Table of Content

Abstract……..……..……..……..……..……..……..1

Chapter 1 Introduction

1.1 Motivation ……...2

1.2 Background…….....3

1.2.1 Speech Disorder……....…….....3

1.2.1.1 Dysprosody…..……..……..…….3

1.2.1.2 Voice Disorder..…….……..4

1.2.2 Arrhythmia...…….....4

1.2.3 Chronic Stress…….6

1.3 Objectives ……6

1.4 Development Tools……..7

Chapter 2 Theories & Algorithms

2.1 Speech Disorder Diagnosis ……8

2.1.1 Digitalization ……...……..….8

2.1.2 Pitch..……..……...……..8

2.1.3 Fast Fourier Transform ……...……..……..……..……9

2.1.4 Harmonic Product Spectrum ……...……..……..……9

2.1.5 Cepstrum analysis ……...……..……..……....……...10

2.2 Heart Rate Measurement ……...……..……..……....…….11

2.2.1 Colour Space……...……..……...……....……..11

2.2.2 YUV & YCbCr formats..……..…..…..……...……....……...12

2.3 Emotion Recognition...……..……..…..…..……..……....…….13

2.3.1 Previous Research Effort.s…..…..……...……..……....……13

2.3.2 EigenFace Model..…….……...……..……....……13

Chapter 3 Design Procedure

3.1 Speech Disorder Checker..…….……...……..……...…15

3.1.1 Record, Sample and Digitalization..…….……...……..……15

3.1.2 Pitch Calculation and Display...…….……...……..……...…….16

3.1.3 Feature Extraction.…….....……..……...…….19

3.1.4 Alerts for Abnormal Feature…….....……..……...……19

3.2 Heart Rate Monitor…….....……..……...…….20

3.2.1 Video Record.…….....……..…….....…...20

3.2.2 Average Red Intensity Calculation & 3.2.3 Heartbeat Counting……21
3.2.4 Heart Rate deduction……...22
3.3 Emotion Tracker……...23
3.3.1 Facial Image Capture…….……...24
3.3.1.1 Camera Configuration…….……...25
3.3.1.2 Face Detection…….……...26
3.3.2 Feed Test Image to EigenFace Model……...29
3.3.3 Expression Classification and result recording……34
3.3.4 Long term monitoring……..……35
Chapter 4 Testing Results
4.1 Speech Disorder Checker…….……38
4.2 Heart Rate Monitor……...39
4.3 Emotion Tracker……...40
Chapter 5 Conclusion & Future Works
5.1 Speech Disorder Checker……..43
5.2 Heart Rate Monitor……43
5.3 Emotion Tracker…….44
References…….45

List of Figures

Figure 1. Data flow diagram for Cooley–Tukey FFT algorithm (N=8)
Figure 2. Harmonic Product Spectrum implementation
Figure 3. Cepstrum analysis implementation
Figure 4. Illustration of YUV colour format
Figure 5. Colour format conversion between RGB and YUV
Figure 6. Colour format conversion between RGB and YCbCr
Figure 7. Speech Disorder Checker working mechanism flowchart
Figure 8. Code illustration for Audio recording
Figure 9. Code illustration for audio data processing in thread
Figure 10. Code illustration of screen display update

Figure 11. Code illustration of Harmonic Product Spectrum method
Figure 11. Code illustration of Harmonic Product Spectrum method
Figure 12. Code illustration of Cepstrum analysis
Figure 13. Heart Rate Monitor working mechanism flowchart
Figure 14. Code illustration of colour format conversion between YCbCr and RGB
Figure 15. Code illustration of average red intensity calculation and heartbeat counting
Figure 16. Code illustration of heart rate deduction
Figure 17. Emotion Tracker working mechanism
Figure 18. Code illustration for opening camera and displaying preview on screen (side the camera preview class)
Figure 19. List of automatics face detection results using Haar cascade classifiers
Figure 20. Screen display for facial image capture
Figure 21. EigenFace model working mechanism flowchart
Figure 22. Samples of Japanese Female Facial Expression Database
Figure 23. Text file for training eigenface model
Figure 24. Code illustration for eigenface model construction
Figure 25. Code illustration for Principal Component Analysis on images
Figure 26. Eigenfaces for every training image
Figure 27. Average Eigenface developed from training images
Figure 28. Welcome page of the DoctorZ application
Figure 29. Introduction page of the DoctorZ application
Figure 30. User interface of Speech Disorder Checker
Figure 31. Cepstrum examples during development
Figure 32. Pitch contour plotted without regression
Figure 33. User interface of Heart Rate Monitor
Figure 34. User interface of Emotion Tracker
Figure 35. Expression classification examples

List of Tables

Table 1. Common voice disorders and corresponding symptoms

Abstract

Health awareness of people is generally improved in the past decade. On one hand, with the accelerating pace of modern life, non-intrusive intelligent health monitoring becomes desirable. On the other hand, the Android smart phones equipped with various sensors and powerful processing unit are increasingly popular in the market. In this project, I programmed an Android App capable of issuing health alerts based on audio and visual processing. Raw data was obtained from the built-in camera and the microphone. Signal processing and machine learning algorithms were applied to give intelligent feedback. The first part of the App is to calculate the user's speech pitch at run time and to check for speech disorders. The second function is to measure the user's heart rate using real time fingertip image processing. The last feature is to classify the user's emotion status from the captured facial image and to record the result on database for mental condition monitoring. Health monitoring on mobile devices is intrinsically challenging due to its complex nature and hardware limitation. Satisfactory recognition accuracy has been achieved in the development stage. Higher accuracy is expected after the App is released as beta version and tested with larger training dataset.

Chapter 1 Introduction

1.1 Motivation

The past decade witnessed the general improvement of the public awareness in healthcare. In the meantime, the work and study stress increases as the modern life pace accelerates. People always find it too late to get medical care. The reason for missing the best timing to take appropriate precautions lies in the limited understanding of personal health condition, resulted from inadequate exposure to health monitoring systems. However, visiting hospitals is not desirable under many circumstances. First of all, time is limited when there is a tight schedule. Secondly, visiting hospital is costly. High fee is required both for doctor inspection and medical equipment access. Moreover, traditional clinic oriented diagnosis is less user friendly. Depends on what you want to check, the whole diagnosis process may be rather complex and may involve quite a few medical equipment whose operation needs professional technicians. Finally, social needs far outweighs the supply of health care services. Hospital reservation is difficult to get.

On one hand, medical research works on smart health monitoring has been promising in recent years. Through persistent monitoring using sensor systems complemented with diagnostic software, personal health monitoring systems offers individuals independence and better timing of symptom detection. The intelligent health monitoring systems developed can also ease the burden on the government's healthcare system. Further, the diagnosis accuracy and reliability would be enhanced by the combination of the objective monitoring from sensory data and subjective ratings from medical professionals and physicians. For instance, research study predicts that around 25% - 50% of patients suffering from chronic disease and/or mental illness (such as depression) will benefit from persistent personal health monitoring solutions [1].

On the other hand, our days see the overwhelming influence of smartphones. The blooming of mobile applications make the capabilities of mobile phones unreachable, from online shopping to concert booking, from portable games to examination preparations. The wide range of mobile applications make phones as smart as we call them nowadays. Smartphones

are becoming an indispensable part of modern life. With the various sensors and powerful processing unit embedded (for instance, the Samsung Galaxy S4 model has 1.9 GHz Quad-Core Processor, 2GB RAM, a 13 Mega pixel main camera and 2 mega pixel front camera), it is achievable for the smartphones to take up more responsibilities and among which, the intelligent health monitoring is of high emergence. The fast Internet access of smartphones and the development of cloud technology also make it convenient to automatically backup user's monitoring records in the cloud and to make it accessible wherever desired.

Inspired by the potentials of smartphones and the needs laid upon less intrusive intelligent health monitoring systems, I was motivated to programme an Android application with health monitoring abilities. The application was mainly developed based on audio and visual processing algorithms.

1.2 Background

1.2.1 Speech Disorder

Speech disorder is the primary type of symptoms that can be monitored by audio processing techniques. Generally defined, speech disorder refers to a type of communication disorder in which 'normal' speech is not continuous [2]. This may take forms as stuttering, lisps, etc. Literature survey on speech disorder was conducted with all disorder types categorized and summarized with corresponding symptoms and common diagnostic practices. After careful comparison and risk assessment, the project focus was narrowed down to the analysis of dysprosody and voice disorders.

1.2.1.1 Dysprosody

Dysprosody, otherwise defined as pseudo-foreign syndrome, refers to a disorder where prosodic functions are either comprised or disabled [3]. Prosody is the supra-segmental property of the speech signal that regulates and improves its meanings [5]. Prosodic functions refer to the abilities to generate variations in pauses, stresses, intonation, melody, vocal quality, intensity, and accents of speech correctly. Dysprosody is a neurological speech

disorder. The cause is usually related to neurological pathologies such as cranioencephalic traumatisms, brain vascular accidents, and brain tumours [4].

At the grammatical or otherwise called linguistic level, dysprosody is interpreted as the comprised ability to pronounce stress and intonation accurately. A face-to-face communication with professional clinicians and their subjective ratings make the traditional diagnosis approach of dysprosody. For example, with an attempt to identify linguistic dysprosody, a patient is requested to read sentences that can either be a statement or a question using both declarative and interrogative intonations. The way how the patient uses prosodic contours to tell between asking a question and saying a statement is recorded [6] and used to diagnose if dysprosody may be developed.

Research efforts conducted on the automatics characterization of prosodic skills have seen promising accuracy. However, most of the processing systems developed for automatic labelling are computer based. Mobile device based audio processing applications are rare due to its limited computation resources and limited sensory accuracy. Still, it is possible to convert the complex processing technique to run on mobile devices. How to guarantee certain degree of accuracy, to choose the most appropriate signal processing algorithm is the key to effective implementation.

1.2.1.2 Voice Disorder

Voice disorder is another type of common speech disorder. As the two terms are often used interchangeably, in this article, voice disorder refers to symptoms related to physical impairments to the larynx or vocal resonance. After studying some medical papers concerning voice disorder classification. The table below summarized common types of voice disorder and their corresponding symptoms [7].

Voice Disorder	Symptom
Vocal fold nodule	vocal fatigue; abnormal voice quality; discomfort after extensive voice use; limited pitch and volume
Vocal fold cyst	comprised quality of voice, hoarseness or a breathy sound; the vocal cords generate multiple tones at the same time;
Bogart–Bacall	an unnaturally deep or rough voice, or dysphonia, and

syndrome	vocal fatigue
Vocal Hemorrhage	loss of pitch range; sudden decrease in voice quality; loss of vocal control; loss of volume
Muscle Tension Dysphonia	rough, hoarse, gravely, raspy, coarse; changing pitch; excessively high or low pitch; inability to produce a loud and clear voice
Vocal fold (Vocal Cord) Bowing	undependable voice, fatigue from voice use, voice gets weaker with continued talking, sense of effort when talking, poor volume

Table 1. Common voice disorders and corresponding symptoms

The traditional way to diagnose voice disorders involves practitioner inspection and medical instrument based tests [8]. Among the many voice disorders, vocal range related Vocal fold Nodule, Vocal Haemorrhage, Bogart–Bacall syndrome and Muscle Tension Dysphonia will be mainly discussed in this project. According to medical research, the voiced speech of a typical adult male will have a fundamental frequency from 85 to 180 Hz, and that of a typical adult female from 165 to 255 Hz [9, 10].

1.2.2 Arrhythmia (Irregular Heartbeat)

Heart rate means the number of heartbeats per minute. It is among the most vital human health indexes. Arrhythmia refers to a group of symptoms where the electrical activity of the heart is irregular, or is beyond the normal range of 60-100 bpm [11]. Although many arrhythmias are not life-threatening, some can cause cardiac arrest [11], the cessation of functional blood circulation caused by heart failure of effective contraction [12].

Over the years, heart rate measurement usually takes two forms: auscultation of the heartbeat with a stethoscope and peripheral pulse feeling. These usually cannot diagnose specific arrhythmia but can give a general indication of the heart rate and its regularity.

Driven by market needs, non-intrusive heart rate monitor has been developed in recent years. A heart rate monitor is a device which calculates the beats per minute based on a sample of heartbeats and to use that the information to track heart condition. Devices with user-friendly interface, low complexity to operate, and real-time information feedback are desired because

general public have little medical background and time is limited in certain emergent cases. Many products available in the market have drawbacks mentioned above and are limited in portability and flexibility [13].

1.2.3 Chronic stress

Chronic stress refers to the prolonged emotional pressure. A high level of stress hormones is created after long-term stressful emotion status, which may lead to inhibition of growth, suppression to muscle tissue, high blood pressure (and subsequently heart disease), damage to the immune system, and danger to mental health. [14]. Chronic stress develops before we notice it. In order to get necessary medical care before the conditions are non-invertible, monitoring beforehand is good practice.

As mentioned before, persistent personal health monitoring solutions benefits patients with chronic disease and/or mental illness (such as depression) [1]. Henceforth, it is desirable to integrate affective computing in the smart health monitoring system. Among the most active research areas on human behaviour studies, affective computing exerts significant impacts upon medical informatics and healthcare society. It studies the automatic recognition, interpretation and processing of human emotions based on collected sensory data. Medical survey indicates that positive emotions and negative emotions can have opposing and yet tremendous impacts on individuals. Prolonged negative emotion status jeopardizes personal health not only psychologically but also physiologically. Moreover, each subject's emotional states can also indicate his/her physical and mental health level. All these evidences suggest that accurate and robust human emotion recognition is crucial in the medical community.

1.3 Objectives

This project aimed to develop an Android application with the ability to record, process and display the user's talking pitch during his/her speech. After the user finishes talking, a pitch related checking report would be generated, including a pitch contour (for linguistic dysprosody diagnosis) and pitch range analysis (for certain voice disorders diagnosis). The application was also expected to measure the user's heart rate using the built-in camera on Android smartphones. Upon arrhythmia detected, alerts would be issued on the application's user interface. Finally, the application was designed with an attempt to detect the user's

emotion status by classifying his/her facial expressions from a simple selfie picture. The daily emotion status information will therefore be recorded to the application's database for long-term tracking. On a monthly basis, a chronic stress evaluation report would be generated for the user's review purpose.

1.4 Development Tools

For the Android application development, the programming language used is Java. The integrated development environment used is Eclipse IDE with Android Software Development Kit (SDK), which comes with API libraries and developer tools important to build, test, and debug Android apps. Screen shots of code will be included if necessary. Full set of project code is accessible at http://jodiezhao.local/~dingzhao/DoctorZ

Chapter 2 Theories & Algorithms

2.1 Speech Disorder Diagnosis

2.1.1 Digitalization

Analogue signals are continuous variable in both the number of samples within a given time period and the number of possible value levels of the signal at a particular time. In opposite, digital signals are discrete in both of these regards. Human voice is analogue signal by nature but only digital data can be processed by computer systems. Digitalization should be performed after raw data obtained.

Digitalization consists of two steps.

Sampling:

Sampling is the reduction of a continuous signal to a discrete signal. According to Nyquist theorem, a continuous band-limited signal can be uniquely determined by its samples taken at a frequency twice as large as its bandwidth.

Quantization:

Quantization is the process of converting a large set of input values to a countable set, usually takes form as rounding values to some unit of precision.

Among many digitalization schemes, Pulse-code modulation (PCM), the standard form of digital audio in computers, is adopted in this project. Bit depth of PCM refers to the number of possible digital values that each sample can take.

2.1.2 Pitch

Pitch, also interchangeably referred as fundamental frequency, is a perceptual property that allows the ordering of sounds on a frequency-related scale [15]. To get the frequency information of the user's speech, the Fourier transform (discussed in section 2.1.3) is necessary to accomplish the time domain to frequency domain shift. To ensure pitch detection accuracy, two pitch detection algorithms: Harmonic Product Spectrum (see section 2.1.4) and Cepstrum Analysis (see section 2.1.5) are proposed.

2.1.3 Fast Fourier Transform

The Fourier transform (FT) is an operation that converts a time domain based signal into its frequency domain. For finite-length discrete signal, the Fourier transform is defined as a discrete Fourier transform (DFT).

Discrete Fourier Transform Representation of a Finite-Length Signal:

$$X_k \stackrel{\text{def}}{=} \sum_{n=0}^{N-1} x_n \cdot e^{-i2\pi kn/N}, \quad k \in \mathbb{Z}$$

Evaluating DFT according to the definition requires $O(N^2)$ operations [16].

A fast Fourier transform (FFT) is an algorithm to compute the discrete Fourier transform (DFT) in O(N log N) running time. The FFT does the same job by grouping the even number indices and odd number indices separately and performing multiplications & summation iteratively.

Figure 1 below illustrates the Cooley–Tukey FFT algorithm's data flow diagram for N=8.

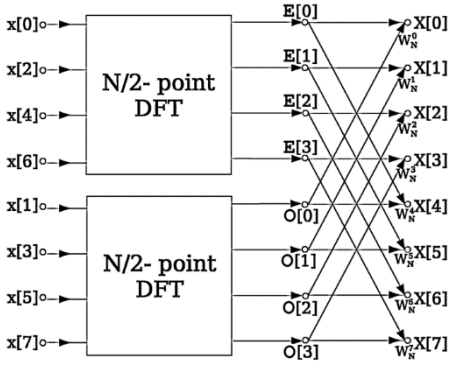


Figure 1. Data flow diagram for Cooley–Tukey FFT algorithm (N=8)

2.1.4 Harmonic Product Spectrum [17]

According to the Harmonic Product Spectrum method, fundamental frequency is determined as follows. First, frequencies of the audio signal's higher harmonic components are measured. Then, the greatest common divisor of these harmonic frequencies is computed, by manipulating the frequency histogram for each harmonic frequency. The fundamental frequency is hence identified by localizing the peak of the histogram. Graphical illustration is included below.

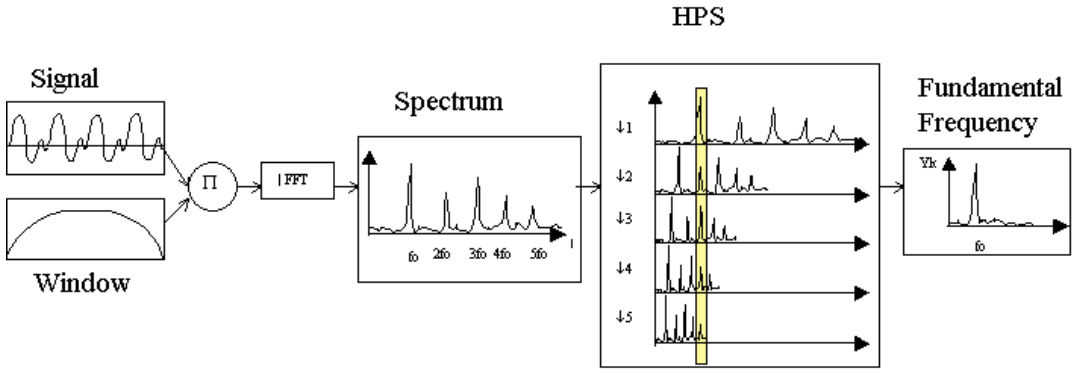


Figure 2. Harmonic Product Spectrum implementation

Harmonic Product Spectrum method is inexpensive to implement, immune to additive & multiplicative noise.

2.1.5 Cepstrum Analysis [18, 19]

The cepstrum is defined as the inverse DFT of the log magnitude of the DFT of a signal

$$c[n] = \mathcal{F}^{-1}\{\log|\mathcal{F}\{x[n]\}|\}$$

Cepstrum analysis takes a series of transforms on digitalized audio data, as shown below.

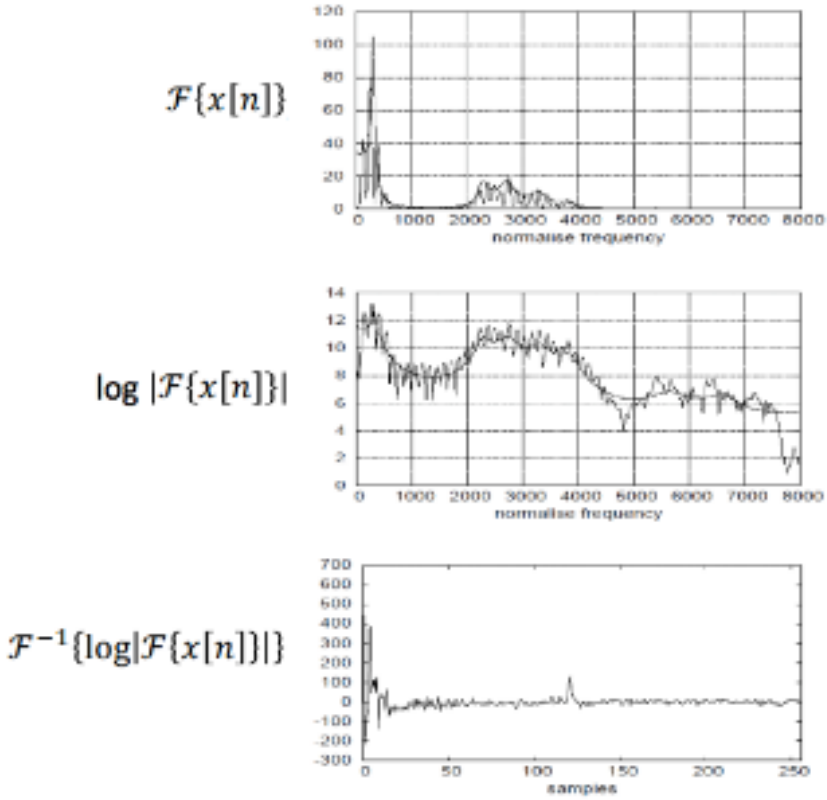


Figure 3. Cepstrum analysis implementation

For the magnitude spectrum $F\{x[n]\}$ of the audio signal, spectra displays harmonics at evenly distributed intervals, whose magnitude drops as frequency rises. The dynamic range is compressed and the amplitude differences in harmonics are decreased by computing the log spectrum $\log|F\{x[n]\}|$. After employing DFT again on the log spectra, the final cepstrum $F^{-1}\{\log|F\{x[n]\}|\}$ is expected to show a large spike around the "period" of the signal. The name of the independent variable of the cepstrum is "quefrency". If there is a large peak in the cepstrum whose quefrency is X samples, the pitch is thus determined by (sampling rate/X) Hz.

2.2 Heart Rate Measurement

Heart rate is among the most vital human health indexes. Existing approaches for monitoring heart rate consists of electrical and optical methods [13]. The electrical method requires a bulky strap around patient's chest. The optical method is more convenient to use and drives the development of low cost heart rate measurement device.

This project developed the heart rate monitor based on the optical technology using the smartphone's built-in camera. Instead of detecting heart rate based on face images, I adopted a more convenient way, to detect heart rate by shooting the fingertip and tracing the colour variation over time.

Heartbeat supplies blood circulation in human body. The red colour intensity of each image frame from the captured video sequences is varying following the heartbeat rhythm. Prior to colour intensity tracking, colour format conversion between the hardware standard and the software implementation should be performed.

2.2.1 Colour space

Colour is the way how the human visual system (HVS) interprets part of the electromagnetic spectrum (300 to 830 nm in wavelength). Due to certain limitations of the human visual system, not all of the visible spectrum combinations can be perceived. Grouping of various spectra forms colours. A colour space is by notation, an approach to specify colours.

2.2.2 YUV and YCbCr formats [22]

YUV and YCbCr formats are widely used colour spaces in electronic applications, such as the TV system. The YUV colour format displays a colour by using the colour components luminance and chrominance. The luminance component (Y) represents the brightness while the chrominance components (U and V) contain the colour differences. Designed for analogue TV transmissions, the YUV colour format guarantees compatibility between black-and-white television and colour television is ensured. Today, the term YUV is often misused with YCbCr, which stands for the colour format of digital video or images. The following image demonstrates how the image is integrated based on the luminance and chrominance components.

Figure 4. Illustration of YUV colour format

Similar to the YUV colour format, the YCbCr colour format consists of a luminance component (Y) and two chrominance components (Cb and Cr). However, the coefficients for colour mapping from and to RGB colour format are different.

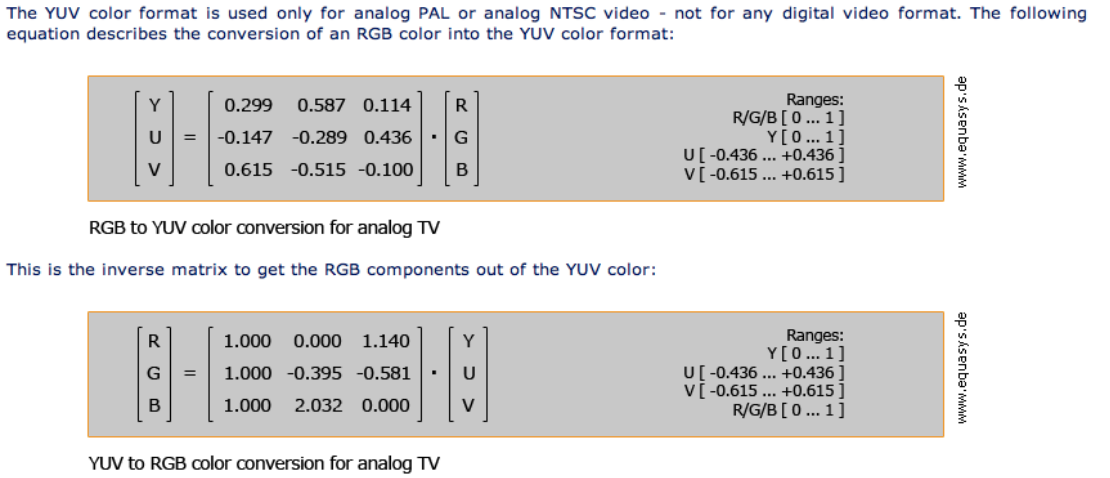


Figure 5. Colour format conversion between RGB and YUV

For digital component video the color format YCbCr is used. For standard definition TV applications (SDTV) the following equation describes the color conversion from RGB to YCbCr (according to ITU-R BT.601):

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} + \begin{bmatrix} 0.257 & 0.504 & 0.098 \\ -0.148 & -0.291 & 0.439 \\ 0.439 & -0.368 & -0.071 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Ranges:
R/G/B [0 ... 255]
Y [16 ... 235]
Cb/Cr [16 ... 240]

www.equasys.de

RGB to YCbCr color conversion for SDTV

To recover an RGB color from a YCbCr color, the following inverse matrix is used:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.164 & 0.000 & 1.596 \\ 1.164 & -0.392 & -0.813 \\ 1.164 & 2.017 & 0.000 \end{bmatrix} \cdot \begin{bmatrix} (Y - 16) \\ (Cb - 128) \\ (Cr - 128) \end{bmatrix}$$

Ranges:
Y [16 ... 235]
Cb/Cr [16 ... 240]
R/G/B [0 ... 255]

www.equasys.de

YCbCr to RGB color conversion for SDTV

Figure 6. Colour format conversion between RGB and YCbCr

2.3 Emotion Recognition

2.3.1 Previous Research Efforts

Though many non-facial approaches have been proposed for emotion recognition, such as speech recognition for emotional keywords [23], voice analysis for tone, pitch, cadence features, and body gesture analysis [1], the most nature, immediate and powerful way humans communicate emotions is through facial expressions [24]. Automatic analysis of facial expression from image data is among the most common approaches in affective computing. Thanks to its non-intrusive property, affective computing on facial images becomes more and more extensively applied, especially with the rapid development of camera and computer vision technology.

The first category of previous work tries to classify facial expressions from image sequences. Essa and Pentland [28] constructed a dynamic parametric model by tracking facial motion over time based on the Facial Action Coding System. The second category of previous work concentrates on static images. Cootes et al. [25] extracted discriminative facial features such as shape and texture information for emotion recognition. Turk and Pentland [29] represented face images by eigenfaces through linear principal component analysis.

Despite the fact that much progress has been accomplished, recognizing facial expression with a high accuracy still remains to be tough due to the variety nature of

facial expressions [26]. An automatic FER system usually tackles the following problems: face detection, facial feature extraction, and facial expression classification [27].

Facial feature extraction aims to locate the most effective representation of the face images for classification. Among the two major approaches, holistic systems process the face image as a whole and obtain a template such as a feature vector. Principal component analysis is widely used to get a low-dimensional representation of the face images. Correspondingly, geometric feature-based systems detect major feature points and/or face components in the images first. A feature vector is established using the distances between feature points and the relative sizes of the major face components. The feature-based techniques are generally more complex to implement than template-based techniques.

2.3.2 EigenFace Model

Eigenface model maps test face images to a small set of characteristic feature images (eigenfaces) which are the principal components of the training face images. Recognition is performed in two steps. Firstly, a new image is projected into the subspace spanned by the eigenfaces trained. After that, the test face is classified by comparing its position in face space with the positions of already known subjects. The approach shows advantages in speed and simplicity, learning capacity, and comparative insensitivity to small or gradual changes in the face image [29]. The principal components analysis (PCA) used in this scheme chooses a dimensionality reducing linear projection that greatly reduces the dimensionality as compared to correlation methods. Researcher have also pointed out that if PCA is computed on face images under varying illumination, the final classification accuracy drops [29].

Chapter 3 Design Procedure

Based on the objectives stated in chapter 1, three main functions were developed in this project named as “Speech Disorder Checker”, “Heart Rate Monitor” and “Emotion Tracker”. Design procedure for each of the feature is discussed in details later in this chapter.

3.1 Speech Disorder Checker

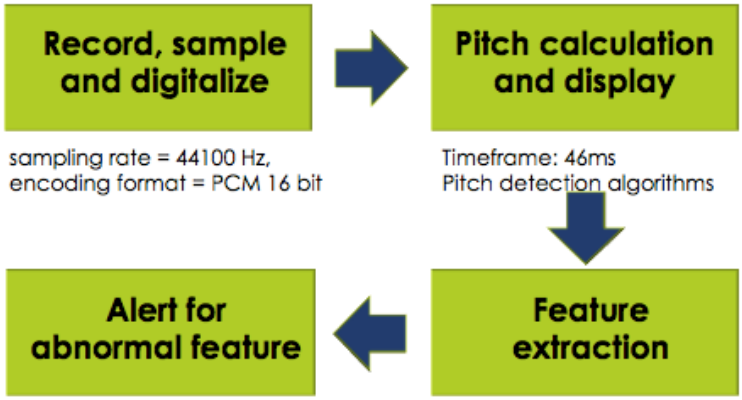


Figure 7. Speech Disorder Checker working mechanism flowchart

3.1.1 Record, Sample and Digitalize

The recording task is performed by Android smartphone’s built-in microphone. With the help of the Android AudioRecord class (provided by Android since API level 3), the sampling rate could be manually configured as 44100 Hz, a standard setting in audio processing field. PCM 16 bit encoding scheme is adopted to digitally represent each audio sample. 16 represents the bit depth of this PCM scheme.

```
private AudioRecord recorder_;
private int buffSize;
private final static int SOURCE = AudioSource.MIC;
//refers to the microphone audio source
private final static int SAMPLE_RATE = 44100;
//audio waveforms are typically sampled at 44.1 kHz
private final static int CHANNEL_MODE = AudioFormat.CHANNEL_IN_MONO;
//describes the configuration of the audio channels
private final static int ENCODING_FORMAT = AudioFormat.ENCODING_PCM_16BIT;
//the audio data is represented PCM 16 bit per sample

buffSize = AudioRecord.getMinBufferSize(SAMPLE_RATE,CHANNEL_MODE,ENCODING_FORMAT);
//Returns the minimum buffer size required for the successful creation
//of an AudioRecord object, in byte units.
recorder_ = new AudioRecord(SOURCE, SAMPLE_RATE, CHANNEL_MODE, ENCODING_FORMAT, buffSize);
```

Figure 8. Code illustration for Audio recording

The AudioRecord class is in charge of the audio resources from the audio input hardware of the platform, in this case, the microphone on Android smartphones. This class manages the resources for Java applications to record audio.

To achieve the goal of real time audio processing such that the user can see his/her talking pitch updating on the device screen, extra thread is allocated to do the background calculation. A Thread is a concurrent execution unit maintaining its own call stack for methods, arguments and local variables. Within the thread, digitalized audio data is stored in a buffer array with the size of 2048 (value of the *CHUNK_SIZE_IN_SAMPLES*) shorts. The buffer size is determined because scientific research has shown that the characteristic of speech signal can be assumed to remain essentially constant over time intervals on the order of 30 or 40 ms [31]. Based on the 44100 Hz sampling rate, the 2048 shorts data buffer can hold the encoded audio data for each 46ms.

```
@Override
public void run()
{
    buffSize = AudioRecord.getMinBufferSize(SAMPLE_RATE,CHANNEL_MODE,ENCODING_FORMAT);
    //Returns the minimum buffer size required for the successful creation
    //of an AudioRecord object, in byte units.
    recorder_ = new AudioRecord(SOURCE, SAMPLE_RATE, CHANNEL_MODE, ENCODING_FORMAT, buffSize);
    if (recorder_.getState() != AudioRecord.STATE_INITIALIZED) {
        Log.e(TAG, "cant initialize");
        return;
    }
    recorder_.startRecording();
    int t = 0;
    while(!interrupted())
    {
        short[] audio_data = new short[CHUNK_SIZE_IN_SAMPLES];
        int numRead = recorder_.read(audio_data, 0, CHUNK_SIZE_IN_SAMPLES);
        //pick audio of 30-50ms each time, 2048 samples, 1/44100 * 2048 = 40+ms
        double best_frequency = AnalyzeFrequencies(audio_data);
        real_pitches.put((double)(t++), best_frequency);
        ff.setPitchResults(best_frequency);
    }
}
```

Figure 9. Code illustration for audio data processing in thread

3.1.2 Pitch Calculation and Display

The stored audio data each 46ms time frame then goes to be manipulated in the function "AnalyzeFrequencies" (algorithm will be elaborated below) with best_frequency value returned, the pitch value of that time frame.

In the mean while, a View class is created and displays the calculated pitch value on the device's screen.

```
public FreqDisplay(Context context) {
    super(context);
    // UI update cycle.
    handler_ = new Handler();
    timer_ = new Timer();
    timer_.schedule(new TimerTask() {
            public void run() {
                handler_.post(new Runnable() {
                    public void run() {
                        invalidate();
                    }
                });
            }
        },
        UI_UPDATE_MS ,
        UI_UPDATE_MS );
}
```

Figure 10. Code illustration of screen display update

Here the constant *UI_UPDATE_MS* is set as 100 ms, meaning the screen updates the current pitch value every 0.1 second.

To implement the pitch detection algorithms in Java code, the task of Fourier transform is of fundamental importance. In this project, I use the JTransforms library developed by Piotr Wendykier to do the FFT calculation [33].

After coding implementation, the performance of methods mentioned in section 2.1.3 – 2.1.5 have been evaluated. The direct FFT works fast but is less robust against environmental noises. This approach takes no account of the property of the human voice such that the interference of the environmental noises is the largest. The second approach, Harmonic Product Spectrum method (code implementation in Figure 11), is inexpensive to implement, comparatively immune to noises and is widely applied in musical tones tracking. After implementing this on Android phones, the recognition rate in low frequency range is in-satisfactory.

```
//method1: harmonic product spectrum analysis
DoubleFFT_1D fft = new DoubleFFT_1D(CHUNK_SIZE_IN_SAMPLES);

double[] data = new double[CHUNK_SIZE_IN_SAMPLES];

final int min_frequency_fft = Math.round(MIN_FREQUENCY
            * CHUNK_SIZE_IN_SAMPLES / SAMPLE_RATE);
final int max_frequency_fft = Math.round(MAX_FREQUENCY
            * CHUNK_SIZE_IN_SAMPLES / SAMPLE_RATE);

data = HanningWindow(audio_data, 0, CHUNK_SIZE_IN_SAMPLES);
fft.realForward(data);

double best_amplitude = 0;
final double draw_frequency_step = 1.0 * SAMPLE_RATE / CHUNK_SIZE_IN_SAMPLES;

List<Double> best_frequencies = new ArrayList<Double>();
List<Double> best_amps = new ArrayList<Double>();

//works as a BPF
for (int i = min_frequency_fft; i <= max_frequency_fft; i++) {
     final double current_frequency = i * draw_frequency_step;
     final double current_amplitude = Math.pow(Math.pow(data[i * 2], 2)
          + Math.pow(data[i * 2 + 1], 2),0.5);

     best_frequencies.add(current_frequency);
     best_amps.add(current_amplitude);

 }

 List<FrequencyCluster> clusters = new ArrayList<FrequencyCluster>();
 FrequencyCluster currentCluster = new FrequencyCluster();

 if (best_frequencies.size() > 0)
 {
        currentCluster.add(best_frequencies.get(0), best_amps.get(0));
 }
```

```
// join clusters
for(int i = 1; i < best_frequencies.size(); i++)
{
        double freq = best_frequencies.get(i);
        double amp = best_amps.get(i);

        if (currentCluster.isNear(freq)) {
                currentCluster.add(freq, amp);
                continue;
        }
        // this isn't near, and isn't harmonic, it's a different one.
        // NOTE: assuming harmonies are consecutive (no unharmonics in between harmonies)
        clusters.add(currentCluster);
        currentCluster = new FrequencyCluster();
        currentCluster.add(freq, amp);
}
// join harmonies,though i dun see the necessity
FrequencyCluster nextCluster;
Log.v(TAG, "cluster size is " + clusters.size());
for(int i = 0; i < clusters.size()-1; i ++) {
    currentCluster = clusters.get(i);
    for(int j = i+1; j < clusters.size(); j++){
        nextCluster = clusters.get(j);
        if (currentCluster.isHarmonic(nextCluster.average_frequency)) {
                currentCluster.total_amplitude *= nextCluster.total_amplitude;
        }
    }
}

best_amplitude = 0;
double best_frequency = 0;

for(int i = 0; i < clusters.size(); i ++) {
        FrequencyCluster clu = clusters.get(i);
        if (best_amplitude < clu.total_amplitude) {
                best_amplitude = clu.total_amplitude;
                best_frequency = clu.average_frequency;
        }
}

return best_frequency;
```

Figure 11. Code illustration of Harmonic Product Spectrum method

The cepstrum analysis method, on the other hand, is also easy to implement (code seen in Figure 12). Moreover, it performs well in both high and low frequency ranges.

```
public FreqResult AnalyzeFrequencies(short[] audio_data) {
    //cepstrum analysis
    FreqResult fr = new FreqResult();
    DoubleFFT_1D fft = new DoubleFFT_1D(CHUNK_SIZE_IN_SAMPLES);

    double[] data = new double[CHUNK_SIZE_IN_SAMPLES];

    final int min_frequency_fft = Math.round(MIN_FREQUENCY
            * CHUNK_SIZE_IN_SAMPLES / SAMPLE_RATE);
    final int max_frequency_fft = Math.round(MAX_FREQUENCY
            * CHUNK_SIZE_IN_SAMPLES / SAMPLE_RATE);

    //add a Hanning window for lower frequency leakage
    data = HanningWindow(audio_data, 0, CHUNK_SIZE_IN_SAMPLES);
    //step 1 of cepstrum
    fft.realForward(data);

    double[] squared_magnitude = new double [CHUNK_SIZE_IN_SAMPLES/2];
    double[] log_magnitude = new double [CHUNK_SIZE_IN_SAMPLES];
    //perform cepstrum and test if vocalic
    double total_energy = 0;
    double band_energy = 0;
    for (int j = 0; j < CHUNK_SIZE_IN_SAMPLES/2; j++) {
        double current_frequency  = j * SAMPLE_RATE / CHUNK_SIZE_IN_SAMPLES;
        //step 2 of cepstrum
        squared_magnitude[j] = Math.pow(data[2*j],2) + Math.pow(data[2*j + 1],2);
        total_energy += squared_magnitude[j];
        if(current_frequency >= 80 && current_frequency <= 2500)
            band_energy += squared_magnitude[j];
        //step 3 of cepstrum
        log_magnitude[2*j] = Math.log10(squared_magnitude[j]);
        log_magnitude[2*j+1] = 0;
    }
    //step 4 of cepstrum
    fft.realInverse(log_magnitude, false);
```

Figure 12. Code illustration of Cepstrum analysis

3.1.3 Feature Extraction

Pitch calculated for each time frame is thereafter stored in a LinkedHashMap data structure. LinkedHashMap is the Hash table and linked list implementation of the Map interface, with sequential iteration order.

As long as the thread specified for background pitch calculation is not interrupted by the user, the LinkedHashMap is updated each time new pitch information in inserted. Due to its property stated above, sequential ordering and the duplicate entry would both be maintained. Once the thread is interrupted by the user, the stored data in the LinkedHashMap would go through a curve-fitting algorithm. At current stage, simple linear regression is used with the help of the Commons Math library developed by Apache [34]. The curve fitting function is written to generate the pitch contour. With linear regression applied, contour with either rising or falling shape will be generated.

3.1.4 Alert for Abnormal Feature

Once the user stops recording and the background processing task finishes, the pitch contour together with the pitch range analysis would be displayed on the device's screen. If the user wants to check for dysprosody, based on the pitch contour generated, the user can check if expected shape is shown. Rising shape pitch contour represents interrogative intonation and falling shape pitch contour can be mapped to declarative intonation. If the user wants to check for voice disorders, he/she can refer to the pitch range analysis which contains the calculated lowest and highest pitch level of the recorded speech. If abnormal range (outside 80-260 Hz) is detected, alert messages will be included in the analysis as well.

3.2 Heart Rate Monitor

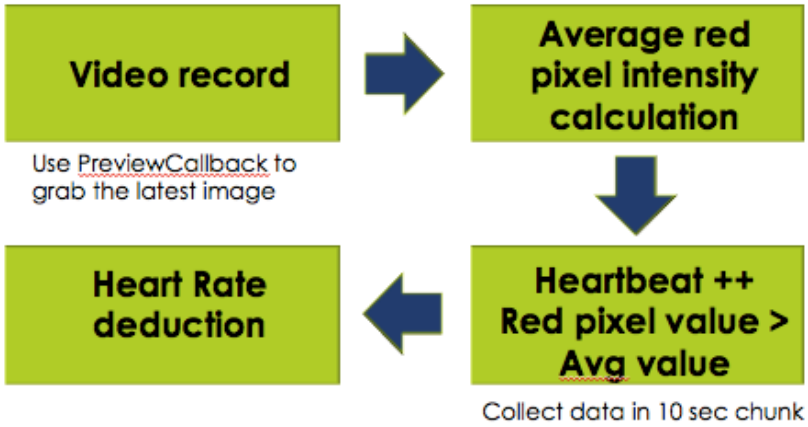


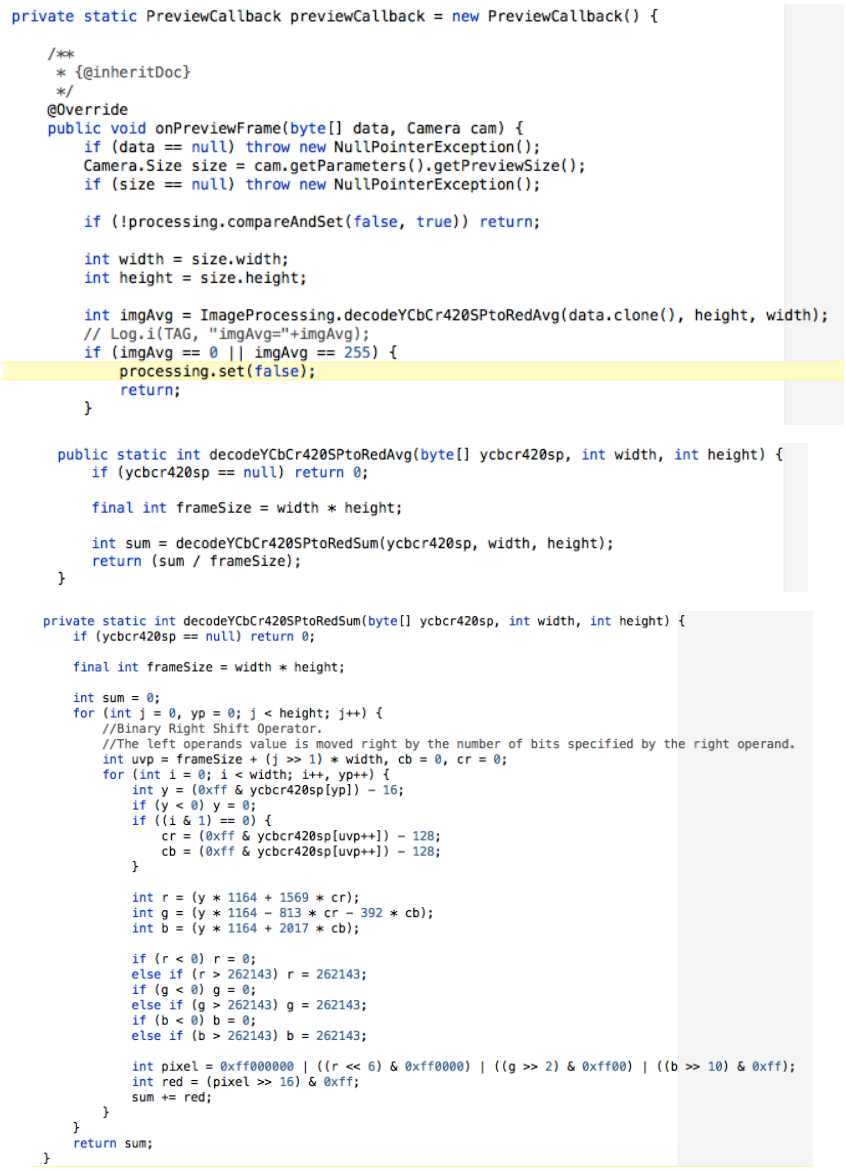
Figure 13. Heart Rate Monitor working mechanism flowchart

3.2.1 Video Record

As mentioned above, optical analysis for heart rate measurement proves to be accurate and convenient. In this project, the heart rate is monitored based on the finger tip colour variation. If you open your phone, better set the flash on, and cover the camera with your index finger tip. You will find that the preview image is red in generally and its red pixel intensity varies with time, with the same rate as blood pulse.

Colour conversion is performed each time an image frame is captured by the Android Camera.PreviewCallback callback interface. It is employed to send copies of preview frames once they are displayed. Within the interface, function OnPreviewFrame(byte[] data, Camera camera) is called as preview frames are displayed and image data of the frame is stored in the "data" byte array. The default colour format in Android display is the YCbCr_420_SP (NV21) format.

Figure 14 below shows that the image data related to the latest frame shown on the device screen is cloned and passed to the "decodeYCbCr420SPtoRedAvg" function, in which the colour format conversion is conducted based on the theories introduced in section 2.2.2. Red, green and blue colour component intensities are calculated.



```
private static PreviewCallback previewCallback = new PreviewCallback() {

    /**
     * {@inheritDoc}
     */
    @Override
    public void onPreviewFrame(byte[] data, Camera cam) {
        if (data == null) throw new NullPointerException();
        Camera.Size size = cam.getParameters().getPreviewSize();
        if (size == null) throw new NullPointerException();

        if (!processing.compareAndSet(false, true)) return;

        int width = size.width;
        int height = size.height;

        int imgAvg = ImageProcessing.decodeYCbCr420SPtoRedAvg(data.clone(), height, width);
        // Log.i(TAG, "imgAvg="+imgAvg);
        if (imgAvg == 0 || imgAvg == 255) {
            processing.set(false);
            return;
        }
```

```
    public static int decodeYCbCr420SPtoRedAvg(byte[] ycbcr420sp, int width, int height) {
        if (ycbcr420sp == null) return 0;

        final int frameSize = width * height;

        int sum = decodeYCbCr420SPtoRedSum(ycbcr420sp, width, height);
        return (sum / frameSize);
    }
```

```
    private static int decodeYCbCr420SPtoRedSum(byte[] ycbcr420sp, int width, int height) {
        if (ycbcr420sp == null) return 0;

        final int frameSize = width * height;

        int sum = 0;
        for (int j = 0, yp = 0; j < height; j++) {
            //Binary Right Shift Operator.
            //The left operands value is moved right by the number of bits specified by the right operand.
            int uvp = frameSize + (j >> 1) * width, cb = 0, cr = 0;
            for (int i = 0; i < width; i++, yp++) {
                int y = (0xff & ycbcr420sp[yp]) - 16;
                if (y < 0) y = 0;
                if ((i & 1) == 0) {
                    cr = (0xff & ycbcr420sp[uvp++]) - 128;
                    cb = (0xff & ycbcr420sp[uvp++]) - 128;
                }

                int r = (y * 1164 + 1569 * cr);
                int g = (y * 1164 - 813 * cr - 392 * cb);
                int b = (y * 1164 + 2017 * cb);

                if (r < 0) r = 0;
                else if (r > 262143) r = 262143;
                if (g < 0) g = 0;
                else if (g > 262143) g = 262143;
                if (b < 0) b = 0;
                else if (b > 262143) b = 262143;

                int pixel = 0xff000000 | ((r << 6) & 0xff0000) | ((g >> 2) & 0xff00) | ((b >> 10) & 0xff);
                int red = (pixel >> 16) & 0xff;
                sum += red;
            }
        }
        return sum;
    }
```

Figure 14. Code illustration of colour format conversion between YCbCr and RGB

3.2.2 Average Red Intensity Calculation & 3.2.3 Heartbeat Counting

The basic mechanism of this heart rate monitor feature is to calculate the average red colour intensity of the preview frame over time and to count the heartbeat each time the latest frame captured displays red intensity exceeds the average threshold value.

Back in the PreviewCallBack interface's OnPreviewFrame(byte[] data, Camera camera) function, the average red component in the preview image sequences is updated each time a new image frame is captured and its calculated red component intensity is processed. The comparison based on the average red component intensity and the update of the red component intensity is running at the same time (see Figure 15 below). To set the average

calculation easy to operate, the averageArray is set to of length 4. The updating of the average value ensures the real-time measuring accuracy and the relatively small array size does not introduce much computation burden.

```
int averageArrayAvg = 0;
int averageArrayCnt = 0;
for (int i = 0; i < averageArray.length; i++) {
    if (averageArray[i] > 0) {
        averageArrayAvg += averageArray[i];
        averageArrayCnt++;
    }
}

int rollingAverage = (averageArrayCnt > 0) ? (averageArrayAvg / averageArrayCnt) : 0;
TYPE newType = currentType;
if (imgAvg < rollingAverage) {
    newType = TYPE.RED;
    if (newType != currentType) {
        beats++;
        // Log.d(TAG, "BEAT!! beats="+beats);
    }
} else if (imgAvg > rollingAverage) {
    newType = TYPE.GREEN;
}

if (averageIndex == averageArraySize) averageIndex = 0;
averageArray[averageIndex] = imgAvg;
averageIndex++;
```

Figure 15. Code illustration of average red intensity calculation and heartbeat counting

The testing stage is set to 10s. During this time period, each time the red value on the latest image frame exceeds the average red intensity, the heartbeat is incremented.

3.2.4 Heart Rate deduction

Based on the data collected, heartbeat is thus calculated by expanding the 10s testing period to 60s, as illustrated in Figure 16. If the irregularDetected() function is called and the alerting message is shown continuously, precautions should be taken.

```
long endTime = System.currentTimeMillis();
double totalTimeInSecs = (endTime - startTime) / 1000d;
if (totalTimeInSecs >= 10) {
    double bps = (beats / totalTimeInSecs);
    int dpm = (int) (bps * 60d);
    if (dpm < 30 || dpm > 180) {
        startTime = System.currentTimeMillis();
        beats = 0;
        processing.set(false);
        irregularDetected();
        return;s
    }

    // Log.d(TAG,
    // "totalTimeInSecs="+totalTimeInSecs+" beats="+beats);

    if (beatsIndex == beatsArraySize) beatsIndex = 0;
    beatsArray[beatsIndex] = dpm;
    beatsIndex++;

    int beatsArrayAvg = 0;
    int beatsArrayCnt = 0;
    for (int i = 0; i < beatsArray.length; i++) {
        if (beatsArray[i] > 0) {
            beatsArrayAvg += beatsArray[i];
            beatsArrayCnt++;
        }
    }
    int beatsAvg = (beatsArrayAvg / beatsArrayCnt);
    text.setText(String.valueOf(beatsAvg));
    startTime = System.currentTimeMillis();
    beats = 0;
}
processing.set(false);
```

Figure 16. Code illustration of heart rate deduction

3.3 Emotion Tracker

Among the many algorithms dedicated to automatic facial expression recognition, their compatibility to mobile computing is limited. Due to high demand of real time processing ability and large volume of data to be analysed, most of the algorithms are computationally expensive, especially when the requirement of accuracy is high. Henceforth, C++ is the optimal language to implement these algorithms. However, problems occur since I use the Java programming languages in developing this project, which is also the dominant language in mobile computing for Android. Similarly, most of the widely used image processing libraries were originally developed in C++ and are still more suitable to be applied in C++ development environment, such as the OpenCV library [35] used in this project.

OpenCV (Open Source Computer Vision) is a library of programming functions mainly aimed at real-time computer vision, developed by Intel Russia research center in Nizhny Novgorod, and now supported by Willow Garage and Itseez. It focuses mainly on real-time image processing.

Facing the problem of incompatible development methodology, certain wrapper functions developed by programmers was used to ease the difficulty of cross-platform programming. In this project, a famous wrapper function called "JavaCV" [36] was used.

JavaCV first provides wrappers to commonly used libraries by researchers in the field of computer vision: OpenCV, libdc1394, videoInput, FFmpeg, PGR FlyCapture, and ARToolKitPlus, OpenKinect.

With the help of these computer vision libraries and wrapper functions, technical difficulties of analysing facial images are eased. Figure 17 below illustrates the working mechanism of this "Emotion Tracker" feature.

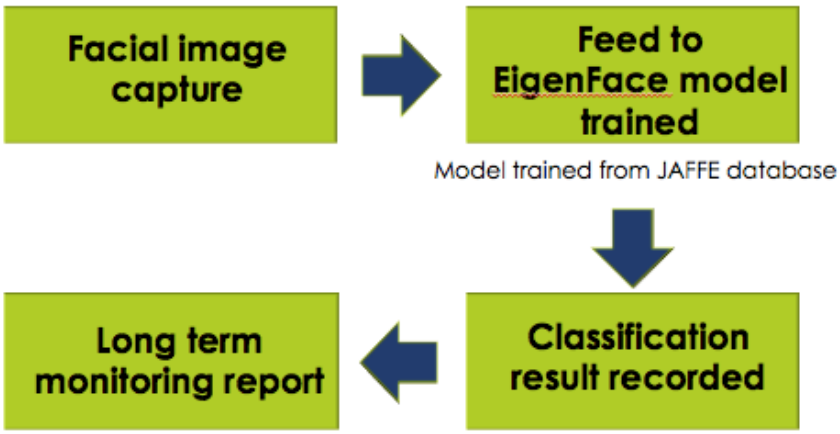


Figure 17. Emotion Tracker working mechanism

3.3.1 Facial image capture

3.3.1.1 Camera Configuration

After granting camera usage permission and external storage access control in the Android application's manifest.xml file, camera object and camera preview on screen could be easily implemented by coding. To better facilitate the user experience, the frontal camera is used such that users can easily take personal pictures for themselves.

```
public void surfaceCreated(SurfaceHolder holder) {
    // The Surface has been created, acquire the camera and tell it where to draw.
    try {
        mCamera = Camera.open(Camera.getNumberOfCameras()-1);
        // WARNING: without permission in Manifest.xml, crashes
    }
    catch (RuntimeException exception) {
        //Log.i(TAG, "Exception on Camera.open(): " + exception.toString());

    }

    // The Surface has been created, now tell the camera where to draw the preview.
    try {
        mCamera.setPreviewDisplay(holder);
        mCamera.setDisplayOrientation(90);
        //mCamera.startPreview();
        updateBufferSize();
        mCamera.addCallbackBuffer(mBuffer); // where we'll store the image data
        //Installs a callback to be invoked for every preview frame,
        //using buffers supplied with addCallbackBuffer(byte[]),
        //in addition to displaying them on the screen.
        mCamera.setPreviewCallbackWithBuffer(new Camera.PreviewCallback() {
            public synchronized void onPreviewFrame(byte[] data, Camera c) {

                if (mCamera != null) { // there was a race condition when onStop() was called..
                    mCamera.addCallbackBuffer(mBuffer); // it was consumed by the call, add it back
                }
            }
        });
    } catch (IOException e) {
        Log.d(TAG, "Error setting camera preview: " + e.getMessage());
        if (mCamera != null){
            mCamera.release();
            mCamera = null;
        }
    }

}
```

```
public void surfaceDestroyed(SurfaceHolder holder) {
    // empty. Take care of releasing the Camera preview in your activity.
    mCamera.stopPreview();
    mCamera.release();
    mCamera = null;
}

public void surfaceChanged(SurfaceHolder holder, int format, int w, int h) {
    try {
        mParameters = mCamera.getParameters();
        //mParameters.set("orientation","portrait");
        for (Integer i : mParameters.getSupportedPreviewFormats()) {
            Log.d(TAG, "supported preview format: " + i);
        }

        List<Size> sizes = mParameters.getSupportedPreviewSizes();
        for (Size size : sizes) {
            Log.i(TAG, "supported preview size: " + size.width + "x" + size.height);
        }
        mCamera.setParameters(mParameters); // apply the changes
    } catch (Exception e) {
        // older phone - doesn't support these calls
    }

    Size p = mCamera.getParameters().getPreviewSize();
    //Log.i(TAG, "Preview: checking it was set: " + p.width + "x" + p.height); // DEBUG

    try {
        mCamera.setPreviewDisplay(mHolder);
        mCamera.startPreview();

    } catch (Exception e){
        // ignore: tried to stop a non-existent preview
        Log.d(TAG, "Error starting camera preview: " + e.getMessage());
    }
}
```

Figure 18. Code illustration for opening camera and displaying preview on screen (inside the camera preview class)

3.3.1.2 Face Detection

As discussed before, prior to facial features extraction and expression classification, face detection is the prerequisite to facial expression recognition.

At first, I planned to implement the automatic face detection functionality with the help of the OpenCV library. The face detection functionality in OpenCV was implemented based on the Haar feature-based cascade classifiers as explained in [32]. The program was expected to automatically detect human faces in the camera preview frame and crop the face images out and feed them for further processing.

Problems arose at this stage. First, the detection accuracy rate was not optimal under certain illumination conditions. Moreover, as the distance from the camera to the person being shot varied, the face area in each camera preview frame varied in size. If automatic cropping of face image got processed, different test images would be generated, which introduced extra complexity to the classification process. Images below shows the detection results when wiping the testing smartphone around the classroom. As you may already notice, certain objects not human faces were detected.

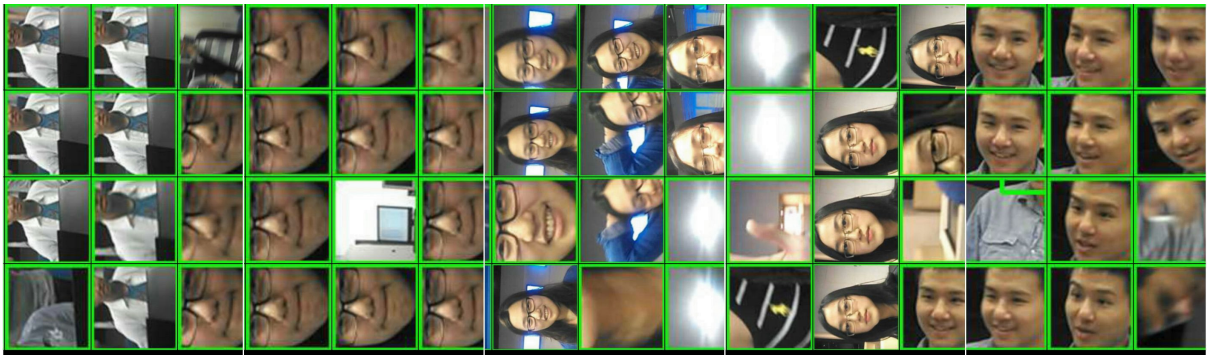


Figure 19. List of automatics face detection results using Haar cascade classifiers

Due to the performance limitation, this automatic feature was hence discarded.

To compensate, also inspired by other application's design, I drew a square box above the camera preview frame and located it in the centre area of the screen. This was realized by creating a view class for drawing a rectangle box. In the layout file of this activity, the self-created box view was merged on top of the camera preview frame (see Figure 20).

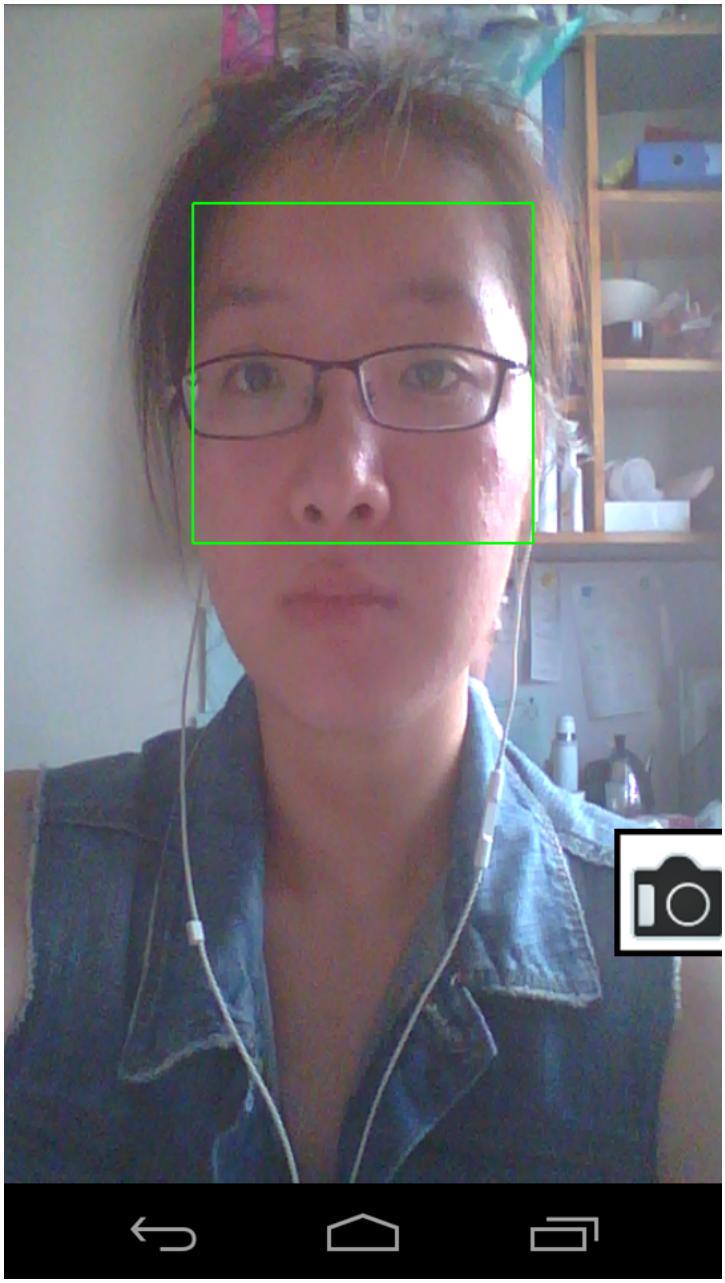


Figure 20. Screen display for facial image capture

Once the capture icon is clicked, the program would capture the image bounded by the box and store it in the device's external storage.

```
// This method takes the preview image, grabs the rectangular
// part of the image selected by the bounding box and saves it.
// A thread is needed to save the picture so not to hold the UI thread.
private OnClickListener previewListener = new OnClickListener() {

    @Override
    public void onClick(View v) {
        //if (mAutoFocus){
            mAutoFocus = false;
            //mPreview.setFlash(false);
            //mPreview.setCameraFocus(myAutoFocusCallback);
            Wait.oneSec();
            Log.i("TESTESTEST","Taking picture");
            Thread tGetPic = new Thread( new Runnable() {
                public void run() {
                    int left = (int) (mView.getmLeftTopPosX());
                    int top = (int) (mView.getmLeftTopPosY());
                    int right = (int)(mView.getmRightBottomPosX());
                    int bottom = (int)(mView.getmRightBottomPosY());
                    savePhoto(mPreview.getPic(left,top,right,bottom), curTime);
                    mAutoFocus = true;
                }
            });
            tGetPic.start();
            Wait.oneSec();
            gotoRec(curTime);

        //}
        boolean pressed = false;
        if (!mTakePicture.isPressed()){
            pressed = true;
        }
    }
};
```

```
public Bitmap getPic(int left, int top, int right, int bottom) {
    System.gc();
    Bitmap b = null;
    //mParameters = mCamera.getParameters();
    Size s = mParameters.getPreviewSize();
    //Returns the dimensions setting for preview pictures.
    Log.d(TAG, "the width of s is " + s.width);
    Log.d(TAG, "the height of s is " + s.height);
    //YuvImage is used, compressed to jpeg and turned to bitmap.
    YuvImage yuvimage = new YuvImage(mBuffer, ImageFormat.NV21, s.width, s.height, null);
    ByteArrayOutputStream outStream = new ByteArrayOutputStream();
    yuvimage.compressToJpeg(new Rect(s.width-bottom + 80, left - 20, s.width-top + 80, right - 20), 100, outStream);
    // make JPG
    b = BitmapFactory.decodeByteArray(outStream.toByteArray(), 0, outStream.size());
    if (b != null) {
        Log.d(TAG, "getPic() WxH:" + b.getWidth() + "x" + b.getHeight());
    } else {
        Log.d(TAG, "getPic(): Bitmap is null..");
    }
    yuvimage = null;
    outStream = null;
    System.gc();
    return b;
}
private boolean savePhoto(Bitmap bm, long curTime) {
    String outputPath = Environment.getExternalStorageDirectory().getAbsolutePath();
    File imageOutput = new File(outputPath + "/" + curTime + ".jpg");
    FileOutputStream image = null;
    try {
        image = new FileOutputStream(imageOutput);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }

    bm.compress(CompressFormat.JPEG, 100, image);
    MediaStore.Images.Media.insertImage(getContentResolver(), rotatedBitmap, "test.jpg" , "test");
    //extra
    if (bm != null) {
        int h = bm.getHeight();
        int w = bm.getWidth();
        //Log.i(TAG, "savePhoto(): Bitmap WxH is " + w + "x" + h);
    } else {
        //Log.i(TAG, "savePhoto(): Bitmap is null..");
        return false;
    }
    return true;
}
```

Figure 21. Code illustration for facial image capture

This functionality was implemented by adding a savephoto function inside the capture icon's onclicklistener. Inside the savephoto function, the coordinates of the bounding box were used. First, a YUV image object with the size of the whole preview screen size was initialized as a buffer to hold the image data. Secondly, the bounding coordinates were used to specify the exact dimension of image data to be used in the buffer. Finally, the YUV image object uses the coordinates information to crop out useless image data and stored the face image.

3.3.2 Feed Test Image to EigenFace Model

Among the many models proved accurate for facial expression recognition, in this project, the task is accomplished by using the EigenFace model.

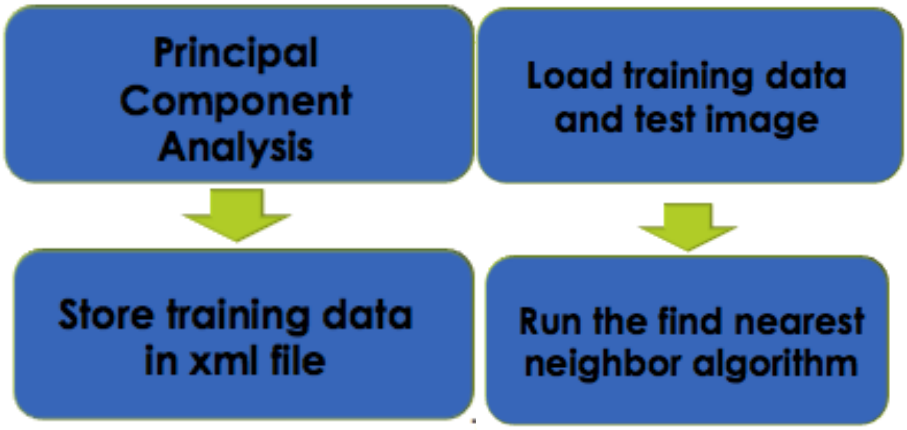


Figure 21. EigenFace model working mechanism flowchart

Figure above illustrates how the eigenface model is constructed by extracting facial features from the training images and how it can be applied on test image to get expression classification.

In this project, training images were derived from the Japanese Female Facial Expression database available online [37].

Figure 22. Samples of Japanese Female Facial Expression Database

To construct the training model, eigenface for each training image is produced. This process was performed during project development on computer not real time on mobile devices.
First, I collected useful images from the database and divide them to two categories: happy and sad. I created a plain text file and dedicated each line for each image instance, apart from specifying the directory path in local drive, I labelled each image with expression category it represents, examples (two emotion status IDs were also assigned with 1 - sad & 2 - happy):

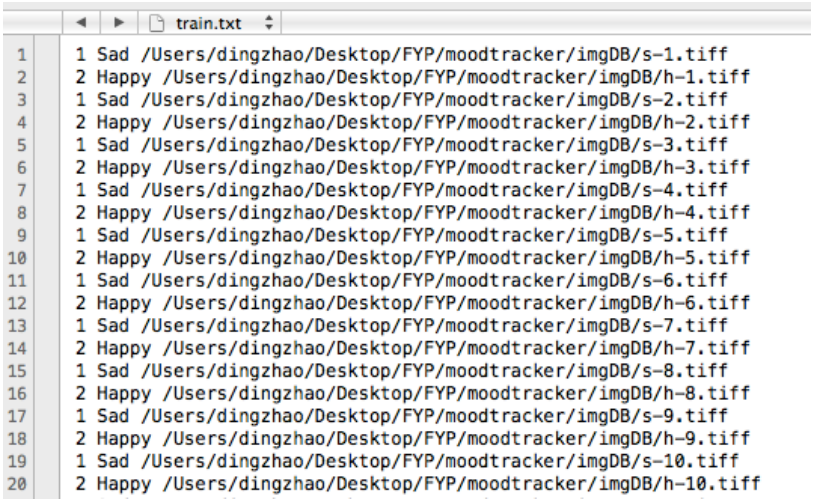


Figure 23. Text file for training eigenface model

Thereafter I included the text file's path as the training program's input. During model training, the program scanned the text file line-by-line, loaded images sequentially and stored their corresponding emotion category in dedicated matrix.

```
public void learn(final String trainingFileName) {
  int i;

  // load training data
  LOGGER.info("=========================================");
  LOGGER.info("Loading the training images in " + trainingFileName);
  trainingFaceImgArr = loadFaceImgArray(trainingFileName);
  nTrainFaces = trainingFaceImgArr.length;
  LOGGER.info("Got " + nTrainFaces + " training images");
  if (nTrainFaces < 3) {
    LOGGER.severe("Need 3 or more training faces\n"
          + "Input file contains only " + nTrainFaces);
    return;
  }

  // do Principal Component Analysis on the training faces
  doPCA();

  LOGGER.info("projecting the training images onto the PCA subspace");
  // project the training images onto the PCA subspace
  projectedTrainFaceMat = cvCreateMat(
          nTrainFaces, // rows
          nEigens, // cols
          CV_32FC1); // type, 32-bit float, 1 channel

  // initialize the training face matrix - for ease of debugging
  for (int i1 = 0; i1 < nTrainFaces; i1++) {
    for (int j1 = 0; j1 < nEigens; j1++) {
      projectedTrainFaceMat.put(i1, j1, 0.0);
    }
  }

  LOGGER.info("created projectedTrainFaceMat with " + nTrainFaces +
      " (nTrainFaces) rows and " + nEigens + " (nEigens) columns");
  if (nTrainFaces < 5) {
    LOGGER.info("projectedTrainFaceMat contents:\n" + oneChannelCvMatToString(projectedTrainFaceMat));
  }


  final FloatPointer floatPointer = new FloatPointer(nEigens);
  for (i = 0; i < nTrainFaces; i++) {
    cvEigenDecomposite(
            trainingFaceImgArr[i], // obj
            nEigens, // nEigObjs
            eigenVectArr, // eigInput (Pointer)
            0, // ioFlags
            null, // userData (Pointer)
            pAvgTrainImg, // avg
            floatPointer); // coeffs (FloatPointer)

    if (nTrainFaces < 5) {
      LOGGER.info("floatPointer: " + floatPointerToString(floatPointer));
    }
    for (int j1 = 0; j1 < nEigens; j1++) {
      projectedTrainFaceMat.put(i, j1, floatPointer.get(j1));
    }
  }
  if (nTrainFaces < 5) {
    LOGGER.info("projectedTrainFaceMat after cvEigenDecomposite:\n" + projectedTrainFaceMat);
  }

  // store the recognition data as an xml file
  storeTrainingData();

  // Save all the eigenvectors as images, so that they can be checked.
  storeEigenfaceImages();
}
```

Figure 24. Code illustration for eigenface model construction

After the images are loaded successfully, the Principle Component Analysis was conducted on each image. In this project, the PCA function implemented in JavaCV library was employed.

```
private void doPCA() {
  int i;
  CvTermCriteria calcLimit;
  CvSize faceImgSize = new CvSize();

  // set the number of eigenvalues to use
  nEigens = nTrainFaces - 1;

  LOGGER.info("allocating images for principal component analysis, using "
     + nEigens + (nEigens == 1 ? " eigenvalue" : " eigenvalues"));

  // allocate the eigenvector images
  faceImgSize.width(trainingFaceImgArr[0].width());
  faceImgSize.height(trainingFaceImgArr[0].height());
  eigenVectArr = new IplImage[nEigens];
  for (i = 0; i < nEigens; i++) {
    eigenVectArr[i] = cvCreateImage(
            faceImgSize, // size
            IPL_DEPTH_32F, // depth
            1); // channels
  }

  // allocate the eigenvalue array
  eigenValMat = cvCreateMat(
          1, // rows
          nEigens, // cols
          CV_32FC1); // type, 32-bit float, 1 channel

  // allocate the averaged image
  pAvgTrainImg = cvCreateImage(
          faceImgSize, // size
          IPL_DEPTH_32F, // depth
          1); // channels


  // set the PCA termination criterion
  calcLimit = cvTermCriteria(
          CV_TERMCRIT_ITER, // type
          nEigens, // max_iter
          1); // epsilon

  LOGGER.info("computing average image, eigenvalues and eigenvectors");
  // compute average image, eigenvalues, and eigenvectors
  cvCalcEigenObjects(
          nTrainFaces, // nObjects
          trainingFaceImgArr, // input
          eigenVectArr, // output
          CV_EIGOBJ_NO_CALLBACK, // ioFlags
          0, // ioBufSize
          null, // userData
          calcLimit,
          pAvgTrainImg, // avg
          eigenValMat.data_fl()); // eigVals

  LOGGER.info("normalizing the eigenvectors");
  cvNormalize(
          eigenValMat, // src (CvArr)
          eigenValMat, // dst (CvArr)
          1, // a
          0, // b
          CV_L1, // norm_type
          null); // mask
}
```

Figure 25. Code illustration for Principal Component Analysis on images

Upon PCA completion, an eigenvector array would thus be generated with each element corresponding to the training images. With certain degree of transformation, each eigenvector could be graphically converted to bitmap images (as shown below).

Figure 26. Eigenfaces for every training image

From the eigenfaces above, it is noticeable that the eye brow area, eye, nose and mouse are of crucial importance for expression recognition as compared to other face area.

In the meanwhile, an average eigenface model based on all the training images merged is generated as well.

Figure 27. Average Eigenface developed from training images

Based on these training data collected, an EigenDecomposite function was engaged to calculate all decomposition coefficients for the input object using the previously calculated eigenvector array and the averaged eigenface merged. It performed the mapping to the lower

dimensional space. In return, a coefficient value was allocated for each training object, which would be used in future classification.

Once the training procedure finished, related data mentioned so far would all be stored in a newly generated xml file. Each time the user enters this "emotion tracker" function in the application, the xml file would automatically be loaded to the user device's local file directory for future access.

So far, the eigenface training model has been fully built and waits for test images to be loaded. Everytime the test image is passed from the camera to the training module, the EigenDecomposite function is performed, based on the eigenvector array and average eigenface stored above. Similarly, a coefficient is assigned to the test image.

```
// project the test image onto the PCA subspace
cvEigenDecomposite(
        testFaceImg, // obj
        nEigens, // nEigObjs
        eigenVectArr, // eigInput (Pointer)
        0, // ioFlags
        null, // userData
        pAvgTrainImg, // avg
        projectedTestFace);  // coeffs

//LOGGER.info("projectedTestFace\n" + floatArrayToString(projectedTestFace));

final FloatPointer pConfidence = new FloatPointer(confidence);
iNearest = findNearestNeighbor(projectedTestFace, new FloatPointer(pConfidence));
confidence = pConfidence.get();
truth = personNumTruthMat.data_i().get(i);
nearest = trainPersonNumMat.data_i().get(iNearest);
if (nearest == 1)
    LOGGER.info("Sad Face Detected!!!!");
else
    LOGGER.info("Happy Face Detected!!!!");
```

Figure 28. Code illustration for EigenDecomposite on the test image

3.3.3 Expression Classification and result recording

Once the coefficients of all training images and of the test image are successfully calculated, a findNearestNeighbor function is called to do the similarity check on the test image's coefficient and all training images' coeffients. The test image's emotion status is assigned as the emotion status of the particular training image whose coefficient is of the highest similarity.

In this project, the findNearestNeighbor algorithm uses the "Mahalanobis distance" as the measuring criteria, which proves to be more accurate than the commonly known Eucalidean distance.

```
*/
private int findNearestNeighbor(float projectedTestFace[], FloatPointer pConfidencePointer) {
  double leastDistSq = Double.MAX_VALUE;
  int i = 0;
  int iTrain = 0;
  int iNearest = 0;

  LOGGER.info(".............");
  LOGGER.info("find nearest neighbor from " + nTrainFaces + " training faces");
  for (iTrain = 0; iTrain < nTrainFaces; iTrain++) {
    //LOGGER.info("considering training face " + (iTrain + 1));
    double distSq = 0;

    for (i = 0; i < nEigens; i++) {
      //LOGGER.debug("  projected test face distance from eigenface " + (i + 1) + " is " + projectedTestFace[i]);

      float projectedTrainFaceDistance = (float) projectedTrainFaceMat.get(iTrain, i);
      float d_i = projectedTestFace[i] - projectedTrainFaceDistance;
      distSq += d_i * d_i; // / eigenValMat.data_fl().get(i);
      // Mahalanobis distance (might give better results than Eucalidean distance)

    }

    if (distSq < leastDistSq) {
      leastDistSq = distSq;
      iNearest = iTrain;
      LOGGER.info("  training face " + (iTrain + 1) + " is the new best match, least squared distance: " + leastDistSq);
    }
  }

  // Return the confidence level based on the Euclidean distance,
  // so that similar images should give a confidence between 0.5 to 1.0,
  // and very different images should give a confidence between 0.0 to 0.5.
  float pConfidence = (float) (1.0f - Math.sqrt(leastDistSq / (float) (nTrainFaces * nEigens)) / 255.0f);
  pConfidencePointer.put(pConfidence);

  LOGGER.info("training face " + (iNearest + 1) + " is the final best match, confidence " + pConfidence);
  return iNearest;
}
```

Figure 29. Code illustration for FindNearestNeighbour algorithm implementation

Once classification result is computed, it should be written to the project's internal database for further tracking.

3.3.4 Long term monitoring

With the aim to detect chronic stress, this feature could not be complete without the long term monitoring, based on data recorded in the database. On a monthly basis, a scanning of the recorded data would be conducted, alert will be issued if negative emotion status takes more than half of the month.

Chapter 4 Testing Results

After the basic functionalities were fully implemented and the user interface of the application was complete, testing work was performed. Figure 28 below shows the welcome page of the App. Since this application is developed for personal health monitoring, personal account should be specifically created for each user with the attempt for accurate detection result and also for better data access. However, since the application has not yet been deployed to the Google's cloud platform, no cloud storage and database have been allocated for this application, this functionality can not be fully implemented. However, testing performance of three major features would not be affected.

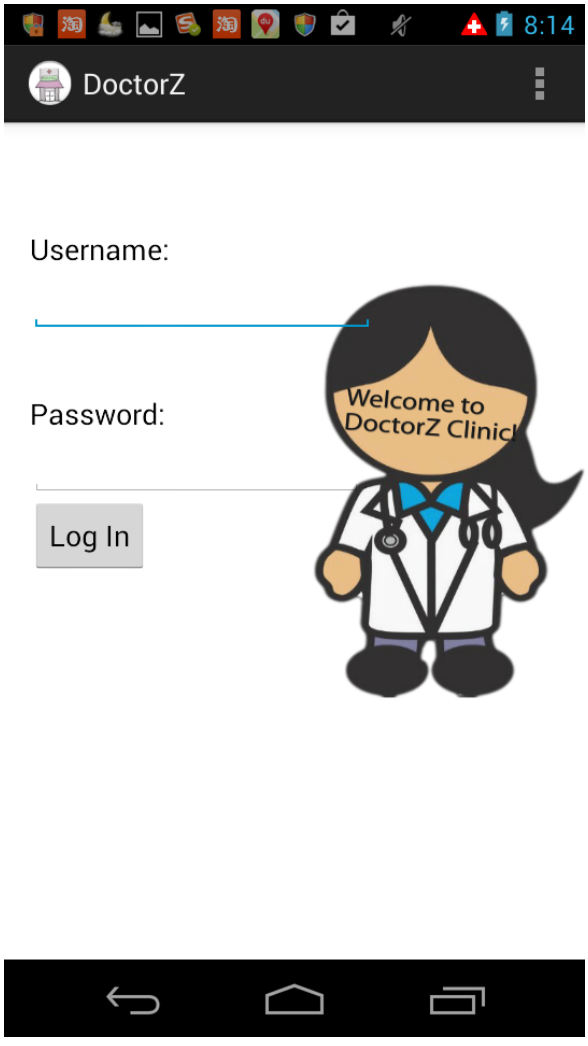


Figure 28. Welcome page of the DoctorZ application

Upon entering the application, first displays an introductory page (Figure 29). By clicking the different navigation drawer items on the left, the user will be directed to the three main functions of the application.

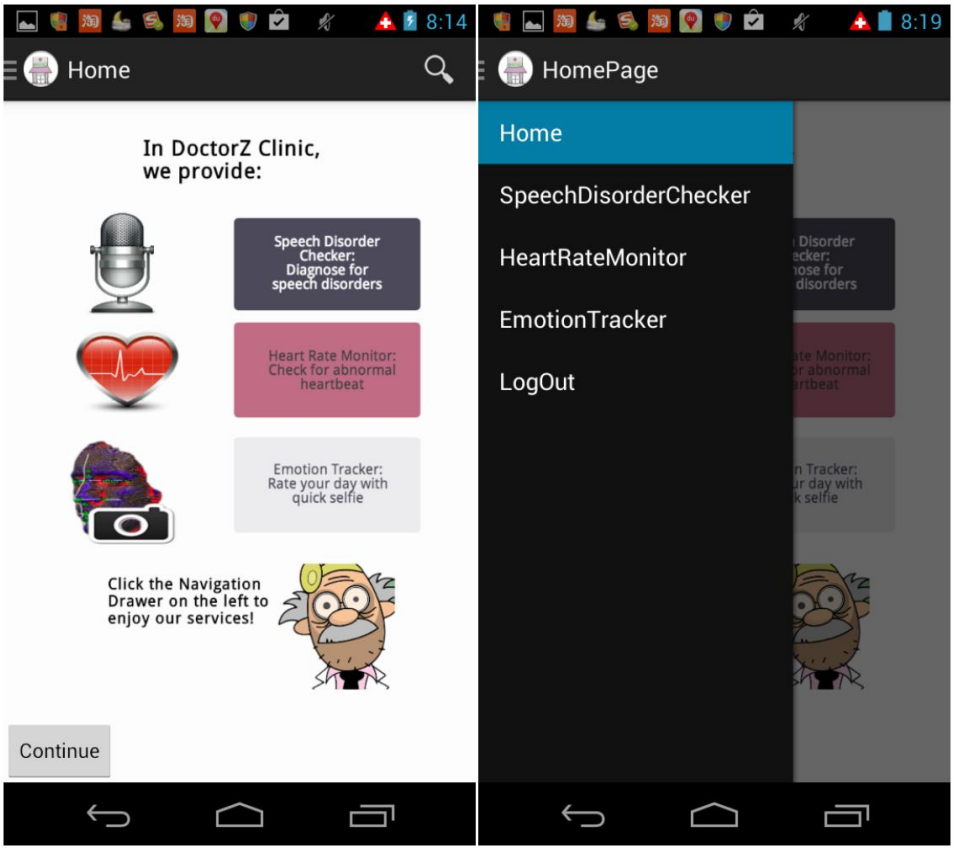


Figure 29. Introduction page of the DoctorZ application

4.1 Speech Disorder Checker

The first function is the Speech Disorder Checker (interface pages shown below in Figure 30).

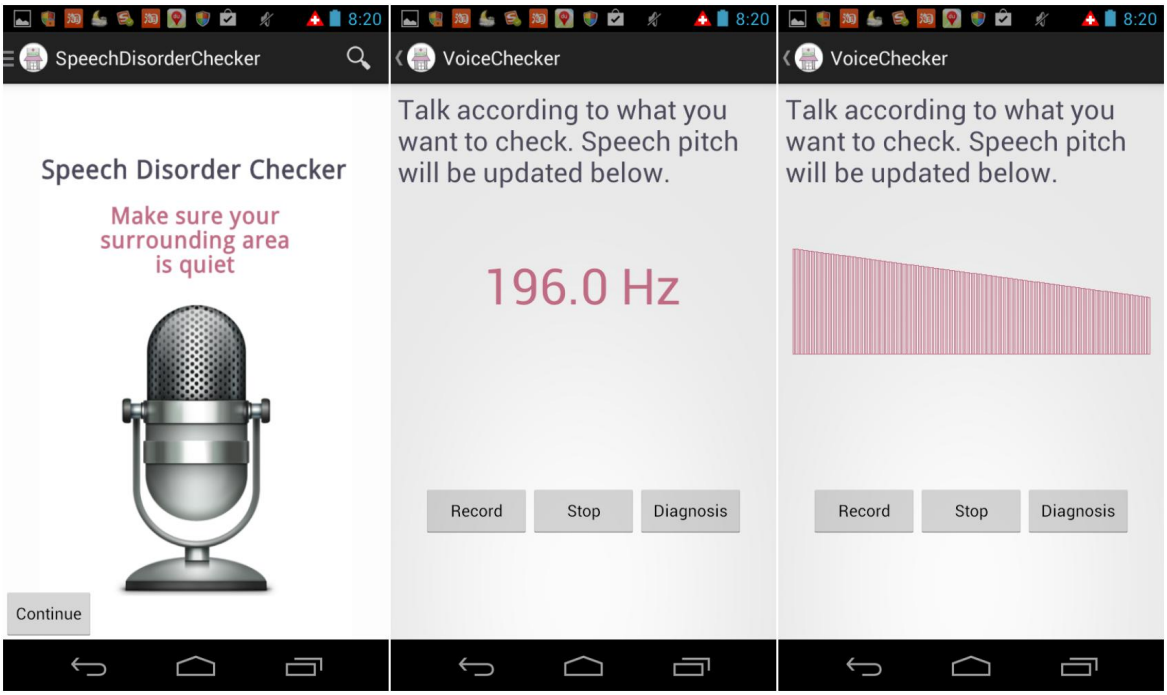


Figure 30. User interface of Speech Disorder Checker

By clicking the "start" button, the user can begin talking and the recording starts. In the meantime, the pitch of his/her speech will be displayed on the screen, updated over time.

Graphs below are cepstrum examples in quefrency domain when I was continuously talking in high-key voice (top graph) and low-key voice (bottom graph).

Figure 31. Cepstrum examples during development

After clicking the “stop” button and the “diagnosis” button, pitch contour would be shown on the screen with pitch range analysis followed.

Figure 32 below shows the pitch contour plotted without linear regression while I was singing “do re mi fa so la si do” with increasing pitch level.

Figure 32. Pitch contour plotted without regression

4.2 Heart Rate Monitor

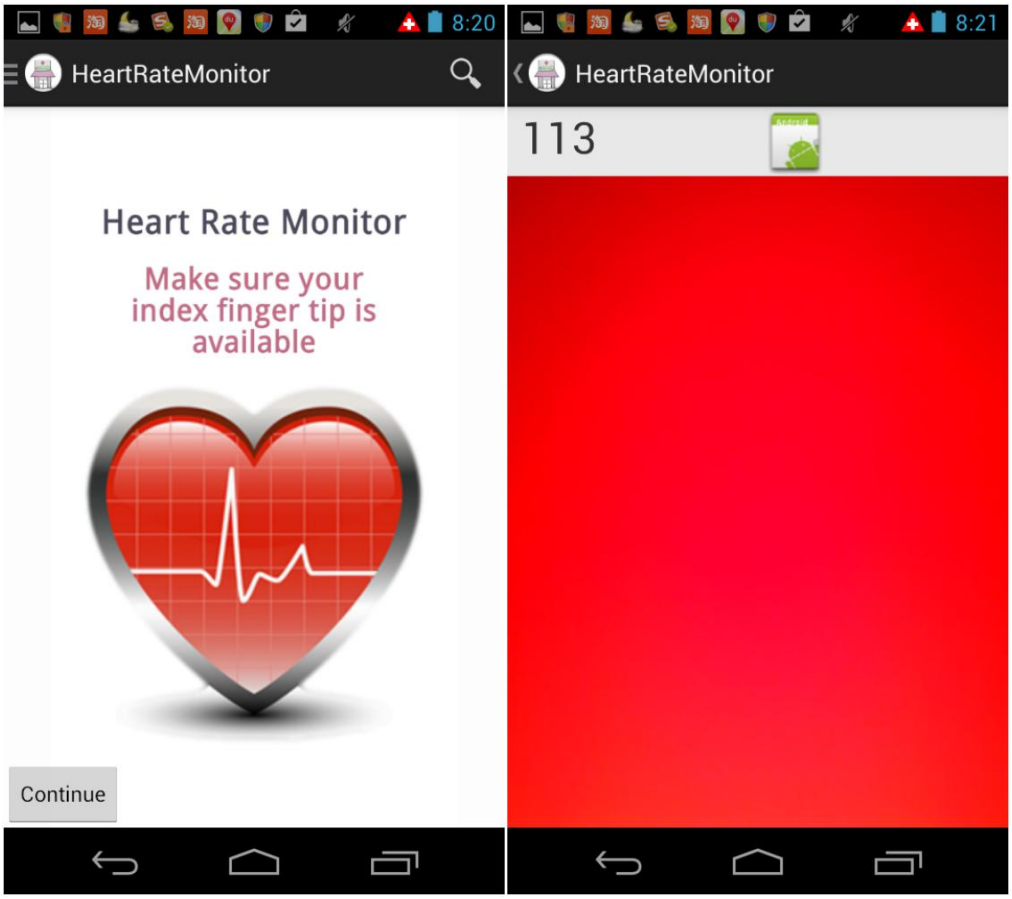


Figure 33. User interface of Heart Rate Monitor

Upon entering this function, the user should put his/her index finger tip above the rear camera of the device. It would be observed that the preview page is generally red. Then, wait until number displays on the top left corner. While waiting, the icons next to the text area is changing colour at the same rhythm of the heartbeat.

As mentioned before, due to the coding methodology that the average red value of captured image sequences is updated with time, heart rate measured in first several time frames generally exceed the actual value. As the time goes by, the measuring accuracy is improved significantly and measured rate is very close to the actual value. Testing was performed during project developed with the Instant Heart Rate application developed by Azumio Inc.

4.3 Emotion Tracker

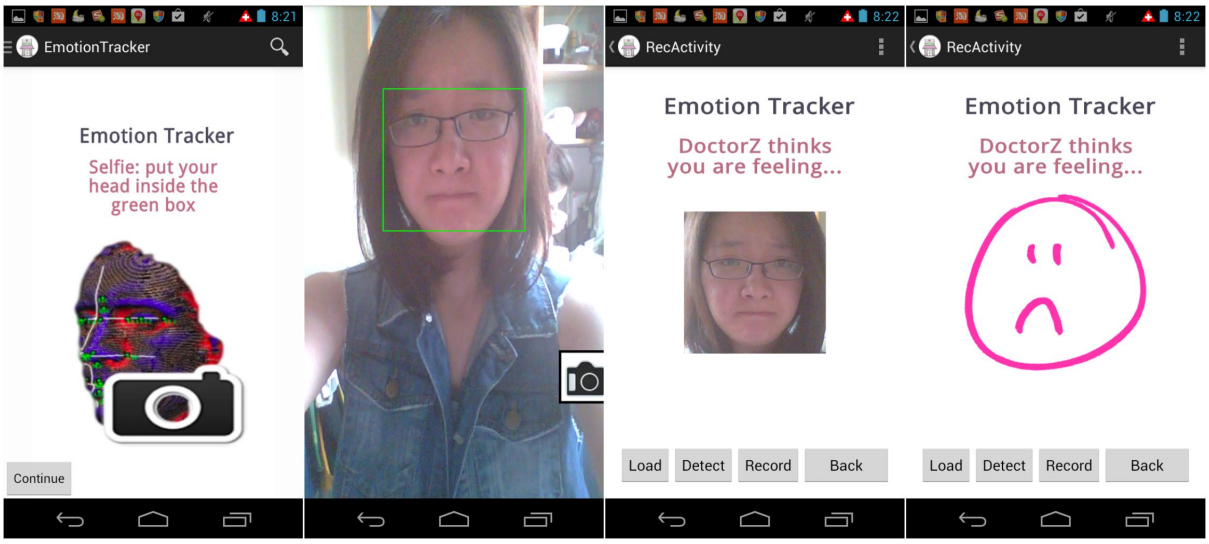


Figure 34. User interface of Emotion Tracker

Upon entering this function, the user would be directed to the camera preview first as shown in Figure 20. Once clicking the capture icon on the right when the face is fully wrapped by the bounding box, the image inside the bounding box would be cropped and displayed. Then the recognition algorithm gets processed in the background if the “Detect” button is pressed. After several seconds, a cartoon face representing the recognized current emotion status is displayed.

Expression classification examples are shown below in figure 35. Generally satisfactory accuracy is achieved, but erroneous recognition occurs due to different lighting conditions and different camera specifications. Improvement will be discussed in chapter 5.

Finally, when the recognition finishes, you can choose to record the result to the database by clicking the “record” button. Emotion status information as well as current date information will be recorded. Establish the daily habit to take simple selfie pictures using this application, without extra work, your emotion status is automatically being monitored by the application over time. On a monthly basis, a chronic distress report would be generated for you. You can check if any alerts have been issued. If so, take precautions accordingly.

```
Apr 6, 2014 8:28:06 PM javaCVPackage.FaceRecognition findNearestNeighbor
INFO: training face 14 is the final best match, confidence -1.4255986
Apr 6, 2014 8:28:06 PM javaCVPackage.FaceRecognition recognizeFileList
INFO: nearest = 2, Truth = 2 (Correct). Confidence = -1.4255986
Apr 6, 2014 8:28:06 PM javaCVPackage.FaceRecognition recognizeFileList
INFO: TOTAL ACCURACY: 100% out of 1 tests.
Apr 6, 2014 8:28:06 PM javaCVPackage.FaceRecognition recognizeFileList
INFO: TOTAL TIME: 13.299915 ms average.
```

```
Apr 6, 2014 8:26:30 PM javaCVPackage.FaceRecognition findNearestNeighbor
INFO: training face 13 is the final best match, confidence -1.3086101
Apr 6, 2014 8:26:30 PM javaCVPackage.FaceRecognition recognizeFileList
INFO: nearest = 1, Truth = 1 (Correct). Confidence = -1.3086101
Apr 6, 2014 8:26:30 PM javaCVPackage.FaceRecognition recognizeFileList
INFO: TOTAL ACCURACY: 100% out of 1 tests.
Apr 6, 2014 8:26:30 PM javaCVPackage.FaceRecognition recognizeFileList
INFO: TOTAL TIME: 14.481737 ms average.
```

```
Apr 19, 2014 8:03:13 AM javaCVPackage.FaceRecognition findNearestNeighbor
INFO: training face 2 is the final best match, confidence -1.0222598
Apr 19, 2014 8:03:13 AM javaCVPackage.FaceRecognition recognizeFileList
INFO: nearest = 2, Truth = 1 (WRONG!). Confidence = -1.0222598
Apr 19, 2014 8:03:13 AM javaCVPackage.FaceRecognition recognizeFileList
INFO: TOTAL ACCURACY: 0% out of 1 tests.
Apr 19, 2014 8:03:13 AM javaCVPackage.FaceRecognition recognizeFileList
INFO: TOTAL TIME: 14.785473 ms average.
```

```
INFO: training face 8 is the final best match, confidence -0.9162779
Apr 19, 2014 8:05:23 AM javaCVPackage.FaceRecognition recognizeFileList
INFO: nearest = 2, Truth = 2 (Correct). Confidence = -0.9162779
Apr 19, 2014 8:05:23 AM javaCVPackage.FaceRecognition recognizeFileList
INFO: TOTAL ACCURACY: 100% out of 1 tests.
Apr 19, 2014 8:05:23 AM javaCVPackage.FaceRecognition recognizeFileList
INFO: TOTAL TIME: 14.968875 ms average.
```

Figure 35. Expression classification examples

Chapter 5. Conclusion & Future Works

5.1 Speech Disorder Checker

With this function, the user can gain a deep knowledge of his/her voice condition. By checking if the generated pitch contour is of the expected shape (rising for interrogative intonation and falling for declarative intonation), the user gets to know if he/she may have prosodic skill impairment. By studying the pitch range, the user can check if his/her vocal range is beyond normal range.

As illustrated in Figure 32, it is observed that effects of environmental noises still exist. The accuracy of cepstrum analysis on pitch detection can be further improved. Concerning the pitch contour generation, more involved curve fitting algorithm such as polynomial regression, Gaussian-Newton algorithms can be used to better illustrate the pitch variations over time.

Currently, the diagnosis criteria are set by general statistics suggested by medical researches. Higher diagnosis accuracy is expected if personal training model could built for each user. With the integration of certain machine learning algorithms, more user-oriented checking mechanism would be possible.

Further, the pitch contour generated for dysprosody detection relates to the work of speech recognition and speech synthesis which are heavily researched in this field and sees wide range of practical applications.

5.2 Heart Rate Monitor

With this function, the user masters his/her heartbeat condition wherever possible. If irregular beating is continuously detected, medical solutions could be delivered timely.

Enhancement could be made to eliminate the over counting of heartbeat during the first several time frames. Besides counting, tracing the red intensity variation's changing frequency is another approach worth implementing, possibly with higher accuracy. After

converting to frequency domain, certain filters could be applied to eliminate the affect of environmental noises, better monitoring result might be achieved.

5.3 Emotion Tracker

With this function, the user records his/her daily emotional information in a more interactive way. Facial expression recognition is conducted every time the user takes a selfie picture. Persistent chronic distress tracking is integrated in camera applications.

Deploying the application to the Google's newly developed cloud platform would provide the application with cloud SQL database for better data management and cloud storage for durable and accessible data maintenance. Better computing performance would also be expected.

On the other hand, as stated by [30] and shown in testing figure 35, Eigenface model is proved to be less robust against environmental lighting variations as compared to Fisherface model. Improvement could be made by implementing the FisherFace model based on the EigenFace model already programmed. Hopefully higher recognition accuracy might be achieved at varying illumination conditions.

References:

[1] Xiaoqing Liu; Lei Zhang; Yadegar, J., "An intelligent multi-modal affect recognition system for persistent and non-invasive personal health monitoring," Personal Indoor and Mobile Radio Communications (PIMRC), 2011 IEEE 22nd International Symposium on , vol., no., pp.2163,2167, 11-14 Sept. 2011

[2] Kennison, Shelia M. (2014). Introduction to language development. Los Angeles: SAGE. ISBN 978-1-4129-9606-8. OCLC 830837502.

[3] Pietrosemoli, Lourdes; Mora, Elsa (April 11–13, 2002). "Dysprosody in Three Patients with Vascular Cerebral Damage". Speech Prosody 2002. Aix-en-Provence, France. pp. 571–4.

[4] Pinto JA, Corso RJ, Guilherme AC, Pinho SR, Nóbrega Mde O (March 2004). "Dysprosody nonassociated with neurological diseases--a case report". J Voice 18 (1): 90–6. doi:10.1016/j.jvoice.2003.07.005. PMID 15070228.

[5] F. Ringeval, J.Demouy, G. Szaszak, M.Chetouani, L.Robel, J.Xavier, D.Cohen and M.Plaza, "Automatic Inonation Recognition for the Prosodic Assessment of Language-Imparied Children," in IEEE Transactions on Audio, Speech, and Language Processing, Vol. 19, No.5, July 2011.

[6] Kempler, Daniel (2005). Neurocognitive Disorders in Aging. Sage Publications. pp. 92–105. ISBN 0-7619-2163-X.

[7] Deirdre D. Michael. (2012, Dec 1). Types of Voice Disorders. [Online]. Available: http://www.lionsvoiceclinic.umn.edu/page3b.htm.

[8] Koichi OMORI, "Diagnosis of Voice Disorders," JMAJ, Vol. 54, No. 4, pp. 248–253, 2011.

[9] Titze, I.R. (1994). Principles of Voice Production, Prentice Hall (currently published by NCVS.org) (pp. 188), ISBN 978-0-13-717893-3.

[10] Baken, R. J. (1987). Clinical Measurement of Speech and Voice. London: Taylor and Francis Ltd. (pp. 177), ISBN 1-5659-3869-0.

[11] Mandel, William J., ed. (1995). Cardiac Arrhythmias: Their Mechanisms, Diagnosis, and Management (3 ed.). Lippincott Williams & Wilkins. ISBN 978-0-397-51185-3.

[12] Jameson, J. N. St C.; Dennis L. Kasper; Harrison, Tinsley Randolph; Braunwald, Eugene; Fauci, Anthony S.; Hauser, Stephen L; Longo, Dan L. (2005). Harrison's principles of internal medicine. New York: McGraw-Hill Medical Publishing Division. ISBN 0-07-140235-7.)

[13] Rahman, M.A.; Barai, A.; Islam, M.A.; Hashem, M.M.A., "Development of a device for remote monitoring of heart rate and body temperature," Computer and Information Technology (ICCIT), 2012 15th International Conference on , vol., no., pp.411,416, 22-24 Dec. 2012

[14] Carlson, Neil (2013). Physiology of Behavior. Pearson. pp. 602–606. ISBN 9780205239399.

[15] Anssi Klapuri and Manuel Davy (2006). Signal processing methods for music transcription. Springer. p. 8. ISBN 978-0-387-30667-4.

[16] Andrew C. Singer (2009) Signal Processing of Discrete-time Signals

[17] TCH DETECTION METHODS REVIEW [Online]. Available: http://ccrma.stanford.edu/~pdelac/154/m154paper.htm

[18] A. Michael Noll, “Cepstrum Pitch Determination,” Journal of the Acoustical Society of America, Vol. 41, No. 2, (February 1967), pp. 293-309.

[19] Ricardo Gutierrez-Osuna, “Cepstral analysis,” [Online]. Available: http://research.cs.tamu.edu/prism/lectures/sp/l9.pdf

[20] Taylor, 2009, ch. 12; Rabiner and Schafer, 2007, ch. 5; Rabiner and Schafer, 1978, ch. 7

[21] Tkalcic, Marko, and Jurij F. Tasic. "Colour spaces: perceptual, historical and applicational background." Eurocon. 2003.

[22] Color Formats. Systementwicklung equasys. [Online]. Available: http://www.equasys.de/colorformat.html

[23] Weishan Zhang, Xin Meng, "A Hybrid Emotion Recognition on Android Smart Phones," in 2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing.

[24] Y. Tian, L. Brown, A. Hampapur, S. Pankanti, A. Senior, and R. Bolle, "Real world real-time automatic recognition of facial expression," in IEEE PETS, Australia, March 2003.

[25] Cootes, Timothy F., Gareth J. Edwards, and Christopher J. Taylor. "Active appearance models." *IEEE Transactions on pattern analysis and machine intelligence* 23.6 (2001): 681-685.

[26] Caifeng Shan; Shaogang Gong; McOwan, Peter W., "Robust facial expression recognition using local binary patterns," Image Processing, 2005. ICIP 2005. IEEE International Conference on , vol.2, no., pp.II,370-3, 11-14 Sept. 2005.

[27] Zhengyou Zhang; Lyons, M.; Schuster, M.; Akamatsu, S., "Comparison between geometry-based and Gabor-wavelets-based facial expression recognition using multi-layer perceptron," Automatic Face and Gesture Recognition, 1998. Proceedings. Third IEEE International Conference on , vol., no., pp.454,459, 14-16 Apr 1998.

[28] I. Essa and A. Pentland. Coding, analysis, interpretation, and recognition of facial expressions. IEEE Transactions on Pattern Analysis and Machine Intelligence, 19(7):757–763, July 1997.

[29] M. Turk and A. Pentland, “Face Recognition Using Eigenfaces,” Proc. IEEE Conf. on Computer Vision and Pattern Recognition, 1991, pp. 586-591.

[30] Belhumeur, P.N.; Hespanha, J.P.; Kriegman, D., "Eigenfaces vs. Fisherfaces: recognition using class specific linear projection," Pattern Analysis and Machine Intelligence, IEEE Transactions on , vol.19, no.7, pp.711,720, Jul 1997.

[31] Alan V. Oppenheim and Ronald W. Schafer, Discrete-Time Signal Processing, Prentice Hall, 2009.

[32] Face Detection using Haar Cascades. [Online]. Available: http://docs.opencv.org/trunk/doc/py_tutorials/py_objdetect/py_face_detection/py_face_detection.html.

[33] Piotr Wendykier. JTransforms. [Online]. Available: https://sites.google.com/site/piotrwendykier/software/jtransforms.

[34] Apache Commons. [Online]. Available: http://commons.apache.org/proper/commons-math/.

[35] OpenCV. [Online]. Available: opencv.org

[36] JavaCV. [Online]. Available: https://code.google.com/p/javacv/

[37] The Japanese Female Facial Expression (JAFFE) Database. [Online]. Available: http://www.kasrl.org/jaffe.html