used the standard ArSL for communication. As a result, the dataset specifically contained the gestures of the standard ArSL.
- Most of the answers related to the question of "what features would you like the app to have?" required having an option to record the gesture along with the translated audio and text. In addition to this, saving the translated text and sharing it on social media. Both of these features were integrated into the application.

In this section, the requirements of the ArSL Translator application are explained with the aid of appropriate diagrams. The functional requirements are divided into two aspects: *OpenCV* interaction with *Swift* and *Swift* interaction with the mobile application.

4.1.1 OpenCV interaction with Swift Functional Requirements

Figure 4.1 shows the use-case diagram for the OpenCV interaction with Swift while
Table 4.1 shows the detailed description for the use cases. Listed below are the functional requirements:

- **StartCamera:** Starts the camera.
- **SwitchCamera:** Switches between front or back camera as suitable (The front camera is used when the person using the phone is translating his/her gestures to the other person who does not know ArSL. Whilst the back camera is used when the person not knowing the ArSL is trying to understand the gestures of the other person).
- **StopCamera:** Ends the process of video capturing.
- **initWithController:** Integrates the device's camera with the application's interface.
- **ProcessImage:** Includes two steps of manipulating the video stream:
    1. *HandSegmentation:* Identifies the hand area excluding all other details throughout the video frames and outputting a binary image of the hand only.
    2. *FeatureExtractionAndGestureMatching:* Processes the binary image by extracting its features and then matching those features with the training data and outputting the index of the word if the matching is successful.
- **getString:** Matches the index with the word and outputs the word text.

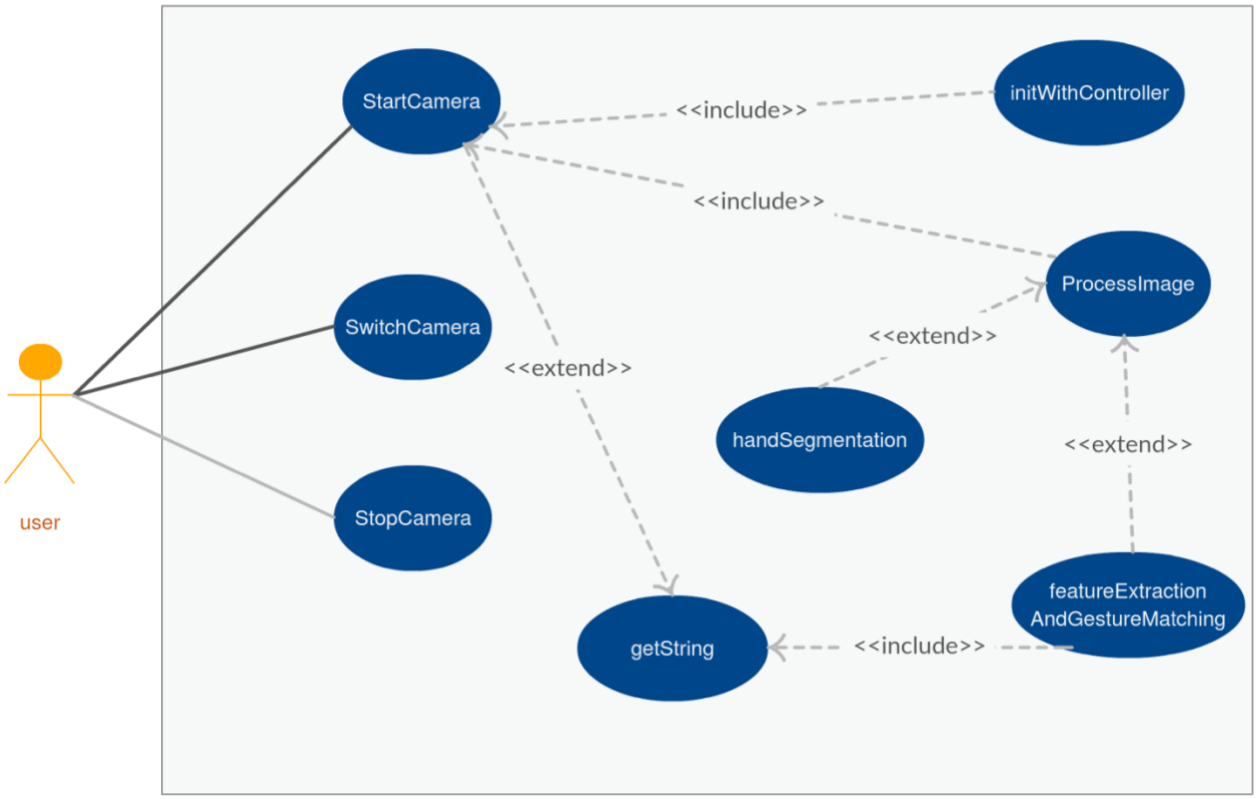


Figure 4.1 OpenCV interaction with Swift Use Case Diagram

Table 4.1 OpenCV interaction with Swift Use Case Description

Use case: StartCamera
Description: Starts the device camera. **Actor:** Application. **Precondition:** Permission to access the camera is granted by the device.
Main scenario: • The application calls StartCamera method. • It is then passed to InitWithController.
Alternatives: -

Use case: SwitchCamera
Description: Switches between back and front camera. **Actor:** Application. **Precondition:** The camera has started.

Main scenario:

- Application will call SwitchCamera method.
- The device camera will be switched between back and front accordingly.

Alternatives: -

Use case: StopCamera

Description: Stops the device camera.

Actor: Application.

Precondition: The camera has started.

Main scenario:

- Application will call the StopCamera method.
- The camera of the device will be stopped.

Alternatives: -

Use case: InitWithController

Description: Starts the camera.

Actor: Application.

Precondition: Permission to access the camera is granted by the device.

Main scenario:

- Initialize the camera.
- Connect the camera with the device's camera.
- Define the number of frames per second to take.
- Automatically calls ProcessImage method.

Alternatives:

- Abort the process if the camera doesn't exist.

Use case: ProcessImage

Description: Translates the gestures.

Actor: Application. **Precondition:** The camera has started.
Main scenario: • Apply the Hand Segmentation algorithm. • Apply the Hand Feature Extraction and Gesture Matching algorithm.
Alternatives: -

Use case: HandSegmentation
Description: Segment the hand from the frame. **Actor:** Application. **Precondition:** Frame retrieved from the camera.
Main scenario: • Convert frame color space to *YCbCr*. • Background subtraction. • Face removal. • Skin color extraction. • Morphology and image smoothing.
Alternatives: -

Use case: HandFeatureExtractionandGestureMatching
Description: Retrieve hand coordinates. **Actor:** Application. **Precondition:** Binary image retrieved from the hand segmentation process.
Main scenario: • Use *SURF* to extract features of the binary image retrieved. • Use bag of words to extract the *descriptors* of the *keypoints*. • Match extracted features (*descriptors*) using *SVM* and *FlannBased* matcher. • If there is a match return an index of the word, otherwise return -1
Alternatives: -

Use case: getString
Description: Extracts the word of a given index **Actor:** Application. **Precondition:** Index retrieved from gesture matching process.
Main scenario: • Find the word matching the given index in the database. • Retrieve the detected word and display it as text and audio if the matching is successful otherwise, return a not found message to the application.
Alternatives: -

4.1.2 Application Functional Requirements

Figure 4.2 shows the use-case diagram for the application while Table 4.2 shows the detailed description for the use cases in Figure 4.2. Listed below are the functional requirements:

- Start Translating/end Translating: Starts the camera and the real-time translation.
- Switch Camera: Switches between the device's front and back camera.
- Menu: Previews Contact Us, About Us, Privacy Policy options.
- Start Recording/end Recording: Records a video with the speech translation (Application screen) using Swift's Replay Kit framework.
- Speech to Text: Converts speech heard from the device's audio to text.
- Share text: The translated text is shared with other applications.
- Contact Us: User can contact the developer's by filling a form in the contact us page.
- Privacy Policy: User can read the privacy policy of the application.
- About Us: User can read information about the application and its developers.

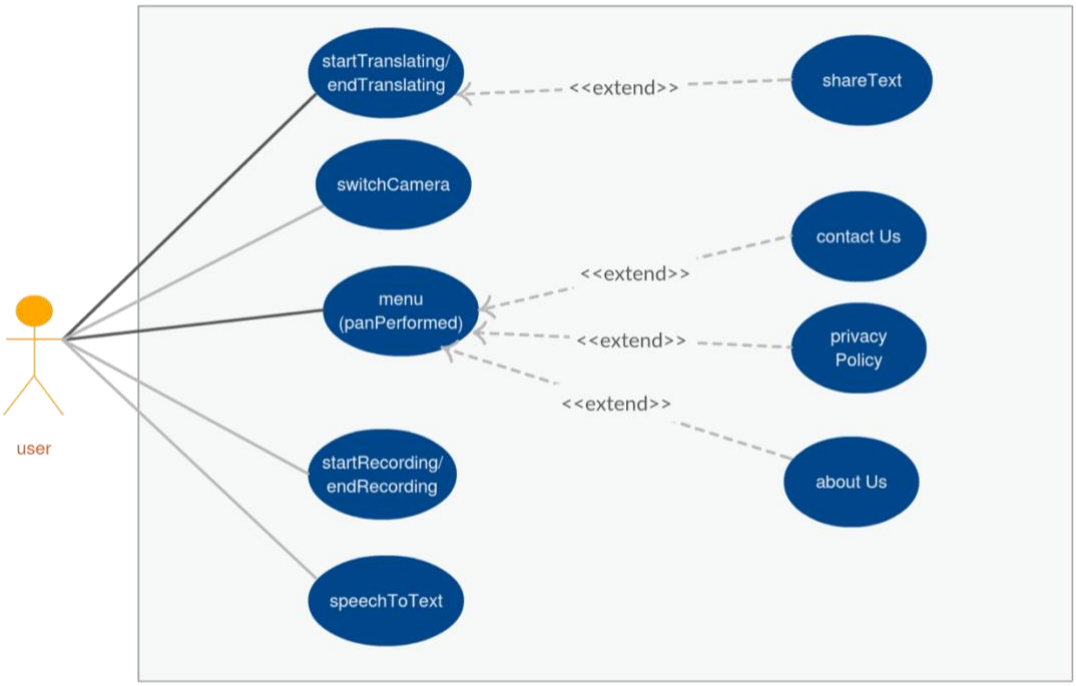


Figure 4.2 Application Use Case Diagram

Table 4.2 Application Use Case Description

Use case: Start Translating/end Translating
Description: Start or end gesturing with real-time translation to speech and text. **Actor:** User. **Precondition:** -
Main scenario: • User presses the start button. • The camera is shown in the UIImageView controller. • The system recognizes the gestures and starts translating them in real-time. • Speech and text translations are produced accordingly. • If there were no gestures from the user, no gesture translation will show. • If user presses the button again it indicates the end of gesturing by user.
Alternatives: -

Use case: Switch Camera
Description: Switches between front and back camera. **Actor:** User. **Precondition:** -
Main scenario: • User double taps the screen. • If front camera is opened, then it switches to the back camera and vice versa.
Alternatives: -

Use case: Menu
Description: Shows menu options for the user to view. **Actor:** User. **Precondition:** -
Main scenario:

- User slides the left side of the screen to open the menu.
- User chooses what to view from the list of options.
- When finished the menu option is closed.

Alternatives: -

Use case: Start Recording/end Recording

Description: Records a video of the camera window with the translated speech.

Actor: User.

Precondition: The camera has started.

Main scenario:

- User presses the recording button.
- Recording starts.
- User starts gesturing and a speech translation will be produced.
- The recording ends when the user presses the record button again.
- The user can choose to save the recorded video and share it with other applications.

Alternatives: -

Use case: Speech to Text

Description: Translates speech heard to text.

Actor: User.

Precondition: -

Main scenario:

- User presses speech to text button.
- Arabic speech heard by the device's microphone is translated to text and displayed in the application.
- The translated speech is discarded afterwards.

Alternatives: -

Use case: Share text
Description: Shares and saves the translated text of the signs. **Actor:** User. **Precondition:** Start/End translating button must be pressed.
Main scenario: • User presses share text. • The translated text is saved and a menu of where to save/share the text is displayed to the user. • The saved text is discarded afterwards.
Alternatives: • If user cancels, the saved text will be discarded.

Use case: Contact Us
Description: Previews our contact details. **Actors:** User. **Precondition:** Contact Us option is selected from the menu.
Main scenario: • A form for feedback is shown for the user to fill. • User sends the form by email or cancels.
Alternatives: -

Use case: About Us
Description: Shows the application details. **Actor:** User. **Precondition:** About Us option is selected from the menu.
Main scenario: • A detailed description about the application will be shown. • User closes the about us section.

Alternatives: -

Use case: Privacy Policy
Description: Previews the privacy policy of the application. **Actor:** User. **Precondition:** Privacy Policy option is selected from the menu.
Main scenario: • A detailed description of the privacy policy is shown. • User closes the privacy policy section.
Alternatives: -

4.1.3 Software and Hardware Requirements

To achieve all previously listed tasks, the following software and hardware requirements are needed:

4.1.3.1 Software requirements

- **Xcode version 9.0 and above:** Used for application design and implementation.
- **OpenCV 2.4.13:** Used for ArSL translation development.
- **Microsoft Project 2010:** Used for project planning.
- **Microsoft Word 2016:** Used for report writing.
- **Creatly:** Online tool for diagrams drawing. [22]

4.1.3.2 Hardware requirements

- A macOS High Sierra 10.13.3 device for development.
- iOS device of version 11.0 and above for testing.

4.2 Non-Functional Requirements

The non-functional requirements for the application are discussed below.

- **Availability:** Ability of application to work online and offline as well.
- **Simplicity:** At first time usage the user will be provided with a guide for the application's tools.

- **Performance:** The process time of the application is made reasonable to users and minimum resources are used.
- **Supportability:** The application is compatible with different versions of iOS.

4.3 Conclusion

This chapter discussed the functional and non-functional requirements regarding the *OpenCV* interaction with *Swift* and the application. The functional requirements were based on the survey that targeted the deaf people society in Muscat, and the non-functional requirements were concluded as a result of testing the application in the Testing Chapter. In the next chapter system design will be discussed.

Chapter 5: Project Design

Based on the requirements specification, we designed a suitable solution for deaf people to help them communicate easily with the other portion of the society that is not aware of the ArSL. We have followed a top-down design approach where we broke down the communication problem into several sub problems, solving each and building our final solution.

5.1 Product Features

The product features of the application are listed below:

- Offline work: The iOS application is able to work offline.
- Regular updates: Includes updates to the dataset containing the ArSL dictionary.
- Feedback and contact means: Users will be able to communicate with the developers quickly and easily through the application.
- Performance: Reasonable waiting time for the translation process is maintained.

5.2 User Interface

The GUI components are menu, buttons, textbox and a camera box. Through user interactions with the application by pressing buttons and through the sliding window the user will be able to use all the functionalities of the application. Moreover, the user should perform gestures within the boundaries of the camera box for the translation process.

Figure 5.1 contains an image view having the camera that captures the hand movements. The text bar below the camera view displays the translated text corresponding to the gesture sequence in the camera. Below the text bar are four buttons. The first button from the left side initiates the start translation/ end translation, the button following it is ShareText, then the next button is speechToText and the final button on the right records a series of gestures and translated text and stores them as a video.

Figure 5.1 Application Sample Screenshot (1)

Figure 5.2 Application Sample Screenshot (2)

Figure 5.2 contains the slider menu. It contains four buttons; the first button is About Us button which opens a page that contains the application details: developers' information, version of application and so on. The second button is Contact Us it launches a forum to allow the user to provide the feedback on the quality and features of the application. The last button is the Privacy Policy, which opens a page to display the privacy policy of using the application.

5.3 Interface to External Hardware and Software

5.3.1 Hardware interface:

- Device Camera: The device camera is a critical requirement to the ArSL translator since we are capturing user gestures.

5.3.2 Software interface:

- OS: iOS, versions from 11.0 and above are supported. The iOS is required to run the application.

– Database: *SQLite3* to obtain the gesture translation from a database of gestures corresponding to ArSL.

5.4 Data Storage

5.4.1 ArSL Translator Database Schema

In order to minimize the storage space required by the application for the gesture recognition phase, the database is stored as *SQLite3* [23] in the device and the database table as shown in Table 5.1:

Table 5.1 Gestures Database Schema

Field	**Data Type**	**Constraint**
index	Number	Primary key
word	Text	-

Shown in Table 5.1 is the **index** field which is a number that denotes the key for every word. It is needed when matching the descriptors to the word as it acts as a label. The **word** field is the Arabic word that represents a gesture and it is a text data type.

5.5 High Level Design

Figure 5.3 of the UML Class Diagram shows the high-level design of our project representing the dependency relation between the *OpenCV*, *Swift* and the application.

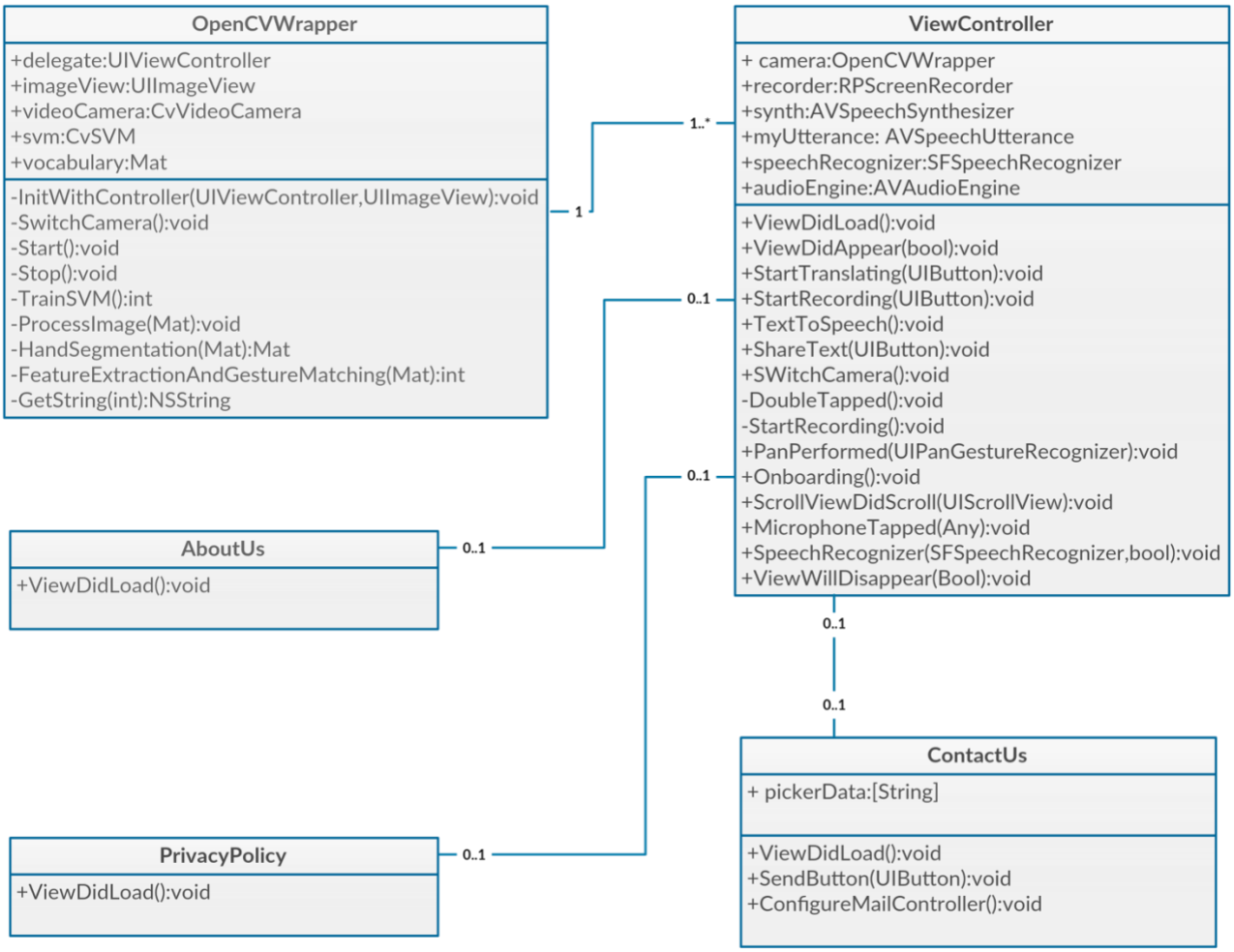


Figure 5.3 UML Class Diagram

5.6 Design verification

Examination walk-through the application has been able to verify our design against the requirements specified, this is achieved through the sequence diagrams as follows:

5.6.1 OpenCV interaction with Swift Sequence Diagrams

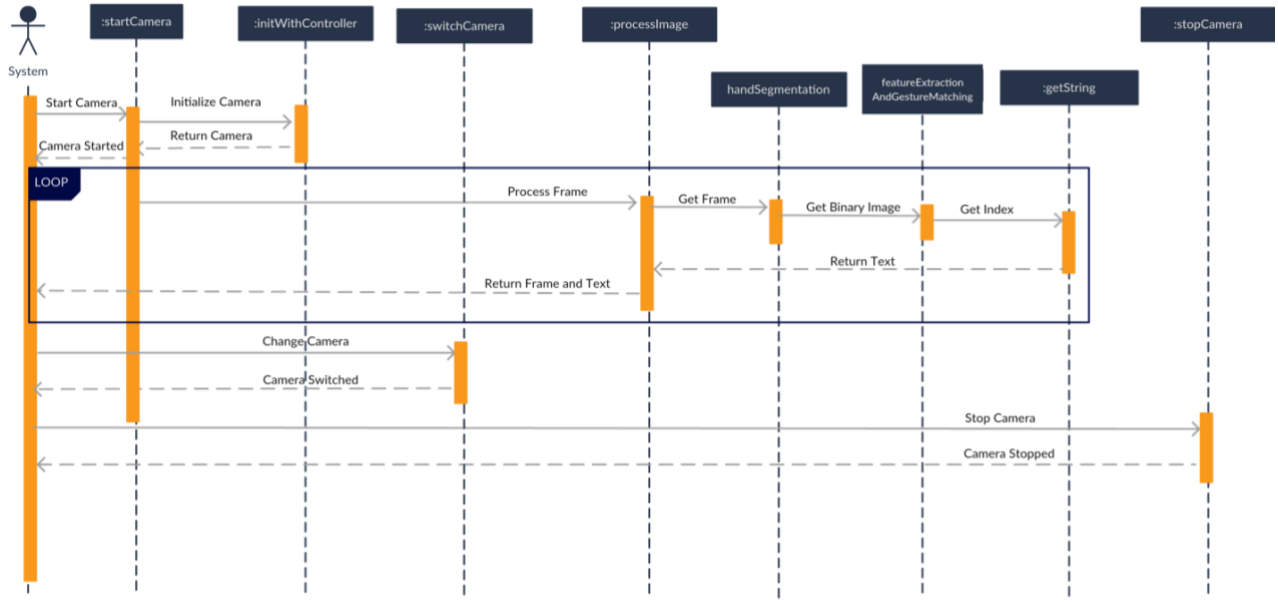


Figure 5.4 OpenCV interaction with swift Sequence Diagram

5.6.1.1 initWithController

Initially, the system will launch the camera. The initWithController binds the camera with the application's interface. In addition to that, it initializes the number of frames per second and sets the orientation to portrait mode and other properties as well.

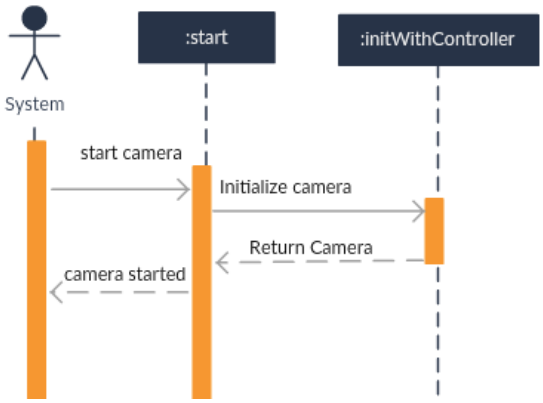


Figure 5.5 initWithController Sequence Diagram

5.6.1.2 SwitchCamera

SwitchCamera is responsible for allowing the user to change the camera choice from front camera to back camera and vice versa.

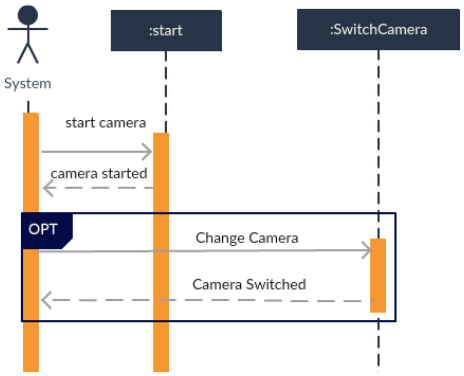


Figure 5.6 SwitchCamera Sequence Diagram

5.6.1.3 ProcessImage

ProcessImage processes the frame after the camera is launched automatically. Firstly, hand segmentation is performed by retrieving the hand object from the frame by performing a couple of processes. The result of this step is a binary image that proceeds to the next phase which is the hand feature extraction. The hand feature extraction and gesture matching phase uses a the smoothed binary image from the previous step to extract the features of the hand using *SURF*. These features are then used for matching the gestures with the trained data using *SVM* this then returns the index of the matched gesture which is the input of *getString*. Finally, *getString* matches the index to the Arabic word corresponding to the gesture and returns the word as a text to the application.

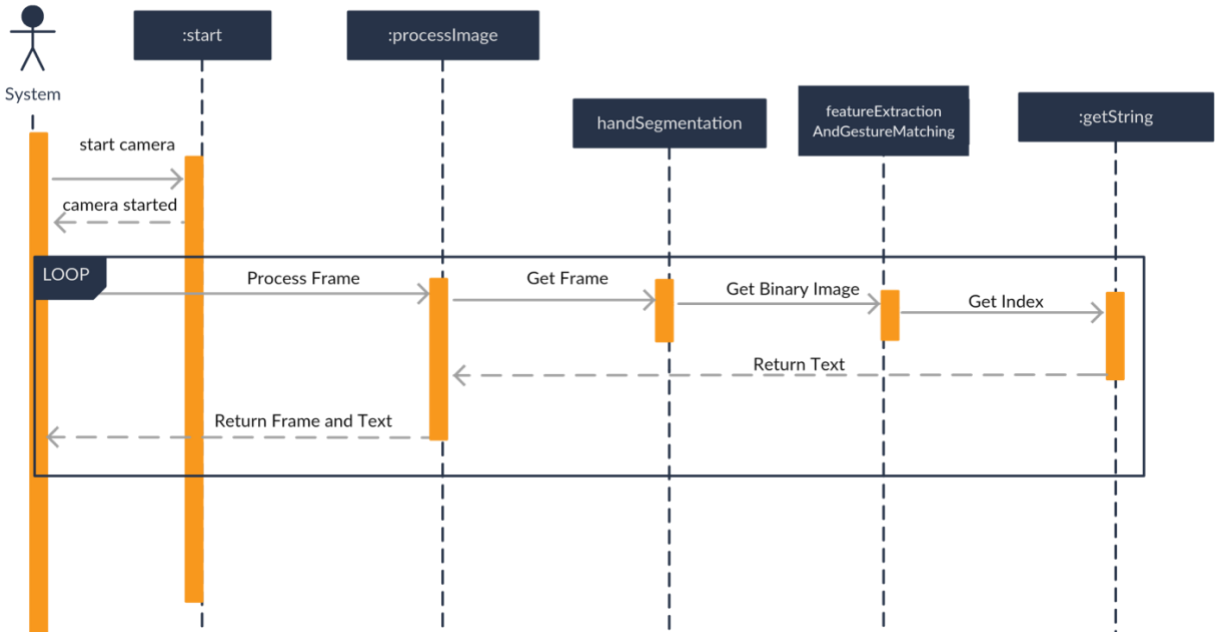


Figure 5.7 ProcessImage Sequence Diagram

5.6.1.4 StopCamera

Stop feature allows the user to stop capturing the frames and disables the camera.

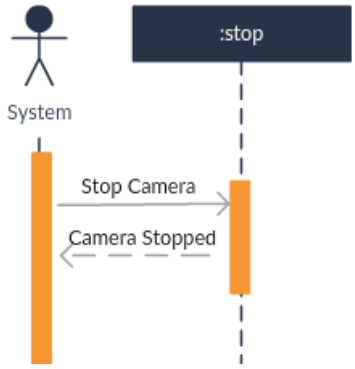


Figure 5.8 Stop Sequence Diagram

5.6.2 Application Sequence Diagrams

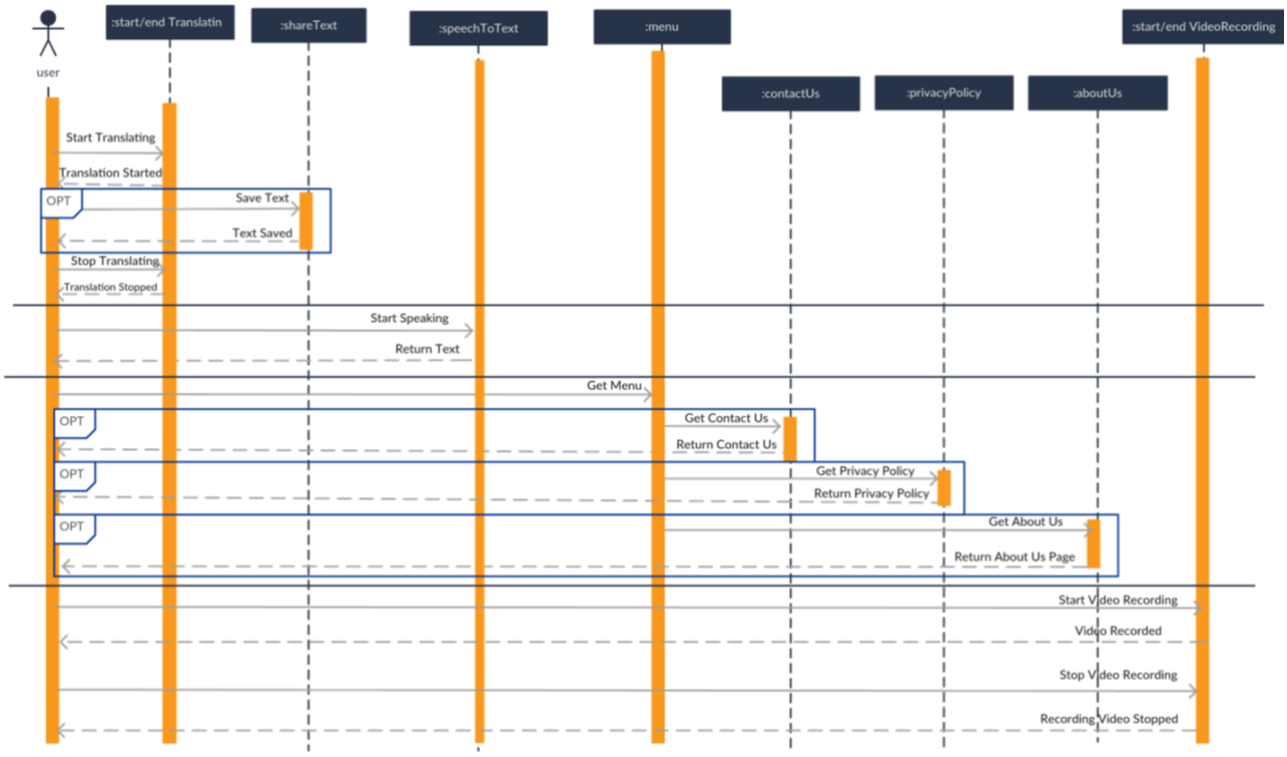


Figure 5.9 Application Sequence Diagram

5.6.2.1 Start/end Translation

The initial step start/end translation allows the user to start the process of gesture translation and the output returned from this process is the translated Arabic sentence. ShareText also requires the start/end translation feature to be performed first before this step, the output of this is a text file of the Arabic translated text from the sequence of gestures. This file can be shared through any other application upon the request of the user.

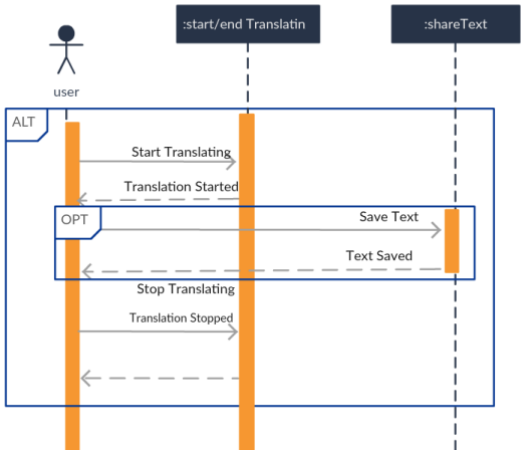


Figure 5.10 Start/End Gesture Sequence Diagram

5.6.2.2 Speech to Text

Initially, this feature is to support two-way communication. To help the other person who cannot speak the ArSL to use the application too to deliver their speech in text format to the deaf person.

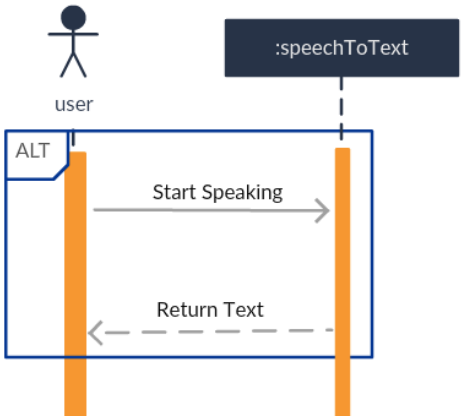


Figure 5.11 Speech To Text Sequence Diagram

5.6.2.3 Menu

Menu option contains: Contact Us, Privacy Policy and About Us. Contact Us launches a forum to allow the user to provide the feedback on the quality and features of the application. Privacy Policy, opens a page to display the privacy policy of using the application. About Us opens a page that contains the application details: developers' information, version of the application and so on.

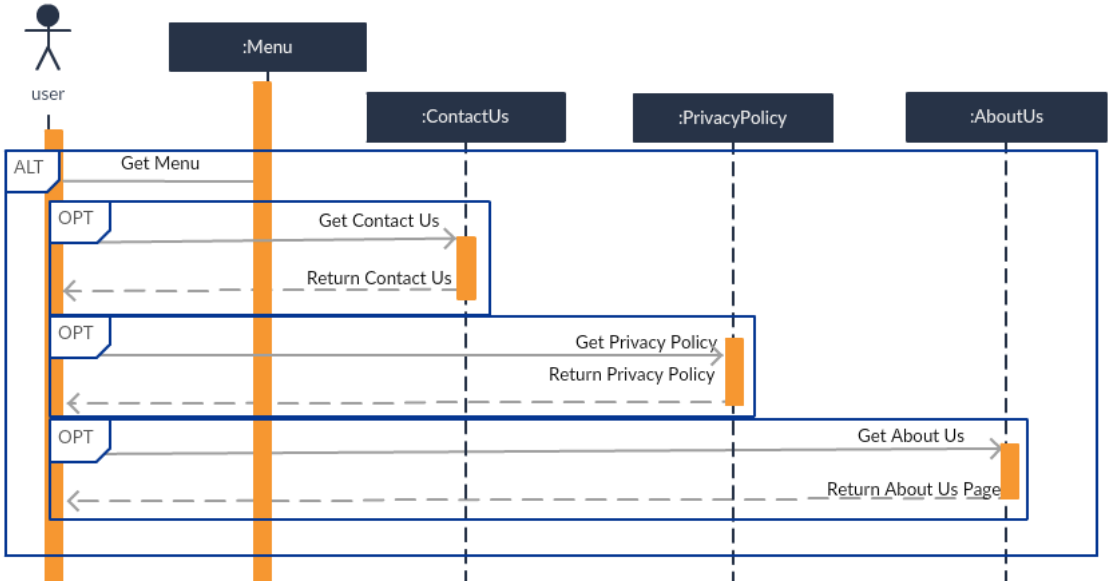


Figure 5.12 Menu Sequence Diagram

5.6.2.4 Start/End videoRecording

Allows the user to record a video of the translated text and sound outputs and the sequence of gestures on the screen.

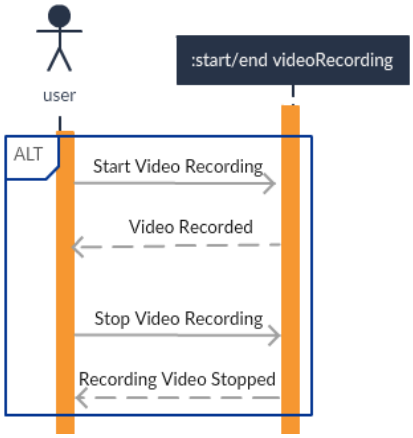


Figure 5.13 Start/End videoRecording Sequence Diagram

5.7 Conclusion

This chapter discussed the interaction with *OpenCV, Swift* and the application designs which are represented through sequence and class diagrams along with the database design that was briefly discussed.

Chapter 6: Project Implementation

In this chapter, the implementation of the four phases of the ArSL Translator: Training *SVM*, Hand Segmentation, Hand Feature Extraction and Gesture Matching is explained in detail including the functions used and the processes.

6.1 Coding

This section consists of the code of the application explained in detail.

6.1.1 ViewController.swift

This section contains some of the functionalities of the application such as: Text to Speech and Speech to Text.

6.1.1.1 Text to Speech

In this process the text translated is converted to speech. This is done by using a class called *AVSpeechSynthesizer* which is part of the *AVFoundation* framework. *AVSpeechSynthesizer* is used in iOS to produce synthesized speech from any text. To do that, an instance of *AVSpeechUtterance* is created to hold the text to be converted to speech. In Figure 6.1 an instance of *AVSpeechSynthesizer and AVSpeechUtterance* is created.

```
let synth = AVSpeechSynthesizer()
var myUtterance = AVSpeechUtterance(string: "")
```

Figure 6.1 Creating an instance of AVSpeechUtterance

In Figure 6.2 below:

- **Line 1:** passes the text to be spoken to *AVSpeechUtterance.*
- **Line 2:** defines the rate of the speaker as a normal speed of *0.5*.
- **Line 3:** defines the language of the speaker as Arabic since the default is English.
- **Line 4:** calls *speak* method on the *AVSpeechSynthesizer* instance with the *AVSpeechUtterance* instance passed as a parameter.

```
//text to speech
func textToSpeech()
{
    myUtterance = AVSpeechUtterance(string: TranslatedText.text!)
    myUtterance.rate = 0.5
    myUtterance.voice = AVSpeechSynthesisVoice(language: "ar-SA")
    synth.speak(myUtterance)
}
```

Figure 6.2 Text to Speech function

6.1.2 Speech to Text

The aim of this process is to convert Arabic speech to text. iOS have built-in *Speech* framework, which adopts *SFSpeechRecognizerDelegate*. Figure 6.3 creates an instance of *SFSpeechRecognizer* to handle speech recognition. The locale identifier *"ar-SA"* defines the language of speech as Arabic and this is needed to know which language the user is speaking.

```
private let speechRecognizer = SFSpeechRecognizer(locale: Locale.init(identifier: "ar-SA"))
```

Figure 6.3 Creating an instance of SFSpeechRecongizer

In order to use the speech recognizer, the user should allow the application to use the device's microphone. Figure 6.4 below shows the implementation of user authorization.

```
@IBAction func microphoneTapped(_ sender: Any) {


    //microphone
    microphoneButton.isEnabled = false

    speechRecognizer?.delegate = self as SFSpeechRecognizerDelegate

    SFSpeechRecognizer.requestAuthorization { (authStatus) in //1

        var isButtonEnabled = false

        switch authStatus { //2
        case .authorized:
            isButtonEnabled = true

        case .denied:
            isButtonEnabled = false
            print("User denied access to speech recognition")

        case .restricted:
            isButtonEnabled = false
            print("Speech recognition restricted on this device")

        case .notDetermined:
            isButtonEnabled = false
            print("Speech recognition not yet authorized")
        }

        OperationQueue.main.addOperation() {
            self.microphoneButton.isEnabled = isButtonEnabled
        }

    }
```

Figure 6.4 Speech User Authorization

1. Requests user's authorization for speech recognition.
2. Checks the status of the authorization and enables the microphone if authorized and otherwise disables it.

Figure 6.5 below shows the objects used for speech recognition and the function used to implement these objects to translate speech to text is shown in Appendix III.

```
private var recognitionRequest: SFSpeechAudioBufferRecognitionRequest?
private var recognitionTask: SFSpeechRecognitionTask?
private let audioEngine = AVAudioEngine()
```

Figure 6.5 Speech Recognition Objects

Line 1: handles speech recognition requests.

Line 2: stores the output of the recognition request.

Line 3: the audio engine responsible for providing audio input.

6.1.3 OpenCV.mm

6.1.3.1 initWithController

CvVideoCamera is a class in OpenCV which is used to display video from the device's camera and capture frames. The code in Figure 6.6 below initializes the camera and then it automatically calls *processImage* method after capturing each frame. The delegate is used to connect the camera to the swift code and display it in the application inside the image view controller.

```
//Initalize the camera
- (id)initWithController:(UIViewController<CvCameraDelegate>*)c andImageView:(UIImageView*)iv
{
    delegate = c;
    imageView = iv;
    videoCamera = [[CvVideoCamera alloc] initWithParentView:imageView];
    self->videoCamera.defaultAVCaptureVideoOrientation =  AVCaptureVideoOrientationPortrait; //Default orientation is portait
    videoCamera.defaultAVCaptureDevicePosition = AVCaptureDevicePositionBack; //Intially back camera is opened
    videoCamera.defaultFPS = 30; // How often 'processImage' is called, (30 frames per second)
    videoCamera.delegate = self;

    count = 0;
    return self;
}
```

Figure 6.6 initWithController sample code

6.1.3.2 switchCamera, start, stop

switchCamera method is called when the user double taps on the screen and if the current camera is the back camera then it is switched to front camera and vice versa. Naturally, *start* method starts the camera and *stop* method stops the camera. Figure 6.7 below shows the code written for these methods.

```
//Change Camera Position
- (void)switchCamera
{
    [self->videoCamera switchCameras];
}

//Start the camera
- (void)start
{
    [videoCamera start];
}

//Stops the camera
- (void)stop
{
    count = 0;
    [videoCamera stop];
}
```

Figure 6.7 switchCamera, start, stop sample code

6.1.3.3 processImage

This method is called every time the frame is captured. It processes each frame and translates matched gestures to text. The output consists of the translated text which is returned to the application's interface.

```
binaryImg = [self handSegmentation:img];
index = [self handFeatureExtractionandGestureMatching:binaryImg];
[self getString:index];
```

Figure 6.8 processImage sample code

6.1.3.4 handSegmentation

This method consists of many different operations which are discussed in detail below. It takes as input the original frame captured from the camera in form of a matrix and returns a matrix of the processed image which is a binary image where the hand is in white color and the rest of the image is in black color.

```
- (cv::Mat)handSegmentation:(cv::Mat &)img
```

Figure 6.9 handSegmentation header

Background Retrieval

The frame captured after one second of starting the camera is considered as a background and is saved in a matrix called *bgImage*. The background is the converted to *YCbCr* color space. Figure 6.10 below shows the sample code of background retrieval.

```
//1. get background frame
if (count == 30) // (1)
{
    img.copyTo(bgImage); // (2)

    //convert background image to YCrCb
    cv::cvtColor(bgImage,bgImage,COLOR_BGR2YCrCb); // (3)
    count++;
}
else if (count < 30)
    count++;
```

Figure 6.10 Background Retrieval

1. Count is used to count the number of captured frames before one second. After one second the total number of captured frames is equal to 30 since the camera captures 30 frames per second.
2. *copytTo* is an *OpenCV* function which one of its functionalities is to copy a matrix into another matrix and in this case, the background captured in *img* is copied to *bgImage* (*img* is the frame captured from the camera and this frame is in the form of a matrix was passed as an input to *handSegmentation method*).
3. *cvtColor* is used to convert the background color space from *BGR* to *YCbCr* color space. It takes as input the following:
   - bgImage: passed as both source and destination matrix since the background is needed in *YCbCr* color space only.
   - COLOR_BGR2YCrCb: it is an *OpenCV* color space conversion code which is used to convert *BGR* image to *YCbCr*.

Convert frames color space to YCbCr

All incoming frames are converted to *YCbCr* color space.

```
//2. Convert incoming img to ycrcb to match template
cv::cvtColor(img, ycrcb, COLOR_BGR2YCrCb);
```

Figure 6.11 Frames color space Conversion

- img: input array source
- ycrcb: output array source (destination).
- COLOR_BGR2YCrCb: color conversion code.

Background Subtraction

The absolute difference between the current frame *ycrcb* and the background *bgImage* is calculated so that when performing skin color extraction, the background is excluded from the threshold.

```
//3. Background subtraction
Mat diffImage = abs(ycrcb-bgImage);
```

Figure 6.12 Background Subtraction

Split Image

The *diffImage* from the step above is split into three channels to later perform thresholding in each channel separately.

```
//4. Split image into 3 channels
Mat chan[3];
split(diffImage,chan);
Mat y = chan[0];
Mat Cr = chan[1];
Mat Cb = chan[2];
```

Figure 6.13 Split Image

split takes as input the matrix *diffImage* and outputs a matrix of matrices *chan*. The first matrix in *chan* is the *Y* channel, the second is the *Cr* channel and the third is the *Cb* channel.

Thresholding and Binarization

Each channel in the *YCbCr* color space is converted to binary separately. Threshold is a function in *OpenCV* which *binarizes* an image.

```
//5. Thresholding and Binarization
threshold(y, y, 54, 255, THRESH_BINARY);
threshold(Cr, Cr, 131, 255, THRESH_BINARY);
threshold(Cb, Cb, 110, 255, THRESH_BINARY);
```

Figure 6.14 Thresholding and Binarization

The threshold method takes as input the image to be *binarized*, the destination matrix after *binarization*, the threshold value, the maximum value and the type of thresholding. Since we want to *binarize* each channel the maximum value is *255* (white). The constant *THRESH_BINARY* means the type of thresholding is binary.

Morphology

The morphological operation *open* is applied to reduce the noise in each channel.

```
//6. Morphology
int elementSize = 2;
Mat kernel = getStructuringElement(MORPH_RECT,
                                   cv::Size(2 * elementSize + 1, 2 * elementSize + 1),
                                   cv::Point(elementSize,elementSize));
morphologyEx(y, y, MORPH_OPEN, kernel);
morphologyEx(Cr, Cr, MORPH_OPEN, kernel);
morphologyEx(Cb, Cb, MORPH_OPEN, kernel);
```

Figure 6.15 Morphology on each channel

To apply morphology, the structuring element *kernel* needs to be defined by calling the method *getStructuringElement* which takes as input the following:

- MORPH_RECT: defines the shape of the structuring element as a rectangle.
- cv::Size(): defines the size of the structuring element as *5 by 5*.
- cv::Point(): places the anchor position in the middle.

The method *morphologyEx* each time takes as input the channel matrix (*Y*, *Cr*, *Cb*) both as source and destination to modify each channel permanently. *MORPH_OPEN* is the type of morphological operation and the *kernel* is the structuring element.

Addition

The three channels (*Y*,*Cr*,*Cb*) are combined by simple addition operation.

```
//7. addition
diffImage = y + Cr + Cb;
```

Figure 6.16 Combining Y,Cr,Cb channels

Binary Image Morphology

After combining the three channels in the previous step, *close* morphological operation is applied twice because it yielded a better noise reduction output. However, in this step the size of the structuring element is 7 instead of 5 like the previous morphological operation because through trial and error when the size was 7 the noise is reduced more than when it is 5 and there wasn't much difference in speed but when the size was 9 the speed of this process slowed down significantly. Hence, size = 7 was chosen. Also, the shape of the structuring element is an ellipse because the hand shape is closer to an ellipse than a rectangle.

```
//8. morphology for binary image
elementSize = 3;
kernel = getStructuringElement(MORPH_ELLIPSE,
                               cv::Size(2 * elementSize + 1, 2 * elementSize + 1),
                               cv::Point(elementSize,elementSize));
morphologyEx(diffImage, diffImage, MORPH_CLOSE, kernel);
morphologyEx(diffImage, diffImage, MORPH_CLOSE, kernel);
```

Figure 6.17 Morphology on combined channels

Apply Mask

The mask is used to copy only the region of interest of the original image *diffimage* into a destination *ycrcb*.

```
//9. Apply mask
ycrcb.copyTo(ycrcb, diffImage);
```

Figure 6.18 Applying mask

- *ycrcb*: used both as input and destination matrix.
- *diffImage*: it is a mask that indicates which matrix elements needs to be copied.

Face Detection

The face is detected using *Haar feature based Cascade Classifier*. *OpenCV* already contains pre-trained classifiers for the face which are stored in an XML file. The following code in Figure 6.19 is loading the XML classifier into our program.

```
CascadeClassifier faceCascade;
NSString* const faceCascadeFilename = @"haarcascade_frontalface_alt";
NSString* faceCascadePath = [[NSBundle mainBundle] pathForResource:faceCascadeFilename ofType:@"xml"];
faceCascade.load([faceCascadePath UTF8String]);
```

Figure 6.19 Loading XML face classifier

After loading the classifiers, faces in the image are searched. *CascacadeClassifier::detectMultiScale* function detects faces which are of different sizes. But since *detectMultiScale* takes a grayscale image as input the captured frame is first converted to grayscale and then equalized.

```
//10. face detection
Mat grayscaleFrame;
cvtColor(originalFrame, grayscaleFrame, CV_BGR2GRAY);
equalizeHist(grayscaleFrame, grayscaleFrame);

std::vector<cv::Rect> faces;
faceCascade.detectMultiScale(grayscaleFrame, faces, 1.1, 2, HaarOptions, cv::Size(60, 60));
```

Figure 6.20 Face detection

In faceCascade.detectMultiScale:

- *grayscaleFrame*: input image in grayscale
- *faces*: destination vector of rectangles containing the detected faces positions
- *1.1*: scale factor; how much the image size is reduced at each image scale
- *2*: number of minimum neighbors, it affects the quality of the faces detected, the higher the number is the less faces detected.
- *HarrOptions*: flag
- *cv::Size(60,60)*: the minimum size of the face to be detected

Face Removal

If there are faces found in the previous step, then they are removed to avoid them being extracted in the skin color extraction phase. Since *faces* returns the positions of the detected faces, a region of interest is created, and the face is covered by a filled circle.

The *center* stores the center x and center y position of the detected face. Then the *circle* function draws a circle over the face

```
bool flag = true;
for (int i = 0; i < faces.size(); i++)
{
    //11. face removal
    cv::Point center=cv::Point( faces[i].x + faces[i].width * 0.5, faces[i].y + faces[i].height * 0.5 );
    cv::circle( ycrcb, center,faces[i].width/1.5,cvScalar( 255, 0, 255 ), CV_FILLED, 8, 0 );

    flag = false;
}
```

Figure 6.21 Face Removal

In cv::Point center:

- x-value: center x position of the detected face (x + width / 2)
- y-value: center y position of the detected face (y + height / 2)

In cv::circle:

- ycrcb: source image
- center: the center of the circle calculated in the previous step
- faces[i]/width/1.5: radius of the circle
- cvScalar(255,0,255): color of the circle defined as black
- CV_FILLLED: circle drawn is fully filled with a color
- 8: default value of the line type
- 0: default value of the shift

Skin color extraction

The hand is segmented using skin color thresholding. The *ycrcb* matrix is thresholded using the values given by D.Chai [19].

Equation 6.1 Skin Color Threshold Values

$$(\mathbf{54} \leq \boldsymbol{Y} \leq \mathbf{163}\ \boldsymbol{and}\ \mathbf{131} \leq \boldsymbol{Cr} \leq \mathbf{157}\ \boldsymbol{and}\ \mathbf{110} \leq \boldsymbol{Cb} \leq \mathbf{135})$$

The *inRange* is an *OpenCV* function which is used to detect an object based on a range of values and then returns a binary image with the object that satisfies the range as white and the rest of the image as black.

```
//13. Skin extraction (check the threshold)
Mat binaryImg;
inRange(ycrcb, Scalar(54, 127, 77), Scalar(163, 173, 127), binaryImg);
```

Figure 6.22 Skin Color Extraction

- *ycrcb*: source matrix
- *Scalar(54,127,77):* lower boundary scalar where Y > 54, Cr > 127, Cb > 77
- *Scalar(163, 173, 127)* : higher boundary scalar where Y < 163, Cr < 173, Cb < 127
- *binaryImg*: output matrix

Morphology

Open morphological operation is used to reduce the noise after extracting the hand based on skin color.

```
//14. Morphology
morphologyEx(binaryImg, binaryImg, MORPH_OPEN, kernel);
```

Figure 6.23 Open Morphological Operation

- *binaryImg*: source and destination matrix
- MORPH_OPEN: defines the morphological operation as open
- *kernel*: the same structuring element used in Figure 6.17.

Gaussian Filtering

Gaussian filtering is used to smooth the image. Gaussian Filter is done by convolving each element in the matrix with a *Gaussian kernel* and then summing them to produce the output. The *OpenCV* function *Gaussianblur* blurs the image using Gaussian filter.

```
//15. Gaussian Filter
GaussianBlur(binaryImg, binaryImg, cv::Size(5,5), 0);

return binaryImg;
```

Figure 6.24 Gaussian Filtering

- binaryImg: source and destination matrix
- cv::Size(5,5): size of the kernel is defined as 5 by 5
- 0: standard deviation of the image

Finally, *handSegmentation* returns *binaryImg*.

6.1.3.5 CreateTrainDataUsingBow

This function is only used once, and its main purpose is to create a trained data to be used by *SVM* later. Figure 6.25 shows the objects used to create the trained data.

```
cv::Ptr<cv::DescriptorMatcher> matcher = cv::DescriptorMatcher::create("FlannBased");
cv::Ptr<cv::DescriptorExtractor> extractor = new cv::SurfDescriptorExtractor();
cv::BOWImgDescriptorExtractor dextract( extractor, matcher );
cv::SurfFeatureDetector detector(400,4,2,false);
cv::BOWKMeansTrainer bow( cluster,  cv::TermCriteria(CV_TERMCRIT_EPS+CV_TERMCRIT_ITER, 10, FLT_EPSILON), 1, cv::KMEANS_PP_CENTERS );
```

Figure 6.25 Objects used to create trained data

- *matcher*: a *FLANN* based matcher.
- *extractor*: *SURF* feature point extractor, it is used to compute the *descriptors* and *keypoints* of an image.
- *dextract*: the extractor is used to compute image *descriptor* and *keypoint* sets and the matcher is used to find for each *keypoint descriptor* the nearest word from the vocabulary.
- *detector*: instance of *SURF* feature detector, it is used to detect 400 features of an image. We have arrived at this number after trying 400, 500, 1500, 200 and through trial and error 400 was found to be the most suitable number.
- *bow*: bag of words used to train the vocabulary, it is based on *kmeans* – approach.

In Figure 6.26 the features of the image are extracted one by one and then added to *bow*.

```
for (int j = 1 ; j <= numWords; j++)
{
    for(int i = 1; i <= sample; i++)
    {
        path = [NSString stringWithFormat:@"image%d_%d.jpg",j,i]; //Read Image by Image
        UIImage* img1 = [UIImage imageNamed:path];
        UIImageToMat(img1, image);
        std::vector<cv::KeyPoint> keypoints;
        cv::Mat descriptors;
        detector.detect( image, keypoints); //(1)
        extractor->compute( image, keypoints, descriptors); //(2)
        if ( !descriptors.empty() )
        {
           bow.add( descriptors ); //(3)
        }
    }
}
```

Figure 6.26 Extracting the features and creating the bow

1. Detects the image *keypoints*.
2. Computes the *descriptors* of a given set of image *keypoints*.
3. The computed *descriptors* are added to *bow*.

In Figure 6.27 below the vocabulary is created.

```
vocabulary = bow.cluster(); //(1)
dextract.setVocabulary(vocabulary); //(2)
for (int j = 1 ; j <= numWords; j++)
{
   for(int i = 1; i <= sample; i++)
   {
       path = [NSString stringWithFormat:@"image%d_%d.jpg",j,i];
       UIImage* img1 = [UIImage imageNamed:path];
       UIImageToMat(img1, image);

       std::vector<cv::KeyPoint> keypoints;

       detector.detect( image, keypoints);
       cv::Mat desc;
       dextract.compute( image, keypoints, desc );
       if ( !desc.empty() )
       {
           train.push_back( desc );   //(3)
           response.push_back((float)j-1); //(4)
       }
   }
}
```

Figure 6.27 Create Vocabulary

1. Clusters the *descriptors* and returns the vocabulary.
2. Sets the visual vocabulary.
3. Updates the training data which stores the bag of words *descriptors*.
4. Update the response data which stores the labels for each *descriptor*.

6.1.3.6 TrainSVM

This function is used to train the *SVM classifier* and it is called only once.

```
svm.train_auto(train, response, cv::Mat(), cv::Mat(), params);
```

Figure 6.28 train_auto code

- *train*: matrix of the trained data created in *CreateTrainDataUsingBow.*
- *response*: matrix of the responses data created in *CreateTrainDataUsingBow.*
- *params*: it is of type *CvSVMParams, train_auto* selects the optimal values of *CVSVMParams*.

6.1.3.7 Feature Extraction and Gesture Matching

The binary image returned by hand segmentation is used to extract its features using *BoW* extractor that uses *SURF* to extract features and *FLANN* matcher. This process is shown in Figure 6.29 below.

```
detector.detect( binaryImg, keypoints);
dextract.compute( binaryImg, keypoints, desc_bow );
```

Figure 6.29 Current Frame Feature Extraction

- **Line 1:** *SURF detector* computes the *keypoints* of the binary image.
- **Line 2:** Bag of Words *dextract* computes the *descriptors* of the binary image.

Then *SVM* is used to classify the *descriptors* computed from the *BoW* extractor. Figure 6.30 shows the implementation *predict*.

```
if(!desc_bow.empty()){
    res = svm.predict(desc_bow);
}
else{
    res = -1.1;
}
```

Figure 6.30 SVM predict code

- *desc_bow*: computed description of the image.
- *res*: the label which represents the index of the word detected.

6.2 Conclusion

In this chapter, the implementation of the four phases of gesture recognition: Training *SVM*, Hand Segmentation, Hand Feature Extraction, Gesture Matching were discussed.

Chapter 7: Project Testing (Evaluation)

After having finished the project implementation, testing is a crucial step to determine if the project has successfully met all the requirements and specifications within the context limited by the constraints specified.

The testing chapter is divided into three main aspects: Metrics specification, Experimental setup for these metrics and the inputs for these experiments.

7.1 Verification

This section consists of testing each unit of the system.

7.1.1 Unit Testing

Unit testing is done to test each unit of the system. The Table 7.1 describes our findings:

Table 7.1 Unit Testing Modules

Module	Input	Expected Output	Validity	Reference to figure
Background Subtraction	Camera feed	Image without background	No	-
Hand Segmentation	Image without background	Binary Image	Yes	Figure 7.1
Image Processing	Binary Image without background	Smoothed Binary image without noise	Yes	Figure 7.2
Features Extraction (Using *SURF)*	Smoothed Binary Image without Background	Set of features of the hand (*descriptors* and *key points* of the image)	Yes	Figure 7.3
Trained *SVM*	Predefined set of binary images	Stores features of the gestures	Yes	Figure 7.4 Figure 7.5
Gesture Matching	Set of features from feature extraction step and the trained *SVM*	Identification of matched image	Yes	-
Text to Speech Conversion	Translated text	Sound Output	Yes	-
Speech to text conversion	Sound Input	Text Output	Yes	Figure 7.6
Storing a video of the translation	Sequence of Gestures on the screen	Video recording	Yes	Figure 7.7

7.1.1.1 Background Subtraction

The first frame of the 30 frames per second is taken and is converted to *YCbCr*. We are subtracting the background from every frame. However, background subtracting causes color leakage. In addition to that, it is not practical since the background may change during the gesturing process, so we cannot assume that the first frame will always be free of the hand object.

7.1.1.2 Hand Segmentation

We are extracting the hand using skin color extraction, Figure 7.1 below shows the hand as a white blob. However, in this stage the hand contains noise because of the thresholding.

Figure 7.1 Image with noise

7.1.1.3 Image Processing

The noise is removed from the binary image using morphological operations and Gaussian Filtering. Gaussian Filtering is used to remove the noise around the edges of the hand as shown in Figure 7.2.

Figure 7.2 Image after processing

7.1.1.4 Feature Extraction (Using SURF)

We have used *SURF* detector function to get the key points, *BoW* extractor to get the *descriptors* of the *keypoints* as shown in Figure 7.3.

```
Frame Features:
[0.0060975607, 0, 0, 0.0060975607, 0.0060975607, 0.0060975607, 0.0060975607, 0.0060975607,
0.024390243, 0, 0.018292682, 0.024390243, 0, 0, 0, 0, 0.012195121, 0.0060975607, 0, 0,
0.048780486, 0.0060975607, 0.0060975607, 0.012195121, 0.012195121, 0.0060975607, 0.0060975607,
0.030487804, 0.012195121, 0.0060975607, 0, 0.0060975607, 0, 0.012195121, 0.018292682,
0.073170729, 0, 0, 0, 0.036585364, 0.0060975607, 0, 0, 0, 0, 0.0060975607, 0.024390243,
0.024390243, 0.018292682, 0.012195121, 0.0060975607, 0, 0, 0, 0.0060975607, 0.030487804,
0.018292682, 0, 0.0060975607, 0.024390243, 0.012195121, 0.018292682, 0, 0, 0, 0, 0.0060975607,
0.0060975607, 0, 0, 0.012195121, 0.0060975607, 0.024390243, 0.018292682, 0, 0.0060975607,
0.0060975607, 0.0060975607, 0, 0.030487804, 0, 0, 0.0060975607, 0, 0, 0.012195121, 0, 0, 0, 0,
0.018292682, 0.012195121, 0.012195121, 0, 0, 0, 0, 0, 0.054878049, 0, 0, 0.0060975607, 0,
0.0060975607, 0, 0.036585364, 0, 0.0060975607, 0.0060975607, 0.0060975607, 0, 0, 0.0060975607,
0.0060975607, 0.0060975607, 0, 0, 0.024390243, 0.012195121, 0.012195121]
```

Figure 7.3 Set of features of a single frame

7.1.1.5 Trained SVM

A sample of binary images of uniform size was used for feature extraction by computing their *keypoints* and storing their *descriptors*. We constructed a vocabulary of the *BoW*, and we clustered them with the help of *Kmeans* algorithm. The *BoW* extractor was used to compute the *descriptors* of all the binary images (Figure 7.4). The output of this stage is the training set and their corresponding labels (Figure 7.5). The training set and their labels are fed into the *SVM* for training.

```
********************************************************************************************
image2_54 descriptor:
[0, 0.0053475937, 0, 0, 0, 0, 0, 0.0053475937, 0.15508021, 0.0053475937, 0, 0, 0.0053475937, 0, 0, 0, 0,
0.026737969, 0.0053475937, 0, 0, 0, 0, 0, 0, 0.010695187, 0, 0, 0, 0, 0, 0.0053475937, 0.0053475937,
0.021390375, 0.19786097, 0.037433155, 0, 0.0053475937, 0, 0.010695187, 0.0053475937, 0.01604278, 0, 0,
0.0053475937, 0, 0, 0, 0, 0.010695187, 0.01604278, 0, 0, 0.010695187, 0.0053475937, 0, 0, 0.037433155,
0.01604278, 0, 0.090909094, 0, 0, 0, 0, 0.010695187, 0, 0, 0, 0.01604278, 0, 0.0053475937, 0.0053475937, 0, 0,
0.0053475937, 0.03208556, 0.010695187, 0, 0, 0.01604278, 0, 0, 0, 0.042780749, 0, 0.0053475937, 0, 0.0053475937,
0, 0.01604278, 0, 0, 0.021390375, 0.0053475937, 0, 0, 0, 0.021390375, 0, 0, 0.0053475937, 0, 0, 0, 0.0053475937,
0, 0.037433155, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.01604278]
********************************************************************************************



********************************************************************************************
image2_55 descriptor:
[0.006849315, 0, 0, 0, 0.006849315, 0, 0, 0, 0.1369863, 0.02739726, 0, 0.01369863, 0, 0, 0, 0, 0, 0,
0.020547945, 0, 0, 0, 0.020547945, 0, 0, 0, 0, 0.01369863, 0, 0, 0.006849315, 0, 0, 0, 0.006849315, 0.24657534,
0.02739726, 0, 0.006849315, 0.006849315, 0.006849315, 0.01369863, 0.020547945, 0, 0, 0, 0, 0, 0, 0, 0, 0.01369863,
0.01369863, 0, 0, 0.006849315, 0, 0.006849315, 0, 0, 0, 0, 0.08219178, 0, 0, 0, 0, 0, 0, 0, 0, 0.006849315, 0,
0, 0.020547945, 0, 0, 0.01369863, 0.02739726, 0, 0, 0, 0, 0, 0, 0, 0.061643835, 0, 0, 0, 0, 0, 0, 0, 0.01369863, 0,
0.034246575, 0, 0, 0, 0, 0.01369863, 0, 0, 0, 0, 0, 0, 0, 0, 0.047945205, 0, 0, 0, 0, 0, 0, 0, 0.006849315,
0.020547945, 0, 0.02739726]
********************************************************************************************
```

Figure 7.4 sample of training SVM descriptors

```
******************
Index: 1

******************
Index: 1

******************
Index: 1

******************
Index: 1

******************
Index: 1

******************
Index: 1

******************
Index: 1

******************
Index: 1

******************
Index: 1
```

Figure 7.5 Index of gesture

7.1.1.6 Gesture Matching

The *descriptors* that we got from feature extraction using *SURF* are fed into *SVM.predict()* function for gesture matching with the existing dataset from the trained *SVM* step. The output of the function are labels indicating the index of the gesture. The *SQLite3* containing the crossholding words to the index of the gestures. The database is used to match the index to the word. The word is finally displayed in the text label on the screen for the user.

7.1.1.7 Text to Speech Conversion

As the gestures are translated to the text, it is also heard as audio.

7.1.1.8 Speech to Text Conversion

Additional functionality is the speech that is converted to text (Figure 7.6).

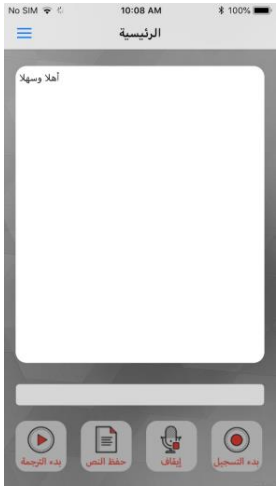


Figure 7.6 Text that was converted from speech

7.1.1.9 Storing a video of the translation

To share a video of the gesture sequence and the translated text, video recording is enabled (Figure 7.7).

Figure 7.7 Video stored of translationInterface Testing

The user requirements are tested at every step and the description of every task is mentioned along with the screenshot of the interface in the list below.

User Manual

Figure 7.8 User Manual Sample Screenshot

Figure 7.9 User Manual Expected Output

Description	This is the user manual. It is displayed to the user upon first launch of the application.
Input	The user presses the red button to start using the application.
Validation Status	Complete

About Us

Figure 7.10 About Us Sample Screenshot

Figure 7.11 About Us Expected Output

Description	This is the about us option. It is accessed through the sliding menu on the left side of the screen. It provides information about the application and developers.

Input	User drags the left of the screen or presses on the menu button and then presses on the first button "حول البرنامج"
Validation Status	Complete

Privacy Policy

Figure 7.12 Privacy Policy Sample Screenshot

Figure 7.13 Privacy Policy Expected Output

Description	This is the privacy policy option. It is accessed through the sliding menu on the left side of the screen. It provides information about the privacy policy of the application.
Input	User drags the left of the screen or presses on the menu button and then presses on the first button "سياسة الخصوصية"
Validation Status	Complete

Contact Us

Figure 7.14 Contact Us Sample Screenshot (1)

Figure 7.15 Contact Us Expected Output (2)

Figure 7.16 Contact Us Expected Output (3)

Figure 7.17 Contact Us Expected Output (4)

Description	This is the contact us option. It is accessed through the sliding menu on the left of the screen. It allows the user to contact the developers in case of any improvement on the application or if he/she faces a problem or in case of suggestions.
Input	1. User drags the left of the screen or presses on the menu button and then presses on the second button "تواصل" 2. The user then selects the title from the scrollable list and inputs the text in the text field. Finally, the user clicks on send. 3. After the user clicks on send, the mail application is launched. 4. The user hits the send button.
Validation Status	Complete

Translate Text

Figure 7.18 Translate Text Sample Screenshot

Figure 7.19 Translate Text Expected Output

Description	This option "بدء الترجمة" enables the user to start using the translation feature through performing gestures and seeing the translated text in the text box below the camera view.
Input	The user presses on the first button from the left with the label " بدء الترجمة".
Validation Status	Complete

Share Translated Text

Figure 7.20 Share Text Sample Screenshot

Figure 7.21 Share Text Expected Output (1)

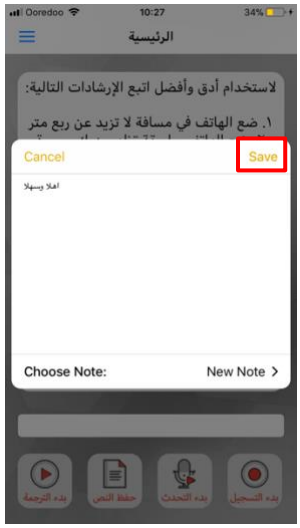


Figure 7.22 Share Text Expected Output (2)

Description	This option "حفظ النص" enables the user to save the translated text and share it upon preference.
Input	1. The user presses on the second button from the left with the label "حفظ النص" this previews a series of option to share the text or save it. 2. Upon choice in this case to choose to save it in a note, the translated text is displayed before saving.
Validation Status	Complete

Speech to Text

Figure 7.23 Speech to Text Sample Screenshot

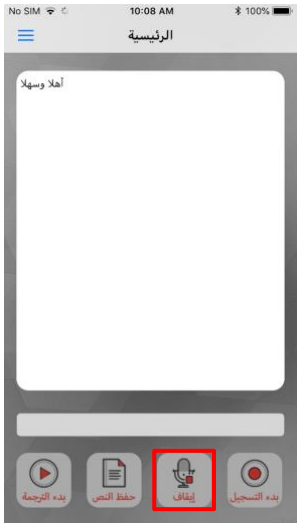


Figure 7.24 Speech to Text Expected Output

Description	This option "بدء التحدث" allows conversion of speech to text to support two-way communication.
Input	The button with the label "بدء التحدث" is pressed and the other person begins to speak in Arabic. The text is seen to be translated to text on the screen.
Validation Status	Complete

Record Translation (Video + Audio Generated + Translated Text)

Figure 7.25 Record Sample Screenshot

Figure 7.26 Record Expected Output

Description	This option "بدء التسجيل" allows the user to record the translated text and series of gestures and stores them as a video format.
Input	The user presses the button labelled "بدء التسجيل". Then the gestures and the translated text are recorded.

Validation Status	Complete

7.2 Validation

Validation testing is done in a stable environment taking into consideration the limitations of the application at this stage. A dataset consisting of 5 signs is injected to the system and the performance is recorded accordingly. The metrics are chosen, and the experiments are performed and explained in the following sub sections.

7.2.1 Metrics

Below are the metrics that we have chosen to validate the application:

Background

We are testing the application with different background colors, when the background is similar to the skin color there is expected to be color leakage (part of the background is considered as a hand giving false readings).

Illumination

Different lighting might affect the gesture translation.

Distance from the camera

Distance of the hand from the camera can affect the gesture translation.

Processer

Since motion detection and recognition requires high processing speed different phones are tested.

Hand in front of face

The face cannot be detected when the hand is in front of it, this is due to the features being hidden from the camera. This will cause the skin color extraction process to extract the face and hand together. This is a limitation.

One/two hands

It is important to test this as signs can be also represented with two hands.

Hand position

The same sign of the hand can be represented at different lengths. It is important to test this.

7.2.2 Validation Experiments

The metrics are tested on the basis of:

1. Sample Space: 5 ArSL words, and a total of 300 images of gestures (60 images for each word for the training data set).
2. Procedure: this will be the way the experiment is done, it varies according to each metric.
3. Percentage of Success.

Table 7.2 Metrics Experiment Summary

Metrics	Sample Space	Procedure	Percentage of Success
Background	– 3 signs – 3 seconds – 90 frames for each sign.	The experiment is repeated in 3 background colors: white, green and yellow (within skin color threshold).	– White bg: 83.33% – Green bg: 60.37% – Yellow bg: 12.59% – Total: 52.09%
Illumination	– 4 signs – 3 seconds – 90 frames for each sign.	The experiment is performed in artificial light and dim light on a standard white background.	– Light: 82.56% – Dark: 75.56% – Total: 79.44%
Distance from the camera	– 4 signs – 3 seconds – 90 frames for each sign.	The experiment is performed within the distance of 15, 30 and 45 inches of distance between the phone and the user.	– 15 inches: 64.44% – 30 inches: 17.78% – 45 inches: 11.94% – Total: 31.39%
Processor	– iPhone 6 – iPhone 7	XCode provides analysis of the performance of the mobile phone.	-
Hand in front of face	– 1 signs – 3 seconds – 90 frames for each sign.	Hand sign is placed in front of the face.	4.44%
One/two hands	– 2 signs – 3 seconds – 90 frames for each sign.	Signs are performed on a standard white background.	– One hand: 84.30% – Two hands: 78.42%
Hand Position	– 4 signs – 3 seconds – 90 frames for each sign.	Signs are performed on a standard white background. The hand is placed in middle, top, bottom, left, right of camera frame.	– Middle: 64.44% – Top: 60.55% – Bottom: 61.38% – Left:61.38% – Right:61.66% – Total: 61.882%

7.2.2.1 Data Analysis and Results

The sample space is 5 words shown in Table 7.3, varying between one hand and two hands. The data analysis had slight variations in the sample space according to the type of test performed.

Table 7.3 Sample space of gestures

Word Index	Word label	Number of hands required for gesture
1	حرف الياء	1
2	جيد	1
3	أنا	1
4	أشاهد	1
5	التلفاز	2

Background

The bar graph shown in Figure 7.27 shows the results of the background testing on three Arabic Signs. The number of frames per second is 30, and in total of 3 seconds it is 90 frames per signs. During this test the lighting, distance of the hand from the camera and the position of the hand are kept constant for all the different backgrounds. The background colors tested are white, green and yellow. Yellow was chosen to test the color leakage and its effect on the hand segmentation process. This is because yellow falls in the threshold range of the skin color and as a result it is seen from the graph that for all the signs the yellow background test showed less accuracy in comparison to the other background colors; white and green. The highest percentage success is of the white background is 83.3%, while the lowest was of the yellow background which is 12.6%. The total mean success percentage of all the background tests is 52.09% the performance of the application here is almost half accurate.

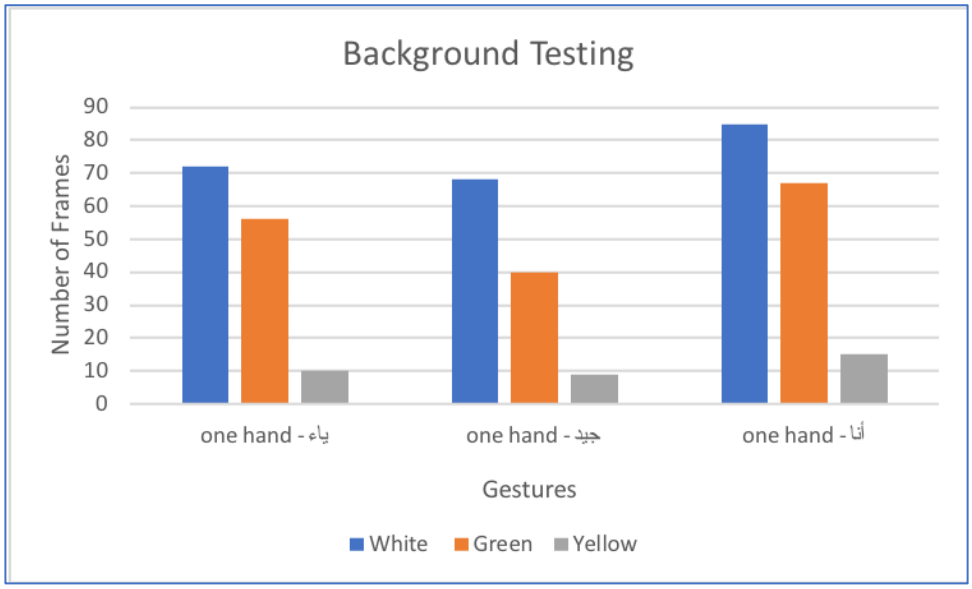


Figure 7.27 Testing Translation with different background colors

Illumination

The bar graph in Figure 7.28 below shows the results of the illumination testing on four Arabic Signs. The number of frames per second is 30, and in total of 3 seconds it is 90 per sign. The background is kept white, the position of the hand and the distance of the hand from the camera are kept constant for fairness in all cases and the illumination (light, dark) is the variant factor. The results of the change in lighting as seen in the bar graph in Figure 7.9 are almost similar. The highest recognition was for word number 3 (أنا) which is almost 85 frames correctly identified out of 90 in bright light, and in dim light 74 frames out of 90. The lowest sign to be recognized is word number 4 (أشاهد). In bright light 7 frames out of 90 are recognized and in dim light 0 frames are recognized. This is due to other factors explained in other tests related to the failure of recognizing a sign when it's in front of the face which is the constraint with word number 4, as it must be placed on the face for it to deliver its meaning. The success percentage of bright light is 82.56% and of dark light it is 75.6%, this is highly accurate.

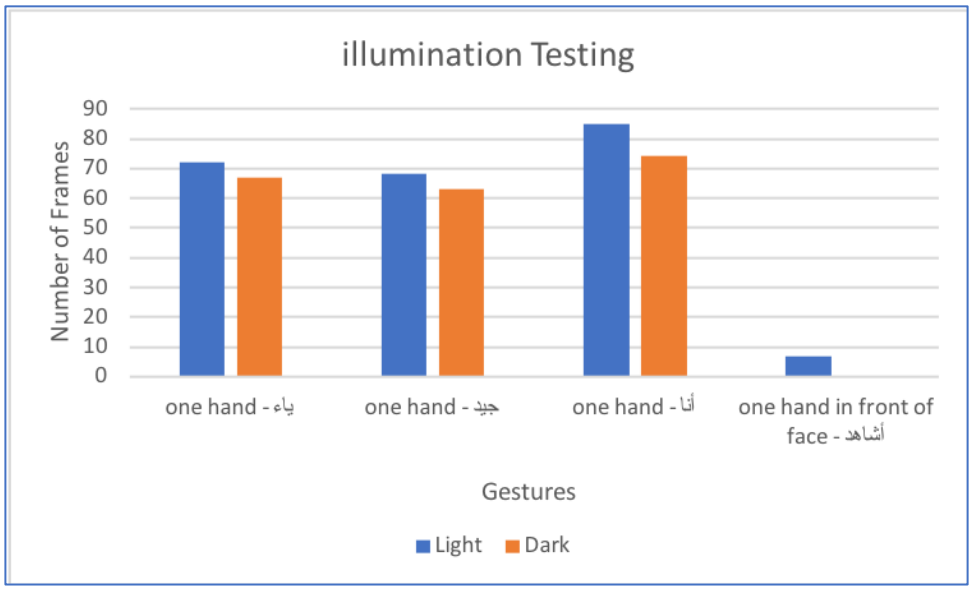


Figure 7.28 Testing Translation with different brightness

Distance of the hand from the camera

The bar graph Figure 7.29 below shows the results of the background testing on four Arabic Signs. The number of frames per second is 30, and in total of 3 seconds it is 90 frames for each gesture. During this test the lighting, background (white) and the position of the hand are kept constant for all the different distances. The distances considered for this test are 15, 30 and 45 inches. The reason of the choice of these distances was to test at which distance is the recognition going to fail. The results in Figure 7.10 are observed to be quite high for 15 inches and steeply lower for 30 inches and 45 inches for the first three signs. This is due to the recognition accuracy reducing as the distance grows larger. The last word 4 is considered an anomaly and as explained

in the previous subsection, it continues to give low accuracy values due to other reasons being the failure of segmentation when the face is involved. The percentage success for 15, 30 and 45 inches is 64.4%, 17.78% and 31.39% respectively. The mean of the total percentage of success is 31.39%, this is seen to be on the low side and could be improved.

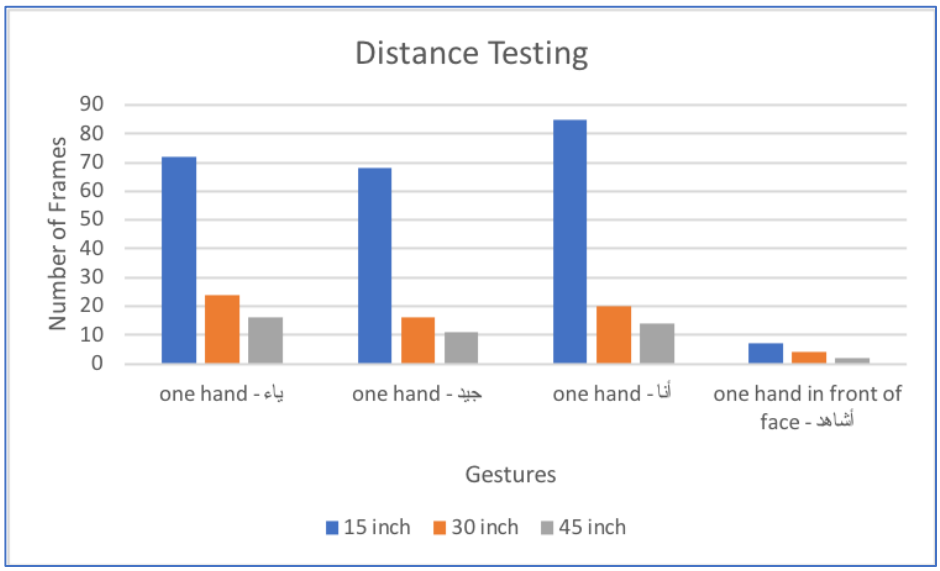


Figure 7.29 Testing Translation with different distances of hand from camera

Position

The bar graph in Figure 7.30 below shows the results of the position testing on four Arabic Signs. The number of frames per second is 30, and in total of 3 seconds it is 90 per sign. During this test the lighting, distance of the hand from the camera and the background (white) are kept constant for all the different positions. The positions are taken to be middle, top, bottom, left and right. The length from each position to the middle is taken to be 15 inches based on the previous test that was most successful for 15 inches distance. The results are very similar for the first three signs, the middle is seen to be the highest value with the percentage success of 64.4%. The following distance are almost similar ranging from 60 to 64. The fourth sign is an anomaly that could have been due to the sign being placed on the face which disturbed the test. The mean of the total successes is 61.9% which is considered average accuracy.

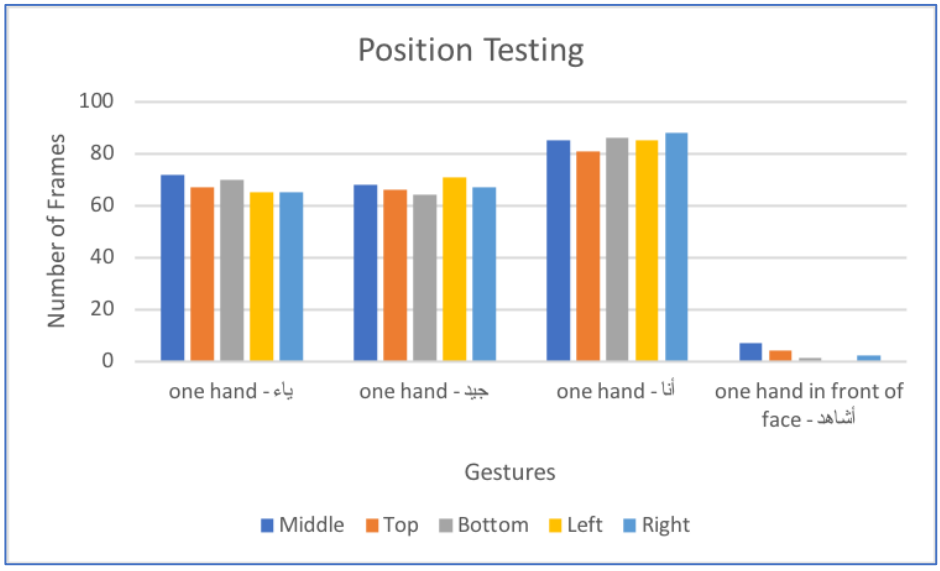


Figure 7.30 Testing Translation with different hand position on camera

Processer

The application is using extremely high CPU power with an average of 131%. Thus, causing significant battery drainage. The CPU power usage of the application should be reduced to 20% to reduce energy impact [24]. Listed below is the comparison of iPhone 6 and iPhone 7 percentages of CPU, memory and energy impact when the application translates:

iPhone 6:

The application running in iPhone 6 with A8 chip is using extremely large CPU power when translating with an average of 145%. This extreme usage is draining the battery power which in turn causes high energy impact. The use of memory when translating is very low with an average of 28.6MB (2.9%). CPU, memory and energy impact when translating in iPhone 6 can be seen in Figure 7.31.

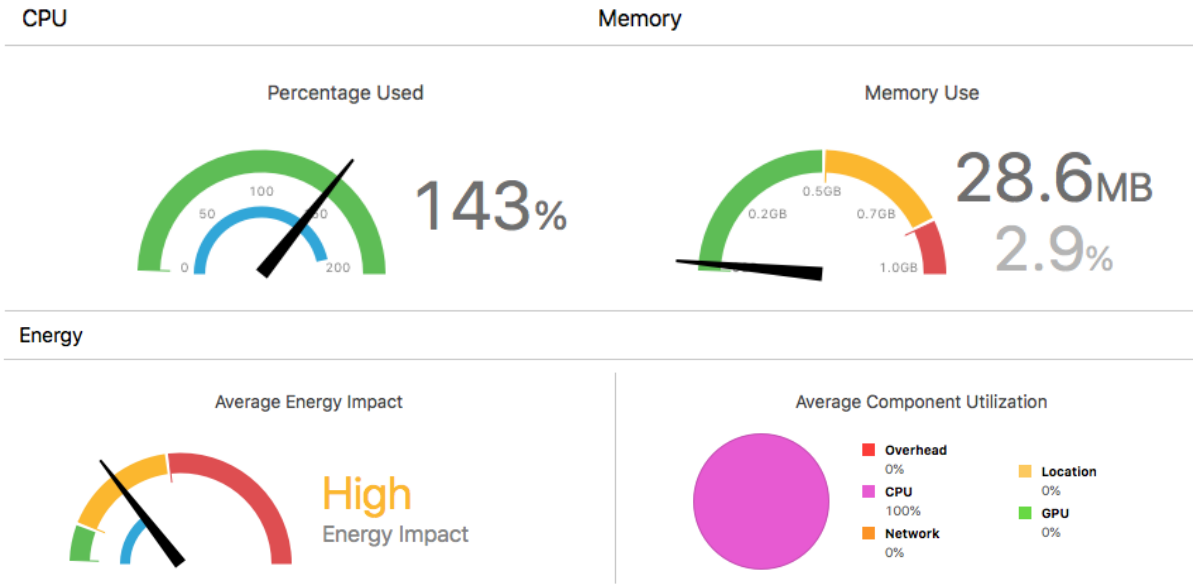


Figure 7.31 Testing Application Translation in iPhone 6

iPhone 7:

When running the application in iPhone 7 with A10 chip which according to Apple have 40% greater CPU performance than the A8 chip, in Figure 7.32 it can be seen that during translation CPU performance, memory and energy impact in iPhone 7 is slightly different than when running the application in iPhone 6. When the application translates, iPhone 7 is using an average CPU power of 120% with high average energy impact and low memory usage of 49.2MB (2.5%).

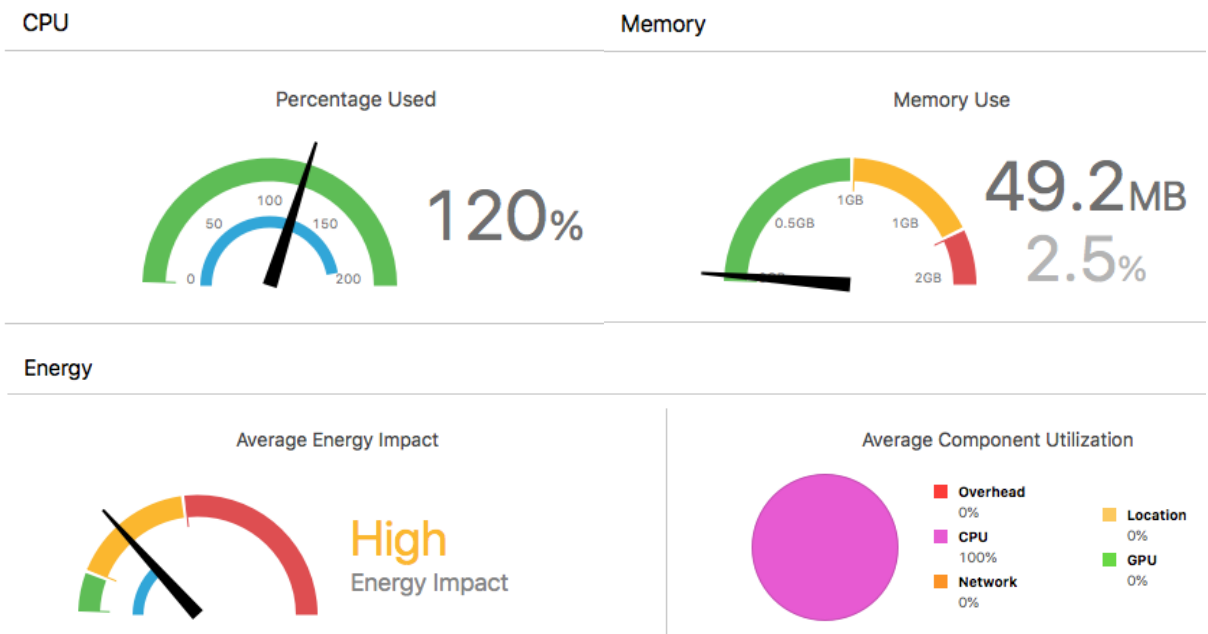


Figure 7.32 Testing Application Translation in iPhone 7

Hand in front of face

Only one sign was tested here which requires to be placed in front of the face. The percentage success was 4.44% which is considered very low.

One/two hands

The result of testing two hands is positive. The application can detect two hands with a percentage success of 78.42%.

7.2.3 Validity

During the testing process there are some factors that are considered threats to the results obtained. Firstly, the limitation of the iPhones available to test the performance on a larger scale of phones. This affected the accuracy of the performance test as it didn't cover a larger sample space. Another threat to validity is human error when varying the position or the distance from

mobile phone. This requires a larger sample space to smooth out any errors that might affect the results of the testing.

7.2.4 Summary of Findings

In this section the summary of the results is discussed. In addition to that, project review and key skills developed are mentioned briefly.

7.2.4.1 Discussion

Based on reading and researching the limitations of other Sign Language Translators we have had some expectations regarding the results to be obtained from the testing. In the background test, it was expected that the white background and the green background will both have a success percentage higher than that of the yellow as the yellow values fall in the skin threshold. So, the yellow background could be mistaken to be part of the hand, therefore it would not be segmented, and this leads to false readings. Which was the case exactly as seen in the previous subsections. With illumination it was expected that the application would work better in bright light and it was true, however the dim light success percentage was also quite close to the bright light condition.

The lowest distance from the mobile phone to the hand object was seen successfully to be 15 inches and the position was seen to be most successful when it's in the middle.

The application succeeded to recognize two hands. However, the application showed very low performance regarding the hand in front of the face, this is because the face area is added to the hand area creating an error in the matching.

iPhone 7 performed better than iPhone 6 as it uses an average CPU power of 120% with high average energy impact and low memory usage of 49.2MB (2.5%) this is due to more powerful hardware utilities using an A10 chip with stronger processors.

It is important to mention that since there are no other ArSL translator application for iPhone there is no margin to compare against as per performance or accuracy so the comparison was according to the percentage of success of each task and the overall performance compared to Apple's standards for applications in general.

7.2.4.2 Project Review and Key Skills

During the time of the project, the biggest challenge was choosing the best algorithm for gesture recognition. Many algorithms were used then changed and as a result time was wasted to find the most efficient one. An improvement here would have been to have more research time to avoid wasting time writing different algorithms then gradually using others.

The greatest skill that was obtained was using *OpenCV* libraries with *Swift* to handle computer vision in an iOS application. We are aiming to create a similar application for Android, so the process will be quicker due to the knowledge developed in using *OpenCV's* libraries effectively.

7.3 Conclusion

In this section, the application was verified and validated. In addition to that the metrics were set and tested. The results were obtained and discussed, and the limitations were concluded to be that the application has a very low performance when detected signs are on the face and when the background is of similar color as the skin tone.

Chapter 8: User Manual

The user manual for ArSL Translator Application is integrated into the application and it is shown upon the first time use of the application. Below are screenshots of the user manual:

This is the first screen of the user manual. It explains that the first button is the button that starts the translation process or stops it.

Figure 8.1 User Manual Screenshot (1)

This is the second screen of the user manual. It explains that the second button is the button that stores the translated text into a text file.

Figure 8.2 User Manual Screenshot 2

Figure 8.3 User Manual Screenshot (3)

This is the third screen of the user manual. It explains that the third button is responsible for converting audio (Arabic spoken language) to text to support two-way communication.

Figure 8.4 User Manual Screenshot (4)

This is the fourth screen of the user manual. It explains that the fourth button is the button that stores the gestures and their corresponding translation in a video document.

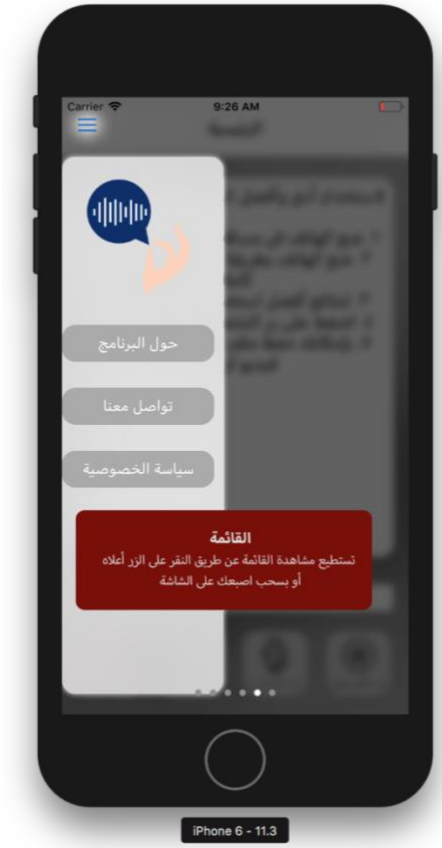


Figure 8.5 User Manual Screenshot (5)

This is the fifth screen of the user manual. It explains the way to get to the main menu.

Figure 8.6 User Manual Screenshot (6)

This is the sixth screen of the user manual. It allows the user to start using the application.

Chapter 9: Conclusion

The Arabic Sign Language Translator application is developed to provide an efficient, maintainable and portable solution to the deaf society to help them eliminate the communication difficulties when dealing with members of the society who do not speak the Arabic Sign Language.
In this final report, all the requirements have been discussed, motivations behind the project and goals. There was a detailed description of the database, project design and interfaces.

9.1 Limitations

There are several limitations on the application these include: it's inability to segment the hand from the background if the background is of the same color as the skin tone. The application is also unable to recognize gestures that are placed in front of the face. In addition to that, the application consumes lots of CPU power and performs better when it runs in an iPhone 7+ versions due to the Apple's hardware upgrades.

9.2 Future Work

In the near future, we plan to upload the application in the App Store and create another version for Android and upload in the Play Store. We plan to increase the signs in the dataset for the application to include more signs. In addition to that, search for ways to increase the accuracy and performance of the application, specifically the performance of the front camera. Finally, we plan to add a variety of languages as well to the application, so it doesn't just contain ArSL.

9.3 Summary

The Arabic Sign Language Translator application is aimed at iOS users, using *Swift* and *OpenCV* framework using C++ as the programming language. This report explained the phases of developing the application including the work plan, requirements, dataset, used functions and motion capture.

REFERENCES

[1] OpenCVTeam, "About OpenCV Library," 11 October 2017. [Online]. Available: https://opencv.org/about.html.

[2] M. A. Abdel-Fattah, "Arabic Sign Language: A Perspective," *The Journal of Deaf Studies and Deaf Education*, vol. 10, no. 2, pp. 212-221, 1 March 2005.

[3] M. A. Kamel, قاموس لغة الإشارة, Cairo: Altalae, 2004.

[4] C. Manresa-Yee, J. Varona, R. Mas and F. J. Perales, "Hand Tracking and Gesture Recognition for Human-Computer Interaction," *Electronic Letters on Computer Vision and Image Analysis*, 2000.

[5] M. S. Abdalla and E. E. Hemayed, "Dynamic Hand Gesture Recognition of Arabic Sign Language using Hand Motion Trajectory Features," *Global Journal of Computer Science and Technology Graphics & Vision*, vol. 13, no. 5, pp. 27 - 33, 2013.

[6] T. Suksil and T. H. Chalidabhongse, "Hand Detection and Feature Extraction for Static Thai Sign Language Recognition," in *Proceedings of the 7th International Conference on Ubiquitous Information Management and Communication*, Kota Kinabalu, Malaysia, 2013.

[7] H.-S. Y.-G. L. Lim, "Hand tracking and gesture recognition system for human-computer interaction using low-cost hardware," *Multimedia Tools and Applications*, vol. 74, no. 8, pp. 2687-2715, 1 April 2015.

[8] "Convert from YCbCr to RGB Color Space - MATLAB & Simulink," MATLAB & Simulink, [Online]. Available: https://www.mathworks.com/help/images/convert-from-ycbcr-to-rgb-color-space.html. [Accessed 18 March 2018].

[9] C. M. Jin, Z. Omar and M. H. Jaward, "A mobile application of American sign language translation via image processing algorithms," in *Region 10 Symposium (TENSYMP), 2016 IEEE*, Bali, Indonesia, 2016.

[10] R. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Journal of Bais Enginnering*, vol. 82, pp. 35 - 45, 1 March 1960.

[11] G. B. Greg Welch, "An Introduction to the Kalman Filter," Chapel Hill, 2006.

[12] "Introduction to SURF (Speeded-Up Robust Features)," OpenCV, 10 Novemeber 2014. [Online]. Available: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html. [Accessed 20 April 2018].

[13] "The Hessian," Khan Academy, 16 June 2016. [Online]. Available: https://www.khanacademy.org/math/multivariable-calculus/applications-of-multivariable-derivatives/quadratic-approximations/a/the-hessian. [Accessed 15 April 2018].

[14] M. Alkhawlani, M. Elmogy and H. Elbakry, "Content-Based Image Retrieval using Local Features Descriptors and Bag-of-Visual Words," *International Journal of Advanced Computer Science and Applications(IJACSA),* vol. 6, no. 9, pp. 212 - 219, 2015.

[15] D. S. P. N. J. Ashwin S.Pol, "Sign Language Recognition Using Scale Invariant Feature Transform and SVM," *International Journal of Scientific & Engineering Research,* vol. 4, no. 6, pp. 1683 - 1686, June 2013.

[16] J. A. H. a. M. A. Wong, "Algorithm AS 136: A K-Means Clustering Algorithm," *Journal of the Royal Statistical Society. Series C (Applied Statistics),* vol. 28, pp. 100-108, 1979.

[17] A. K. Gary Bradski, Learning OpenCV: Computer Vision with the OpenCV Library, O'Reilly Media, 2008.

[18] "OpenCV: Face Detection using Haar Cascades," OpenCV, 13 February 2018. [Online]. Available: https://docs.opencv.org/3.4.1/d7/d8b/tutorial_py_face_detection.html. [Accessed 28 April 2018].

[19] K. N. D. Chai, "Face segmentation using skin-color map in videophone applications," *IEEE Transactions on Circuits and Systems for Video Technology ,* vol. 9, no. 4, pp. 551 - 564, June 1999.

[20] "KinTrans," 23 March 2016. [Online]. Available: http://www.kintrans.com/#product. [Accessed 18 January 2018].

[21] "MotionSavvy," 27 July 2007. [Online]. Available: http://www.motionsavvy.com/. [Accessed 18 January 2018].

[22] "Creatly.com," Cinergix Pty. Ltd., 13 August 2013. [Online]. Available: https://creately.com/. [Accessed 6 March 2018].

[23] D. R. Hipp, "Sqlite.org," 17 August 2000. [Online]. Available: https://www.sqlite.org/index.html.

[24] "Energy Efficiency Guide for iOS Apps: Fundamental Concepts," Apple, 13 September 2016. [Online]. Available: https://developer.apple.com/library/content/documentation/Performance/Conceptual/EnergyGuide-iOS/FundamentalConcepts.html. [Accessed 30 April 2018].

[25] P. G. V. K. B. S. J. M. M. J. Khamar Basha Shaik, "Comparative Study of Skin Color Detection and Segmentation in HSV and YCbCr Color Space," *Procedia Computer Science,* vol. 57, pp. 41 - 48, 21 August 2015.

[26] R. C. &. R. Winters, "How It Works: Xbox Kinect," Jameco Electronics, 7 February 2012. [Online]. Available: https://www.jameco.com/jameco/workshop/howitworks/xboxkinect.html. [Accessed 29 January 2018].

[27] M. J. J. Paul Viola, "Robust Real-Time Face Detection," *International Journal of Computer Vision,* vol. 57, no. 2, pp. 137-154, 1 May 2004.

[28] W. McIlhagga, "The Canny Edge Detector Revisited," *International Journal of Computer Vision,* vol. 91, no. 3, pp. 251 - 261, 1 February 2011.

APPENDIX I (SURVEY)

دراسة حول صعوبات التواصل بلغة الإشارة و تطوير نظام ذكي لحل الصعوبات.

هذا الإستبيان وضع من قبل مجموعة من طالبات قسم علوم الحاسب الآلي في جامعة السلطان قابوس لأجل مشروع التخرج.
يهدف هذا الإستبيان إلى معرفة الصعوبات الي قد تواجهونها خلال الأعمال والتعاملات اليومية، حيث أننا نعمل على استخدام هذه المعلومات لتطوير حل مناسب لمواجهة تلك الصعوبات.

لمشاركاتكم و اقتراحاتكم يرجى التواصل معنا عبر بريدنا الإلكتروني : fypwamm@gmail.com

العمر

- أقل من 18 سنة
- من 18 إلى 30 سنة
- أكبر من 30 سنة

المهنة

- طالب
- موظف
- غير موظف

* حدد مستوى الصعوبة عند تعاملك مع من لايُجيدون لغة الإشارة؟ (1 = سهل جدًا، 5 = صعب جدًا)

	1	2	3	4	5	
سهل جدا	○	○	○	○	○	صعب جدا

* نظام الهاتف الذي تستخدمه

- iOS (آيفون)
- Android (سامسونج، هواوي، سوني، ...)
- اخرى

* ما هي لغة الإشارة التي تتقنها؟

- [ ] الإشارة العربية الموحدة
- [ ] العامية

* عند استخدامك للغة الإشارة ما اليد التي تستخدمها باستمرار؟

- ( ) اليد اليمنى
- ( ) اليد اليسرى
- ( ) كلاهما

* هل تعتقد أنك بحاجة إلى جهاز ناطق ؟

- ( ) نعم
- ( ) لا

* هل لديك مترجم خاص للمعاملات اليومية مع الأشخاص الذين لا يجيدون لغة الإشارة ؟

- ( ) نعم
- ( ) لا

* هل تستخدم تطبيق هاتف للتواصل مع من لا يجيدون لغة الإشارة في الحياة اليومية ؟

- ( ) نعم
- ( ) لا

إذا أجبت بنعم، اذكر اسم التطبيق الذي تستخدمه

Short answer text

* لو تم تصميم برنامج يهدف إلى مساعدتك في التواصل مع الآخرين هل ستبادر باستخدامه؟

- ( ) نعم
- ( ) لا
- ( ) ربما

ماهي المميزات التي تفضل تواجدها في تطبيق ترجمة لغة الإشارة للصوت؟

- [ ] تسجيل مقطع فيديو
- [ ] تسجيل مقطع صوتي
- [ ] حفظ الترجمة المقالية
- [ ] إمكانية إضافة مفردات للغة الاشارة إذا لم تكن متوفرة
- [ ] Other...

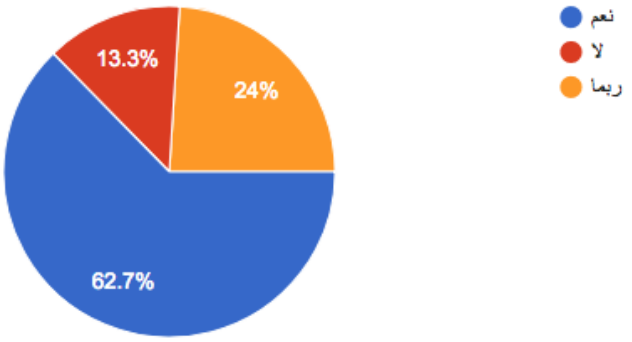
الرجاء كتابة اقتراحاتكم لتطبيق ترجمة لغة الإشارة للصوت

Long answer text

APPENDIX II(SURVEY RESULTS)

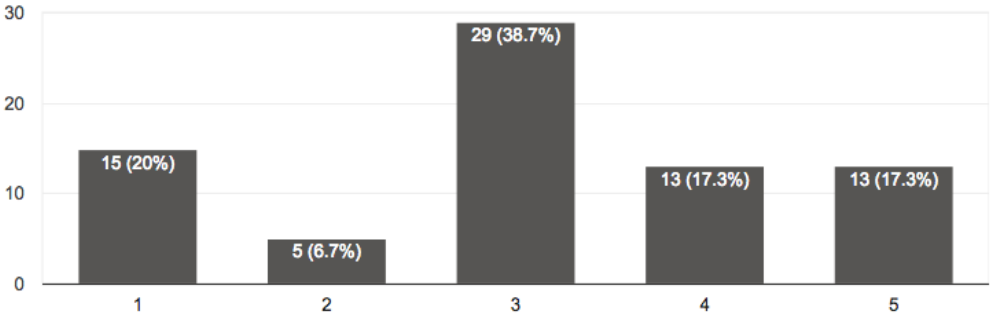
لو تم تصميم برنامج يهدف إلى مساعدتك في التواصل مع الآخرين هل ستبادر باستخدامه؟

75 responses



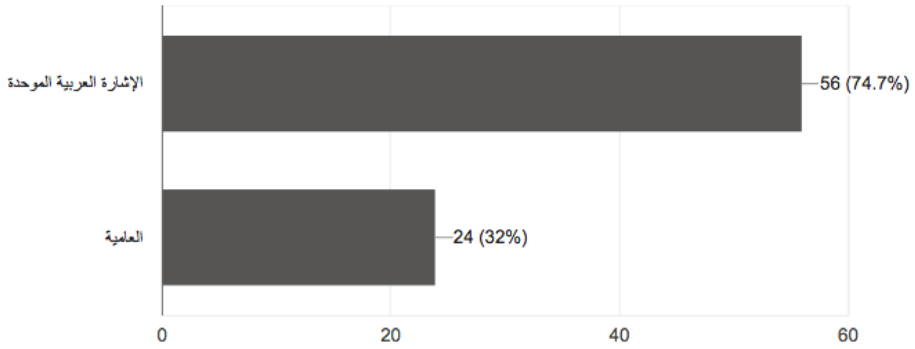
(حدد مستوى الصعوبة عند تعاملك مع من لايُجيدون لغة الإشارة؟ (1 = سهل جدًا، 5 = صعب جدًا

75 responses



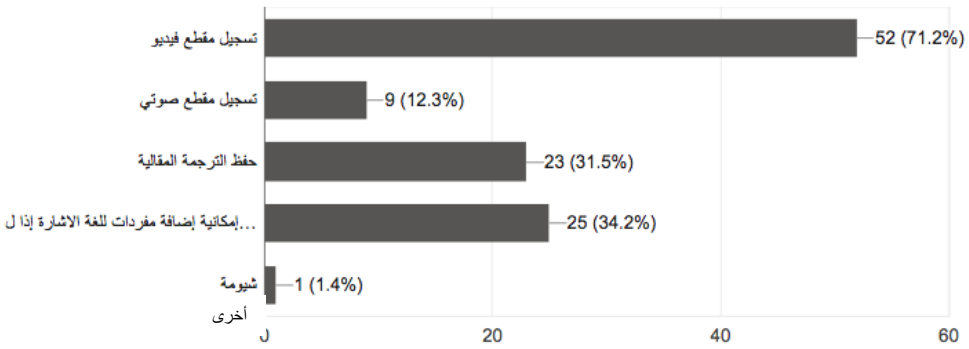
ما هي لغة الإشارة التي تتقنها؟

75 responses



ماهي المميزات التي تفضل تواجدها في تطبيق ترجمة لغة الإشارة للصوت؟

73 responses



APPENDIX III (SPEECH TO TEXT IMPLEMENTATION)

```
func startSpeechToText() {

    if recognitionTask != nil {
        recognitionTask?.cancel()
        recognitionTask = nil
    }

    let audioSession = AVAudioSession.sharedInstance()
    do {
        try audioSession.setCategory(AVAudioSessionCategoryRecord)
        try audioSession.setMode(AVAudioSessionModeMeasurement)
        try audioSession.setActive(true, with: .notifyOthersOnDeactivation)
    } catch {
        print("audioSession properties weren't set because of an error.")
    }

    recognitionRequest = SFSpeechAudioBufferRecognitionRequest()

    let inputNode = audioEngine.inputNode;

    guard let recognitionRequest = recognitionRequest else {
        fatalError("Unable to create an SFSpeechAudioBufferRecognitionRequest object")
    }

    recognitionRequest.shouldReportPartialResults = true

    recognitionTask = speechRecognizer?.recognitionTask(with: recognitionRequest, resultHandler: { (result, error) in

        var isFinal = false

        if result != nil {

            self.speechTextView.text = result?.bestTranscription.formattedString
            isFinal = (result?.isFinal)!
        }

        if error != nil || isFinal {
            self.audioEngine.stop()
            inputNode.removeTap(onBus: 0)

            self.recognitionRequest = nil
            self.recognitionTask = nil

            self.microphoneButton.isEnabled = true
        }
    })

    let recordingFormat = inputNode.outputFormat(forBus: 0)
    inputNode.installTap(onBus: 0, bufferSize: 1024, format: recordingFormat) { (buffer, when) in
        self.recognitionRequest?.append(buffer)
    }

    audioEngine.prepare()

    do {
        try audioEngine.start()
    } catch {
        print("audioEngine couldn't start because of an error.")
    }

    speechTextView.text = "ابدأ التحدث"

}
```