CSE 700: Independent Study

**Audio Situational Awareness**

**Device with Speaker Recognition**

Supervised by

**Prof. Michael Buckley**

Prepared by

**Sriparna Chakraborty** (UB Person No.: 50314303)

**Upasana Ghosh** (UB Person No.: 50317396)

**Ankita Das** (UB Person No.: 50317491)

**Manisha Biswas** (UB Person No.: 50317483)

**Department of Computer Science**

**University at Buffalo**

Buffalo, NY 14260

**Audio Situational Awareness Device with Speaker Recognition**

# **1. Scope of the Product**

The *Audio Situational Awareness (ASA)* device involves working with a hardware platform i.e the ReSpeaker Core v2.0 which captures speech and extends it to work with the Rev.Ai APIs that provide the speech to text translation feature. The speech is then displayed in the form of text on an Android-based application. The ASA also provides features like text translation, transcript persistence in the form of a file, and transcript sharing via other messaging tools. The ASA application is currently built to support only Android devices. As a future prospect, the application can be extended to support multiple platforms. The current application is built on android studio using java as the programming language.

# **2. Objective**

Our objective was to implement the ‘**Speech to Text Conversion interface using Rev.ai’** instead of Google Translation APIs, as the latter did not work out with the ReSpeaker device. Along with that, we also wanted to have in place the ‘**Microphone Information and Direction of Arrival of the Speech’** to identify a speaker based on the direction of his/her speech and assign a colour to the text message displayed on the mobile application, based on the microphone number that picked up and recorded the speech. Our third objective was to set up the ‘**Bluetooth Interface**’ between the ReSpeaker device and the Android ASA App for a smooth flow of speech in the form of text messages from the ReSpeaker to the Android App. As a future extension of the ASA device, we wanted to introduce an ‘**Artificial Intelligence Module**’, that would uniquely identify a speaker based on the device’s past interactions with the Speaker.

In this report, we have detailed the progress that we have made so far towards the above mentioned objectives and the challenges faced while doing so.

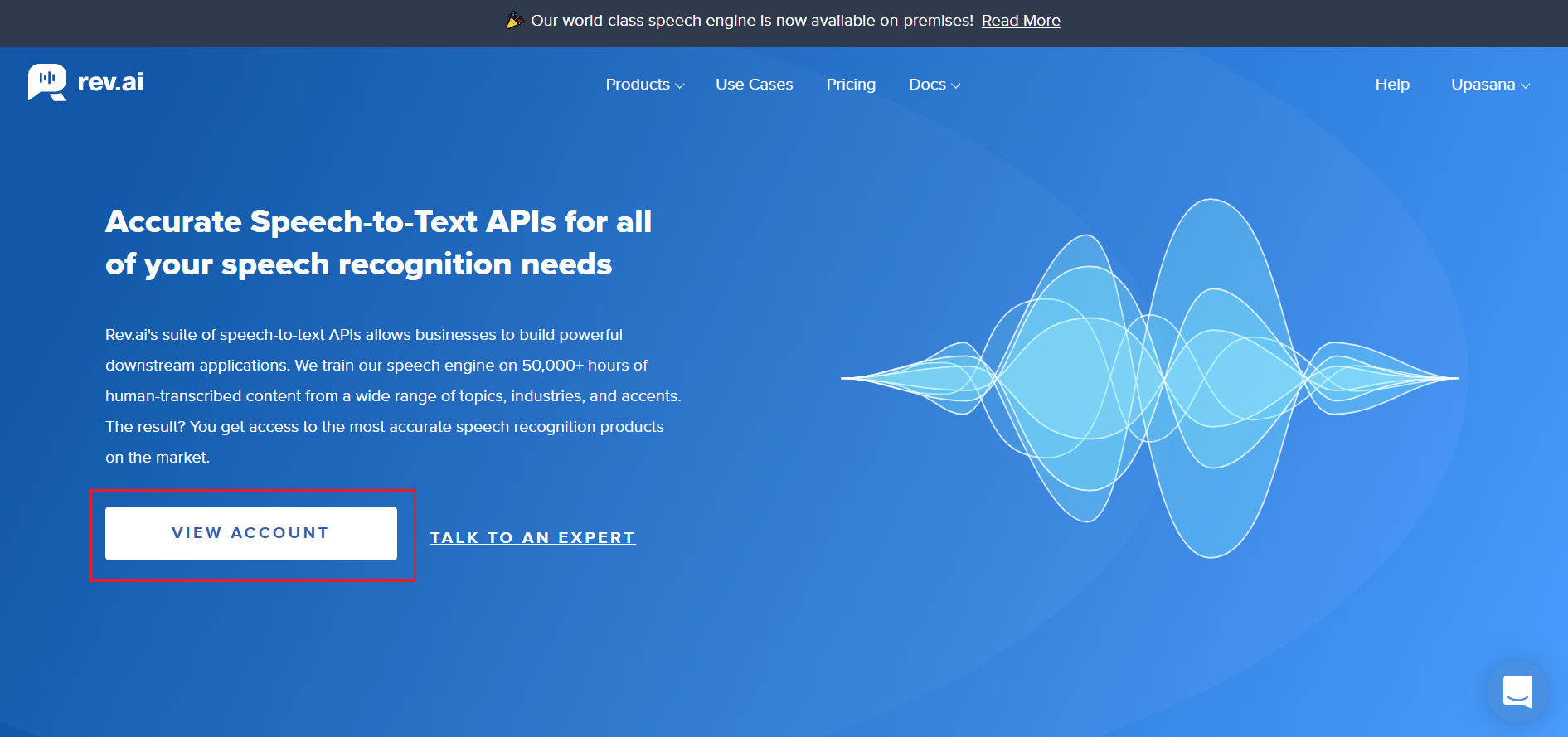
# **3. Speech to Text Conversion using Rev.ai**

## **3.1 Overview**

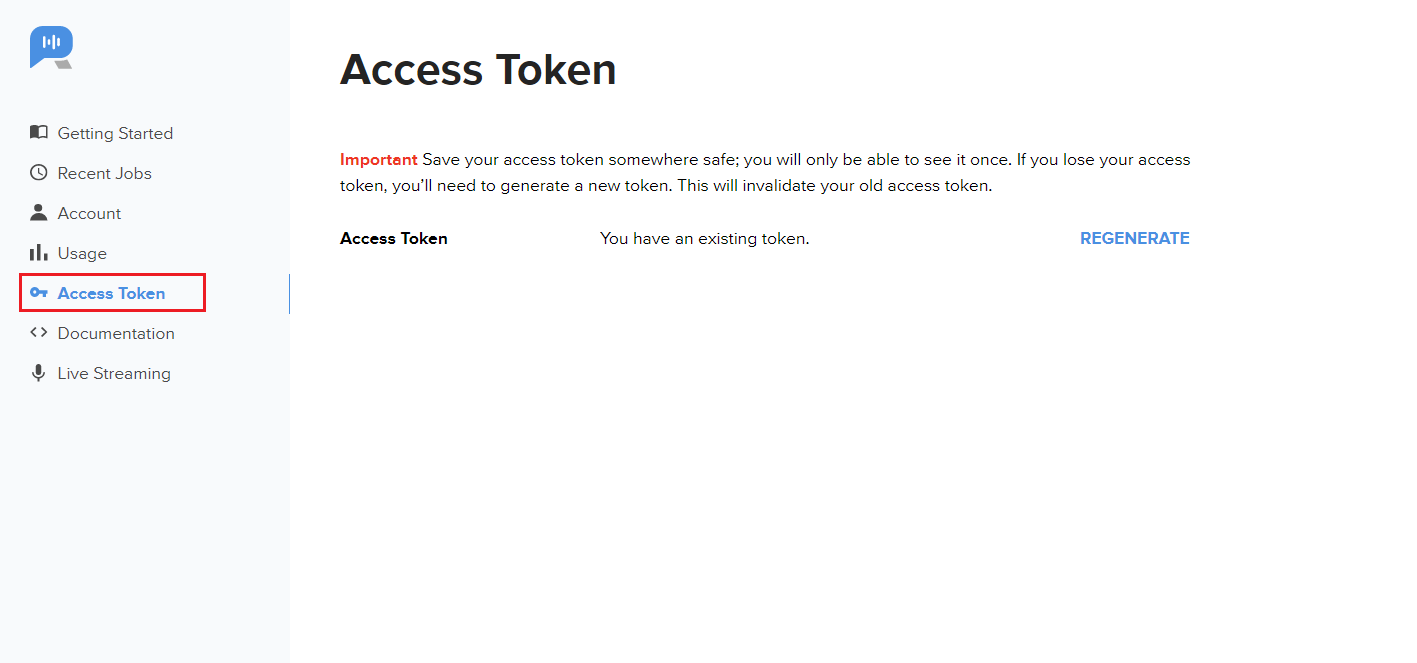
The Streaming Speech-To-Text API from Rev.ai was used to perform the speech to text conversion. The speech recordings were collected from the ReSpeaker board and persisted in a folder ‘data’ in the ‘rev\_ai\_test’ codebase. The 'generator\_streaming.py' holds the code to consume the rev.ai API for converting the speech to the text format and to parse the response JSON into readable chunks of messages that can be sent directly to the ASA Android App.

## **3.2 Implementation**

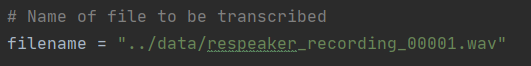
1. To access and consume any of the Rev.ai APIs, we need to have an access token. This access token is passed with every request that we make to the rev.ai’s APIs for authentication. Once the access token is generated, it should be saved securely as it will be displayed to us only once during the token generation. In case if we lose the access token, it can be regenerated but it will not be the same as the old token, so we would need to replace it in all the API calls.
2. To generate the access token, we need to create an account with Rev.ai. Once the account is created, we can log into our account and click on ‘View Account’.



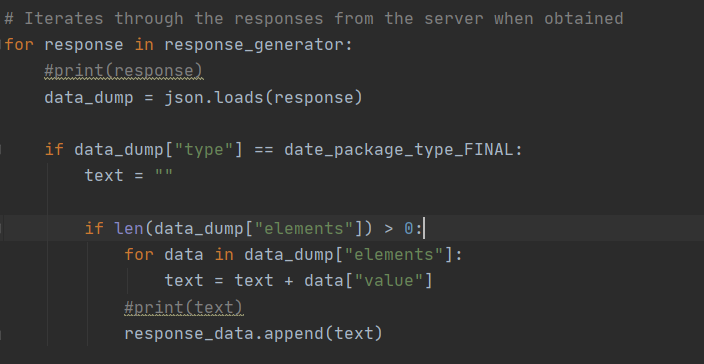
The ‘Access Token’ tab in the left-hand navigation panel will lead us to the screen, where we can generate the access token.



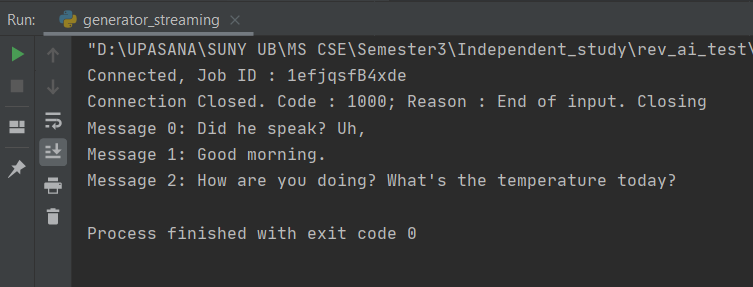
1. For this project, we have consumed the following two APIs from rev.ai -
   1. **Streaming speech-to-text**: For continuous conversion and streaming of the speech, recorded by the ReSpeaker to text messages that can be passed on to the ASA App.
   2. **Custom Vocabulary**: For the streaming API to recognize custom or non-dictionary words (like ‘ReSpeaker’) while converting the speech to text.
2. The ‘**generator\_streaming.py**’ file contains the code for continuous streaming of the speech recordings to text. We pass the access token and the speech recording with the request and get the response as a stream of JSON messages.
3. For this code base, we have persisted the recordings in a folder named ‘data’.



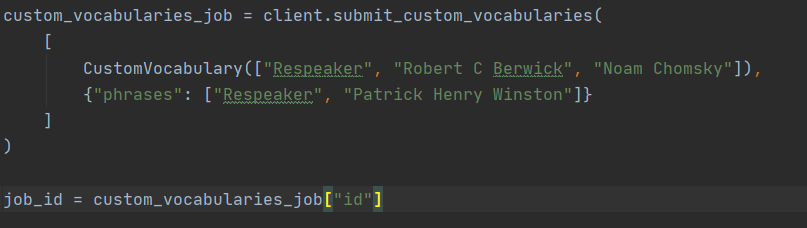
1. The response JSON stream is parsed and only the data packages labelled as ‘data\_package\_type\_FINAL’ are considered and collected for further processing.



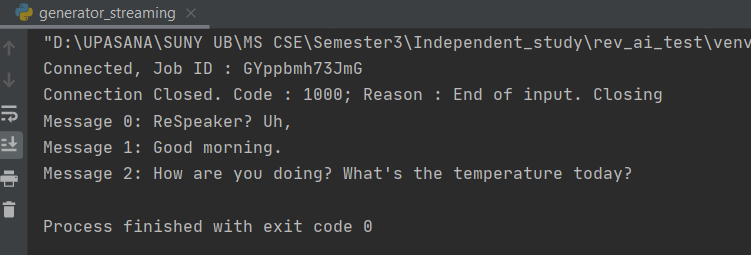
1. Once the processing is done, we get the following parsed and formatted JSON responses



1. The ‘**custom\_vocabularies.py**’ contains the code for accessing the Custom Vocabulary API. We create a list of custom or non-dictionary words and pass it onto the API as a part of the request payload along with the access token. The **‘job\_id’** that is received as the response needs to be passed as a part of the request payload for the streaming API, for the speech-to-text convertor to identify the custom words.



1. Once the job\_id is passed with the payload in the Streaming API, the speech-to-text convertor identifies ‘ReSpeaker’ in the recorded speeches and the parsed messages look as below:



## **3.3 Resources**

We referred the following documentations for implementing and consuming the Rev.ai APIs:

* <https://www.rev.ai/docs/streaming#section/Overview>
* <https://www.rev.ai/docs/streaming#section/Examples/Python-SDK>
* <https://www.rev.ai/docs#section/Quick-Start/Get-your-Access-Token>
* <https://github.com/revdotcom/revai-python-sdk>

# **4. Microphone Information and Direction of Arrival of the Speech**

To get the microphone information i.e the index number of each mic channel we looked into three libraries:

**4.1 Mycroft AI**

Mycroft is a free and open-source software voice assistant that uses a natural language user interface. We can easily build up a personal assistant using Mycroft AI with respeaker core v2. Using Mycroft also helps us to get the index number of unique six mic arrays.

**4.1.1 Challenges**

Mycroft AI provides similar features to Rev.ai but as we decided to move forward with Rev.ai providing more flexibility and free credits. So, we were not able to consider it for development.

**4.1.2 Resources**

Below are the links to the resources which we followed to built the personal assistant

* <https://www.j1nx.nl/project-1/diy-home-personal-ai-assistant-installing-configuration-part-4/>

**4.2 ODAS Library:**

ODAS is a library dedicated to perform sound source localization, tracking, separation and post filtering.ODAS is coded entirely in C, for more portability and is optimized to run easily on low-cost embedded hardware.

**4.2.1 Challenges**

As the library is coded entirely in C, we get through initial complications to implement it. But we have worked on the demo code available on the website but still was not able to get the unique index number of mic arrays using which we could uniquely identify the mics and assign different colors to them to show different speakers in the front-end of Android Application.

**4.2.2 Resources**

Below are the links to the resources which we followed to set up the library.

* [https://github.com/introlab/oda](https://github.com/introlab/odas)s

**4.3 Librespaker**

The librespeaker is an audio processing library which can perform noise suppression, direction of arrival calculation, beamforming, hotword searching. It reads the microphone stream from linux sound server, e.g. PulseAudio. It exposes a few APIs which enable users to get indicated when the hotword is said and the processed microphone data in PCM format, which then can be sent to cloud services like Alexa or Google Cloud Platform for further processing. Audio processing chain consists of several nodes and also has a specific node for Direction of Arrival which is:

[**respeaker::DirectionManagerNode**](http://respeaker.io/librespeaker_doc/classrespeaker_1_1DirectionManagerNode.html) - defines an interface of getting DoA result and setting direction.

**4.3.1 Challenges**

The library is basically built on top of C and C++ programming. So, it was difficult to implement the classes and nodes to automate the entire communication flow from respeaker to the android application.

**4.3.2 Resources**

Below are the resources which we looked into to implement the library:

* <http://respeaker.io/librespeaker_doc/>
* [https://github.com/respeaker/respeaker](https://github.com/respeaker/respeakerd)

# 

# 

# 

# **5. Bluetooth Interface**

### **5.1 Overview**

The end user of this product will have an android application that will have the capability to connect with the ReSpeaker via Bluetooth. Once connected, the user will have the capability to control the ReSpeaker via the application to get the speech data via bluetooth.

### **5.2 Challenges**

1. We tried to establish the bluetooth connection using Android Bluetooth APIs(BluetoothA2dp class). BluetoothA2dp profile defines how high quality audio can be streamed from one device to another over a bluetooth connection. Android provides the BluetoothA2dp class which is a proxy for controlling the Bluetooth A2DP Service.
2. As a second alternative, we tried emulating a Bluetooth android application using Android Studio and an Arduino to toggle an LED and send data back-and-forth. But not having proper support for ReSpeaker hardware devices leads us to explore other options also.
3. iBeacon was another option we researched for. Though it is actually built for iOS applications, it can be used for android also. Under Android 5, there are only a few devices that support iBeacon and two devices that support iBeacon does the transmission. But the main problem we faced here is that we didn't find any support or docs feature over the internet, talking about the interface through which ReSpeaker can connect to the iBeacon.

### **5.3 Resources**

* <https://www.pubnub.com/blog/building-android-beacon-android-ibeacon-tutorial-overview/>
* <http://mcuhq.com/27/simple-android-bluetooth-application-with-arduino-example>

# **6. Artificial Intelligence Module**

### **6.1 Research and Background Work**

As an extension to this project, we worked on implementing an AI module for speaker recognition. We went through a few papers and related articles on Speaker Recognition as listed below:

1. For Audio Source Separation,we are referring the following paper: ([**http://ijcert.org/ems/ijcert\_papers/V3I1103.pdf**](http://ijcert.org/ems/ijcert_papers/V3I1103.pdf))
2. For Speech recognition analysis using Spectrograms: [**https://towardsdatascience.com/speech-recognition-analysis-f03ff9ce78e9**](https://towardsdatascience.com/speech-recognition-analysis-f03ff9ce78e9)
3. We found this interesting project by Mozilla -DeepSpeech: Mozilla’s project DeepSpeech([**https://github.com/mozilla/DeepSpeech**](https://github.com/mozilla/DeepSpeech)) is an open-source Speech-to-text engine that uses a model trained by machine learning techniques based on Baidu’s DeepSpeech research paper ([**https://arxiv.org/pdf/1412.5567.pdf**](https://arxiv.org/pdf/1412.5567.pdf)).
4. Speaker recognition using machine learning techniques: This is a comprehensive paper where the authors have extracted **MFCC (Mel Frequency Cepstral Coefficients)** from the audio signals after a few preprocessing steps ( trimming, split and merge, noise reduction, and vocal enhancements) and evaluated the accuracy of various machine learning techniques (SVM, KNN, Random Forest Classifier) to get the best accuracy with random forest to figure out the speaker ([**https://scholarworks.sjsu.edu/cgi/viewcontent.cgi?article=1685&context=etd\_projects**](https://scholarworks.sjsu.edu/cgi/viewcontent.cgi?article=1685&context=etd_projects)).

### **6.2 Implementation**

We implemented a Speaker Recognition module based on Keras libraries. We created a model to classify speakers from the frequency domain representation of speech recordings obtained via Fast Fourier Transform (FFT).

1. **Dataset**

From the Tensorflow library, <**tf.data**> was used to load, preprocess and feed audio streams into the model. We used the Kaggle ‘Speaker Recognition Dataset’ for training and testing purposes. The dataset can be downloaded from the link below:

<https://www.kaggle.com/kongaevans/speaker-recognition-dataset/download>

This dataset contains speeches of prominent leaders like Benjamin Netanyahu, Jens Stoltenberg, Julia Gillard, Margaret Tacher and Nelson Mandela which also represents the folder names. Each audio in the folder is a one-second 16000 sample rate PCM encoded. A folder called background\_noise contains audios that are not speeches but can be found inside around the speaker environment e.g. audience laughing or clapping. This is mixed with speech for training the model to identify the speakers in real-life settings.

1. **Dependencies**
2. **Procedure**

### **6.3 Challenges**

The AI module that we built using the Keras libraries uses a comparatively smaller dataset for which the model could not be trained to be a robust one. So as a future scope, we need to collect more and more live voice samples and build a much larger dataset so that the model is trained well.

### **6.4 Resources**

<https://keras.io/examples/audio/speaker_recognition_using_cnn/>