Samsung R&D Institute Bangladesh Intern Report

Name Fardin Saad

Job Description Intern

Group Mobile Application 2 Group

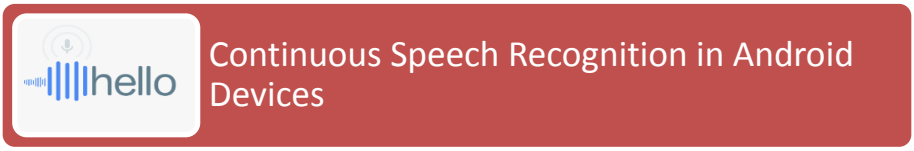
INTRODUCTION:

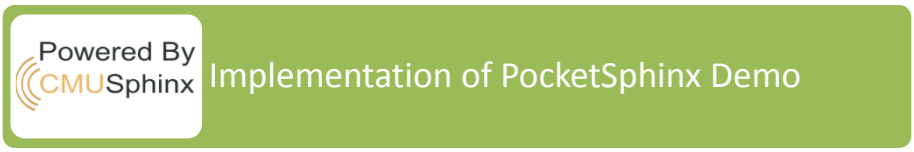
From 18th November, 2018 I have been given the opportunity from Islamic University of Technology (IUT) to participate in a 2 month long Industrial Training Course at Samsung Research and Development Institute, Bangladesh (SRBD). For the past two months I have been assigned to a team in Mobile Application 2 Group led by Md. Mahmud Muntakim Khan. I have completed my 2 month course at SRBD under the supervision of M. Shaykat Shuva and Faisal Khan along with the guidance of Maruf Ahmed Mridul and Arnab Sen Sharma. Throughout my time at SRBD, I have worked with various modules of certain projects to gain valuable knowledge on Android Studio and extensive research to debug certain critical issues in hand. My mentors have guided me every step of the way to be able to learn and help with project related work.

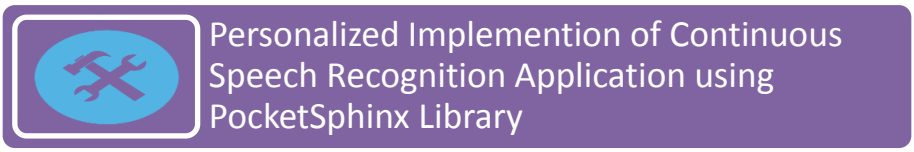
WORK DESCRIPTION:

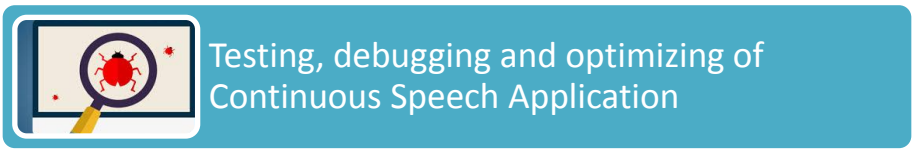
During my stay at SAMSUNG Mobile Application Group 2 I started learning and working on Android Studio and gradually progressed into accomplishing different tasks on that platform. The project that I worked was a Robot project. The modules that I worked on for this project included Android Studio and comprehensible Research on various issues and topics. Those modules are briefly explained in the following points divided into few main categories.

Continuous Speech Recognition









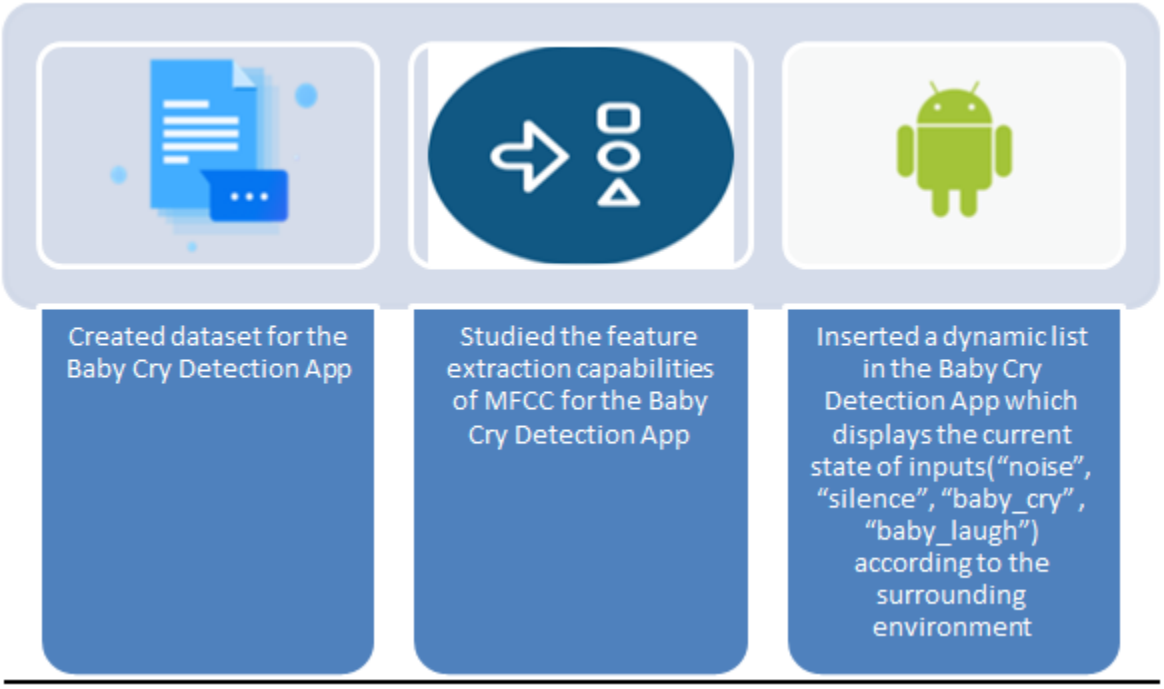
Application development

Built a Video Recorder App

Resolution, Aspect ratio control

Voice recognition / Authentication API

Research work relative to Baby Cry detection



Created dataset for the Baby Cry Detection App

Studied the feature extraction capabilities of MFCC for the Baby Cry Detection App

Inserted a dynamic list in the Baby Cry Detection App which displays the current state of inputs("noise", "silence", "baby_cry", "baby_laugh") according to the surrounding environment

- **Continuous Speech Recognition in Android Device:**

While studying Continuous Speech Recognition I came to acknowledge the existence of CMUSphinx and its various models which were already in place to carry out Continuous Speech Detection/Recognition. I thoroughly studied PocketSphinx which was an upgraded version of CMUSphinx.
Generally for Continuous Speech to start listening to any Speech a Recognizer Speech class is required which is an inbuilt class of Android Studio. This class then with the help of an interface class known as Recognition Listener times and processes the procedure of taking audio inputs. By manipulating both these class it is possible to acquire Continuous Speech. The Recognizer Speech class has a function which initiates the listening process and the Recognition Listener class has a function which restarts the process again thus making it continuous.

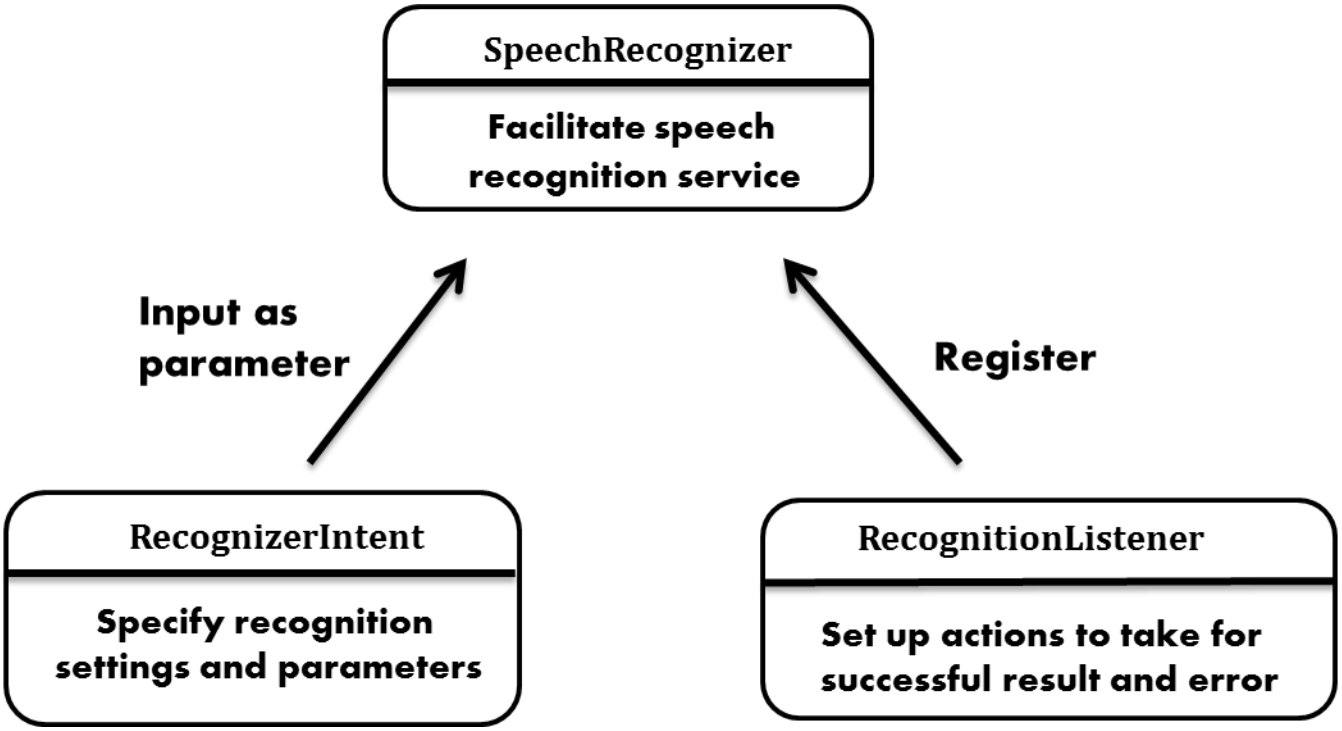
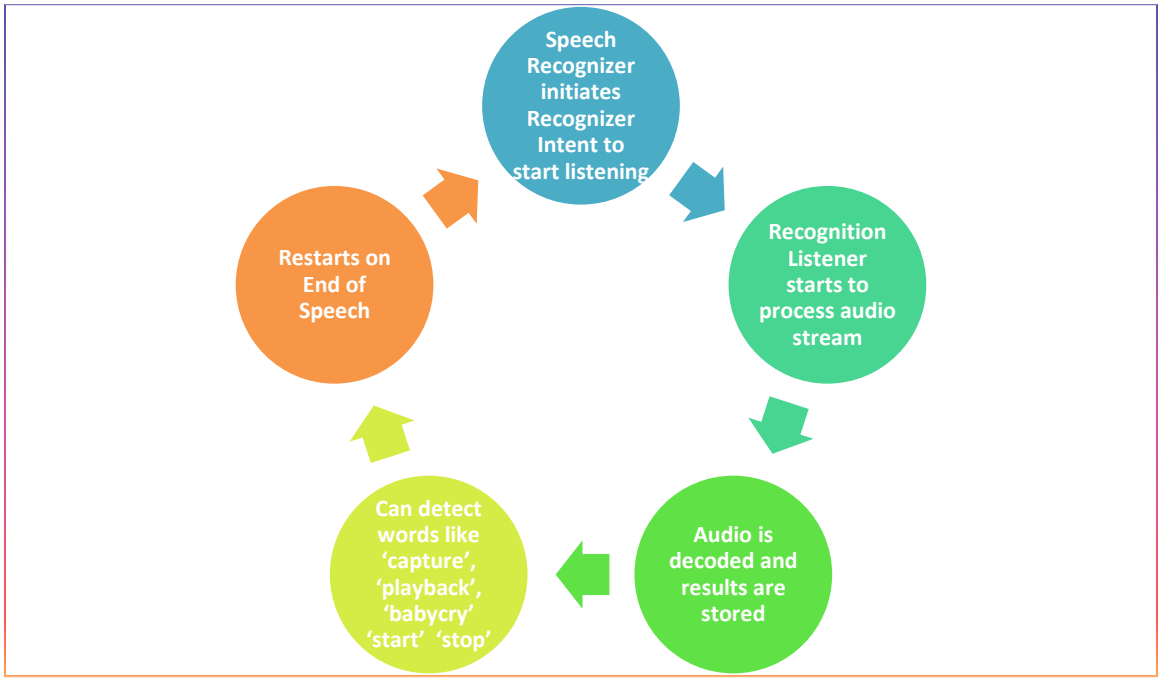


Fig1: Relationship between SpeechRecognizer, RecognizerIntent and RecognitionListener

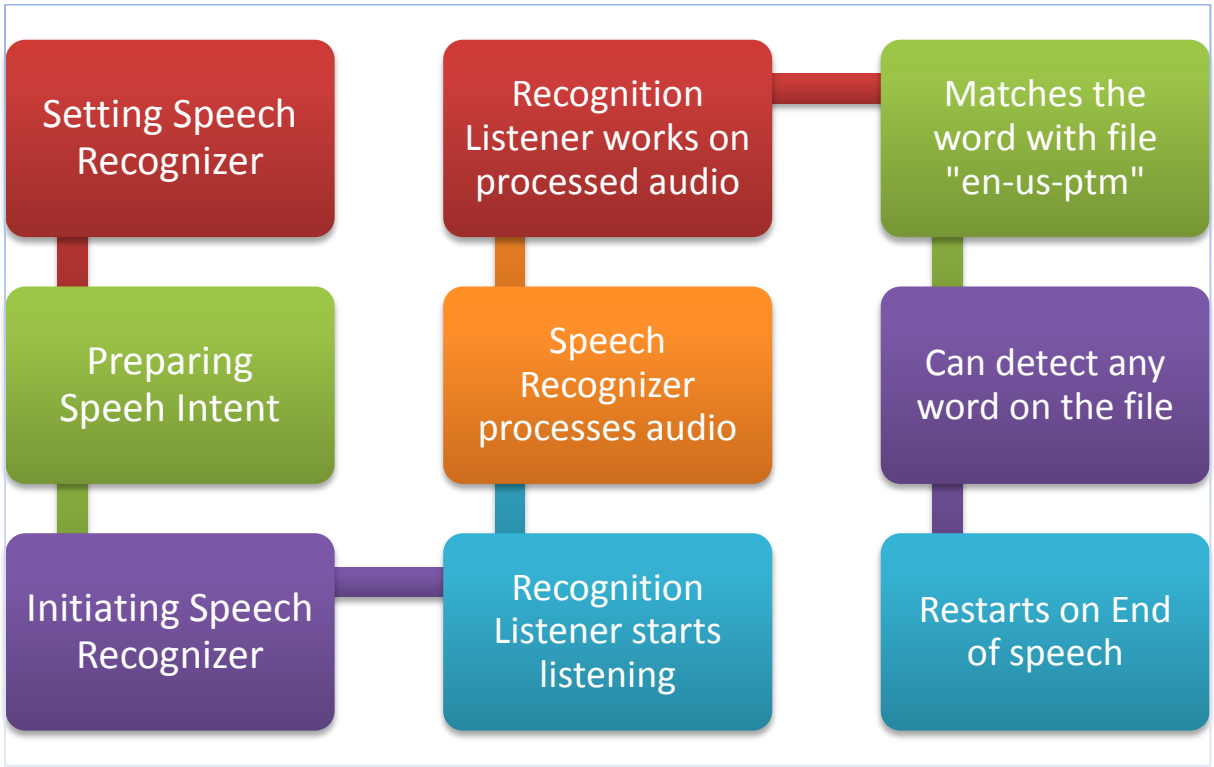
- **Implementation of PocketSphinx Demo:**

I used the PocketSphinx demo at first hand to learn how to take Continuous Speech as input. It used the interface class Recognition Listener to restart the recording process. I further studied how the audio stream is recorded, decoded and detected. The audio stream is processed by the Speech Recognizer class. It processes the audio stream and gives it a pathscore along with an utterance id. The output is then saved in the hypothesis class. From the hypothesis class we can derive the actual words that were spoken and match it with given dictionary in the PocketSphinx Library. We can train the Acoustic model of the PocketSphinx Library to get more accuracy by model adaptation. The string that we find from the hypothesis class can be matched with a key phrase to execute various commands.
In my PocketSphinx demo code I initially setup the speech recognizer intent. Then I start recording and eventually use the recognition listener's on End of Speech method to restart the process. This app could recognize the commands "right", "start", "capture", "stop", "left", "playback", "cry", "baby cry" and many more. I trained my Acoustic model only to recognize these words.
PocketSphinx is very light for which it can continuously take voice commands without any interval. As a result its accuracy is less than 50%. Thus to increase accuracy I tried to adopt the Acoustic Model Adaptation. But in order to adopt this model I needed to have plenty of data to train the model and optimize the parameters.



- **Personalized Implementation of Continuous Speech Recognition Application using PocketSphinx library:**

With the help of PocketSphinx Library I implemented my own Continuous Speech Recognition app. Here instead of recognizing only few words I modified it in such a way so that it could recognize all English words using a file of PocketSphinx known as "en-us-ptm".
My app could detect almost all the words spoken continuously but since I used the PocketSphinx library its accuracy was not satisfactory. I checked the accuracy using a paragraph of 121 words and only 70 words were accurate which gives an accuracy of 57.85%. This accuracy percent with context to the project I'm doing was very low. In order to increase accuracy I needed to train my Acoustic model. But as stated before, I needed plenty of data to train my model and since I didn't have the time to collect this data which usually requires many days depending on the hours of recording and number of speakers, I started working on the Continuous Speech Recognition App that was given to me by this project's main developer.



- **Testing, Debugging and optimizing of Continuous Speech Recognition Application of the Actual Project:**

The Continuous Speech App that I was provided by this project's main developer used the server to recognize words instead of PocketSphinx. Since it didn't use the PocketSphinx API its accuracy was much better despite of having some limitations. One of its limitation was it wasn't as fast as PocketSphinx since a delay was introduced in order to process the audio stream at End of Speech. It was carried out by keeping a countdown timer and the delay was set to be 100ms seconds long. But the probability of recording audio during that interval was minimal and provided if the speaker spoke during that interval then the App would display Error and ask the speaker to speak again. As a result this will ensure that the audio is not recorded within the interval. I was assigned to test, debug and optimize this app.

I. Testing: For testing I used a paragraph of 117 words. I read out the paragraph naturally as a speaker would have done and found 110 words matched which gives an accuracy of 94.01%. The words that didn't match were caused mainly due to the surrounding noise.
II. Debugging: This app was designed in such a way that it would wait for 5 seconds after recording was initiated to take audio input. If it found no input then it would start again. It also had certain audio commands to trigger certain events. While debugging, I said a command to trigger the display text event. Then I went on saying nothing for 3-4 seconds which resulted in an error known as Recognition Service Busy. Ultimately the app crashed.
III. Optimizing: So with the help of the main developer I found what was causing this error. This was caused due to the insufficient delay time for processing audio input when it found the Recognition Service Busy error. Therefore I optimized it by imposing a delay of 2000ms provided the error found was Recognition Service Busy and for any other error the delay was set to be 100ms. Besides this, I removed the delay that was onEndofSpeech method since it was unnecessary and slowing down the recording process

- **Built a Video Recorder App:**

I built an app that can successfully record and stop recording video with the help of a button along with storing it in my device. For video recording initially I had to thoroughly study the Camera API of Android Studio. In order to build a video recorder for my app I did the following:

Camera HW check	Initially I checked my camera hardware to see if the camera feature is available or not
Camera Instance	Then I accessed my camera by getting an Instance of either my front or back camera.
Camera Preview Class	I implemented a Camera Preview Class that extends Surface View and implements Surface Holder.
Surface and Camera Preview Layout	I implemented a Surface Preview Layout that basically incorporates the Camera Preview class. The Surface Preview Layout is responsible for changing the aspect ratio of the video and the Camera Preview class can be modified to control the resolution of the video.
Live Camera	This Surface Preview Layout presents a live camera to the user before recording is started.
Preparing MediaRecorder	For preparing the MediaRecorder I unlocked and set my camera. Then I set my audio source, video source, preview surface profile to achieve the highest quality by setting the output format, audio encoder and video encoder.
Setting Output File & Preview Display	I set the file in which my video will be stored and after that I set the Preview Display by calling the Camera Preview class.
Start/Stop Recording	The MediaRecorder is finally prepared. Now I can start/stop recording video by using the start/stop button.

My Video Recorder Application has a button for recording video. The button is read as “START RECORDING” and will start recording video once it has been clicked. After pressing the button its text is altered to “STOP RECORDING”. Now, to stop recording video at any time the button should be clicked again and on clicking the button its text will be read as “START RECORDING”. The video which was recorded is automatically saved in the specified location after the stop button has been pressed.

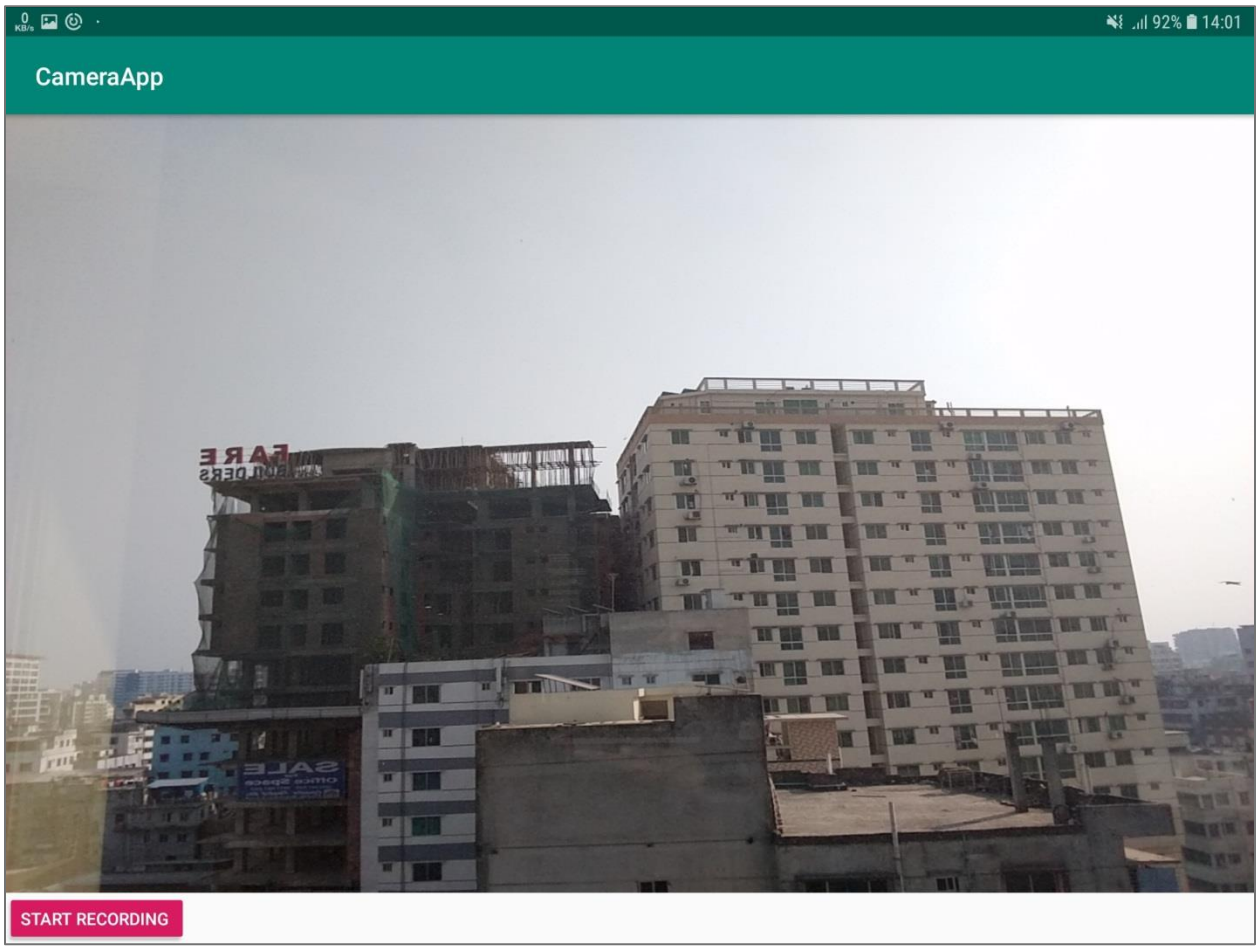


Fig 2.1: Camera Preview before Recording (START RECORDING BUTTON)

This is what the Surface Preview Layout of the Video Recorder Application looks like before recording is started. Initially it presents the live camera to the user and the user can start recording at any time by just clicking the “START RECORDING” button.

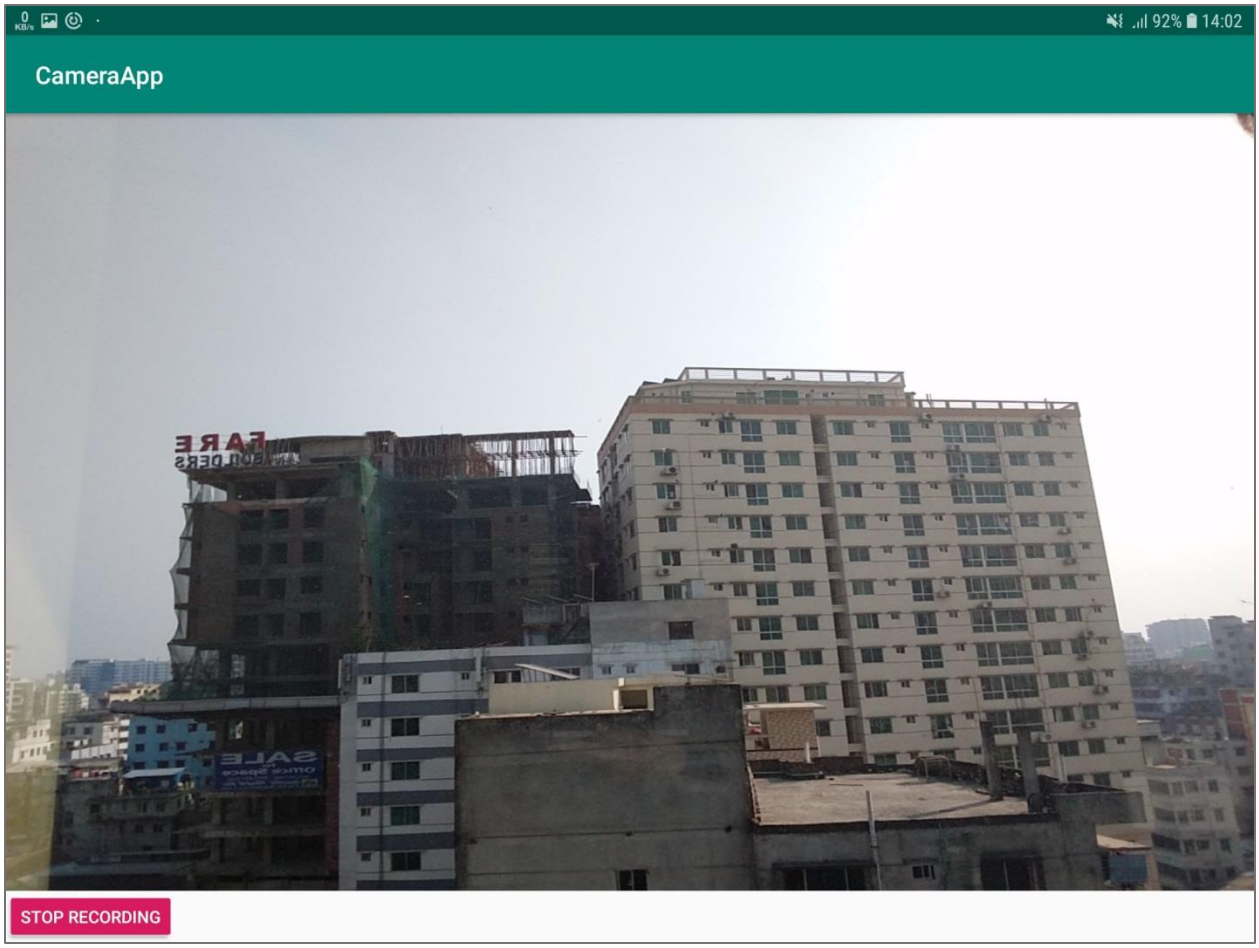


Fig 2.2: Camera Preview while Recording (STOP RECORDING BUTTON)

- **Controlling Camera parameters(Resolution, Aspect Ratio):**

1. Resolution: I modified the resolution of my Video Recorder app in such a way that my Camera Preview Layout has the best quality. In order to do that I acquired the height and width that was best suited for my device. Thus in my Camera Preview class I made a private method that will fetch me the optimal size and width for the device. After that I set the Preview size in my default surfaceChanged method of my Camera Preview class. In Fig 2.1 and 2.2 it can be observed that the resolution of the device has the highest quality settings.

2. Orientation: For different devices the orientation is different. So this parameter needs to be handled otherwise the app will display the default orientation. By display.getRotation () method I changed my

orientation according to the hardware's rotation. From Fig 2.1 and 2.2 it can be observed that the orientation is precisely set as per as the device's requirement.

3. Aspect Ratio: I handled the aspect ratio of my Video Recorded app while preparing the MediaRecorder. After setting the output file and before setting the Preview display I locked the camera and fetched the height and width of the Surface Preview Layout. Now by bringing about changes in the height and width I can control the aspect ratio and set it to according to my requirements.

- **Voice Recognition/Authentication APIs:**

For Voice Recognition/Authentication I researched various Machine Learning and Android based biometric models/APIs for the Robot Project. I was assigned to make a list of such models/APIs. These platforms are listed below:

I. Pitch Detection using Tarsos DSP
II. Voice Vault Biometrics(Embedded Voice Biometrics)
III. Automatic Speaker Recognition System using Transfer Learning
IV. Voice Print Recognition System
V. Microsoft Speaker Recognition API.
VI. Deep Neural Network based Speaker Embedding for end to end speaker verification
VII. Voice Print Recognition System
    - 1:1 Recognition System
    - 1:N Recognition System
VIII. Tensor flow Speech Recognition(Deep Learning
IX. Text Independent Speaker Recognition
X. Voice Biometric Authentication with Twilio
XI. VoiceIt API
XII. Aimbrain API
XIII. JIRA API
XIV. Say-Tec API
XV. Machine Learning for Text Independent Speaker Verification(paper)
XVI. Text Independent Speaker Recognition

I studied the list of models and API's shown above and reported it to the project's main developer for deeper understanding and project progress.

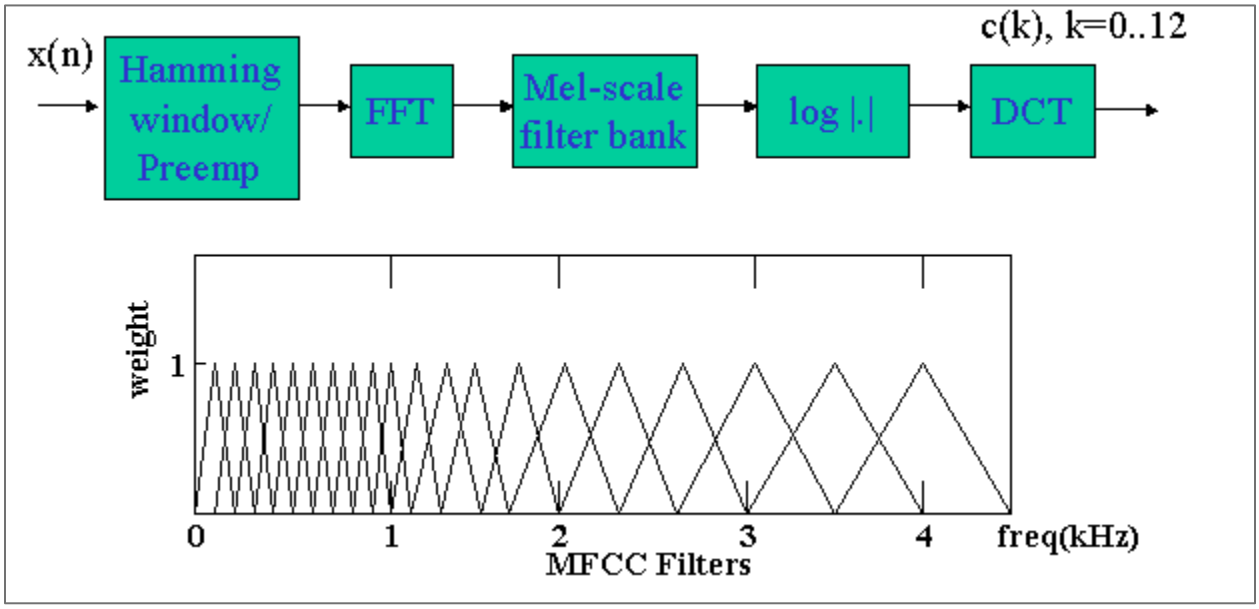
- **Created dataset for the Baby Cry Detection App:**

For the Baby Cry Detection App I was assigned to create a data set for 4 voice inputs such as “baby_cry”, “baby_laugh”, “noise” and “silence”. For each input there were 108 files which contained audio of 5 seconds. The Baby Cry Detection App had three buttons. One was for recording and pausing audio and the other was for saving the processed data of the recorded audio stream as .json files. The last button was the refresh button which upon clicking discards the current data samples in the MFCC vector and enables the users to take new data samples. For creating the data set I turned on the audio file (mp3) and pressed the record button at the same time. After I finished recording I save the data sample, thus creating a dataset. The data obtained during recording was mapped into a MFCC vector in order to form an image. The feature of this image was then used to detect inputs such as “baby_cry”, “baby_laugh”, “noise” and “silence”.

Fig3: Creating Dataset for Baby Cry Detection App for detecting “baby_crying”, “baby_laughing”, “noise” and “silence”.

- **Studied the feature Extraction Capabilities of MFCC for the Baby Cry Detection App:**

MFCC which is widely known as Mel Frequency Cepstral Co-efficient is the representation of the short term power spectrum of a sound based on the linear cosine transformation of a log power spectrum on a non-linear Mel scale frequency. Here cepstrum can be defined as the result of taking the inverse Fourier transform (IFT) of the logarithm of the estimated spectrum of a signal. We use MFC instead of normal cepstrum because it can approximate the human auditory system's response more closely than the linearly spaced frequency bands used in the normal cepstrum. The steps to derive MFCC are:



I. Firstly we apply FFT along with hamming window for each segment to reduce noise.

II. Map the spectrum obtained above onto the Mel scale using triangular overlapping windows which is a further attempt to reduce the redundant information. This is performed by using Mel Filterbank.

III. Once we have the Filterbank energies, we take logarithm of them. This is also motivated by human hearing: we don't hear loudness on a linear scale. Generally to double the perceived volume of a sound we need to put 8 times as much energy into it. This means that large variations in energy may not sound all that different if the sound is loud to begin with. This compression makes our features match more closely what humans actually hear.

IV. Take DCT (discrete cosine transformation) of the list of Mel log powers as if it were a signal. DCT Transforms frequency domain into time domain which is known as quefrequency domain. Feature extraction from this domain becomes easier and faster than LPCC (Linear Prediction Cepstral Coefficients).

V. Keep DCT coefficients 2-13 and we discard the rest because the higher DCT coefficients represent fast changes in the Filterbank energies and it turns out that these fast changes actually degrade Automatic Speech Recognition performance.

VI. Lastly these DCT coefficients are used for features extraction and Speech Recognition.

After studying MFCC and its feature extraction capabilities I gave an informal presentation to this project's main developer to bring further improvement in the Baby Cry Detection App.

- **<u>Inserted a dynamic list in the Baby Cry Detection App:</u>**

In the Baby Cry Detection App I was assigned to create a dynamic list that will be updated automatically according to the received audio streams. The list will display the predicted inputs such as "baby_cry", "baby_laugh", "noise" or "silence". In order to do this firstly, I used a Listview and was able to dynamically add the predicted inputs in the list. But due to some limitations of Listview I was unable to set color to the inputs.
Therefore I implemented the list using TextView and displayed 11 TextView in my app at a time to display the predictions. Besides that, in this attempt I was successful to change the colors of the predictions. For "baby_cry" I set the text color to red, for "baby_laugh" the text color was pink, for "noise" the text color

was green and for “silence” the text color was blue. The list was designed in such a way that it will always display the last 11 predictions.

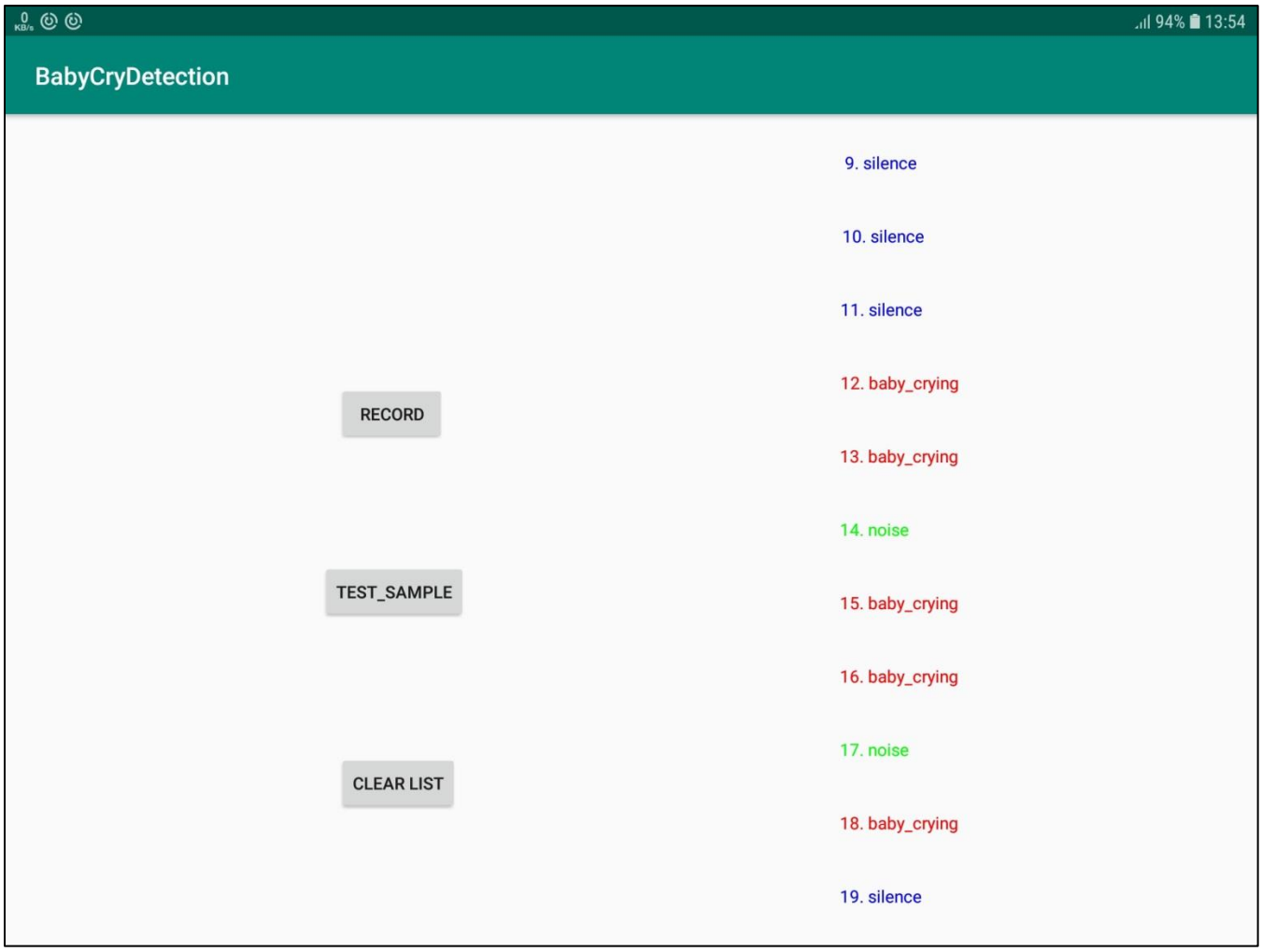


Fig4: Dynamic List showing “baby_crying” in red text, “baby_laughing” in pink text, “noise” in green text and “silence” in blue text.

CONCLUSION:

During my 2 month long internship at Samsung R&D, Bangladesh I learned, experienced and received so much help from my supervisors. My Project supervisors respectfully M. Shaykat Shuva and Faisal Khan consistently guided me and always helped me throughout any issues. Their advice and guidance has been invaluable. My mentors respectfully Arnab Sen Sharma and Maruf Ahmed Mridul never stopped pushing me to reach my goals and took time off their busy schedule to assist me whenever I asked for any help. The tasks that I completed are due in no small part to their support and encouragement. I appreciate and value everything that I have learned from them.

I would also like to show my profound gratitude to Islamic University of Technology (IUT) for giving me this opportunity to participate in this industrial training course. I will try my best to cherish and further improve my knowledge that I gained from here and will be forever grateful for their support and advice.