Título: **Natural voice interactions in Android Apps**

Author: **André Amarante, 89198**

Teacher: **Ilídio C Oliveira**

Date: **27/04/2020**

Index

Introductory Note

This document is the author's android paper assignment for the ICM course and aims to explore the topic "Natural voice interactions in Android Apps", presenting discussion on the matter in a developer-oriented document.

1. Overview of the topic

Many might think human-machine interaction through voice is a recent invention but the first records of it date to 1952 when Bell laboratories developed "Audrey", a machine capable of understanding only numbers. Over the years leading up to the 1990s, improvements were made but a major flaw kept existing: the need for a pause between each word. It was only in the 90s, thanks to the development of faster microprocessors, that speech software became feasible.

A big change came in the new millennium, when Google and eventually Apple joined the party. This two companies became leaders in human-machine voice interaction software, with Google assistant and Siri.

2. Motivation

So, what's the motivation behind voice interaction in Android? For one, I believe it comes from the fact Google is constantly looking for ways to innovate and become even more relevant. But it also comes from the pure necessity of it. A good example is using phones while driving. As we became more settled with the constant use of devices in our day-to-day lives, we started using them in pretty much every daily task. A main reason behind car crashes is the use of phones while driving and voice interaction appears as a solution for phone interaction while still focusing on the road.

We can't discard the role science fiction has as well. Motivation for these developments also comes from famous movies where characters talk to their watches, cars and other devices, getting a level of intelligent interaction that still hasn't been matched in real life.

3. State of the Art and applications

Most android apps that use voice interaction do so using either Google tools and APIs or one of these android libraries (android.speech or android.speech.tts). This is mainly because it's not worth it to design speech recognition algorithms from scratch when giants like Google already offer quick to implement solutions based on their cloud speech processing or android libraries prepared to deal with both text to speech processing and the reverse.

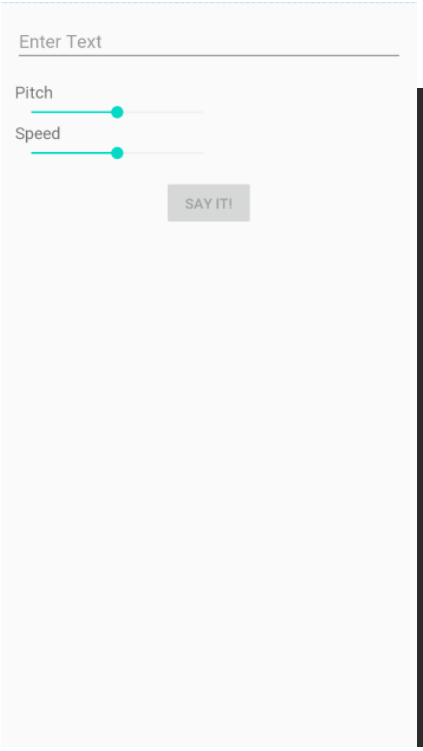
Voice search in Android is mainly used for Google search and Google Assistant interaction. This second one, displays enough "intelligence" to be used for many different interactions, such as telling the weather, searching online for answers to an user's question, giving suggestions on close by highly rated restaurants, and much more. Another big use of this feature is note taking apps, that use speech to text to offer the user a faster way of writing notes and setting reminders. Again, most of these apps use Google tools and APIs to support voice interaction. It's also notable to mention that software like Google Assistant has the power to interact with other apps, seemingly taking away the need for these apps to have embedded voice interaction technology. If you want to listen to music on your android device, just ask Google Assistant to play music or a certain song and you'll even get the option to decide if it's played from Spotify, Youtube or other installed application. If you wish to get directions to a certain place, Google assistant can quickly do that for you. That being said, the need for apps with embedded voice interaction technology still exists and is very valid.

It's then easy to understand that voice technology is indeed growing, with some of the main reasons being the fact it is our main way of communicating with each other (if we are used to it, why not make the machines get used to it), it is faster than typing, it has been implemented as multilingual (Google assistant for example, supports over 30 major languages) and it has become less error prone over the years (Error rates in 2017: Google-4,9%, IBM and Microsoft-5%).

One of the possible uses for voice interaction with devices would be unlocking them safely. In fact, from Android 5.0 to Android 7.1.2, Google implemented this feature in Google Assistant. All one has to do is register his voice model and are good to go. However, Google has removed this feature from Android 8.0 and up, probably because it can't get pass other useful locking techniques like fingerprint, pin or face ID.

4. **Concepts for developers**

The TextToSpeech class from android.speech.tts allows you to process written words into outputted voice, in several different available languages. This functionality is very useful if, for example, you are trying to implement your own language translation app and want to add a "see how it sounds" feature. Here's a simple android app that outputs the inserted text as voice with a user defined pitch and speed, showing how this TextToSpeech class can be used:

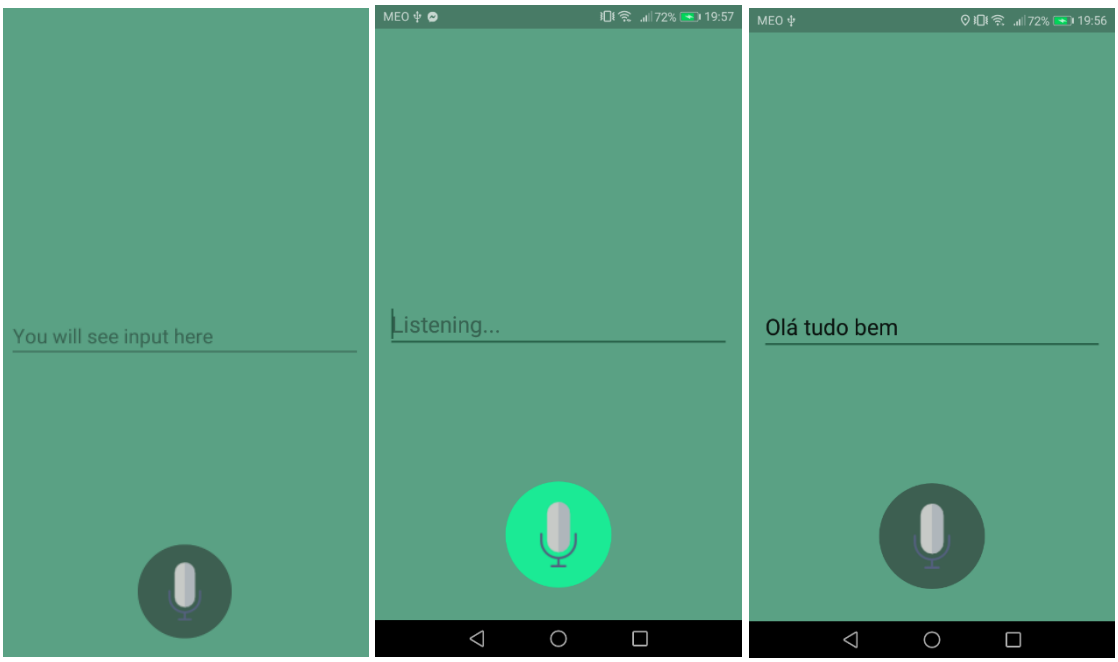



```
mTTS = new TextToSpeech( context: this, new TextToSpeech.OnInitListener() {
    @Override
    public void onInit(int status) {
        if (status == TextToSpeech.SUCCESS) {
            int result = mTTS.setLanguage(Locale.ENGLISH);

            if (result == TextToSpeech.LANG_MISSING_DATA
                    || result == TextToSpeech.LANG_NOT_SUPPORTED) {
                Log.e( tag: "TTS", msg: "Language not supported");
            } else {
                mButtonSpeak.setEnabled(true);
            }
        } else {
            Log.e( tag: "TTS", msg: "Initialization failed");
        }
    }
});

mEditText = findViewById(R.id.edit_text);
mSeekBarPitch = findViewById(R.id.seek_bar_pitch);
mSeekBarSpeed = findViewById(R.id.seek_bar_speed);

mButtonSpeak.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        speak();
```

```
private void speak() {
    String text = mEditText.getText().toString();
    float pitch = (float) mSeekBarPitch.getProgress() / 50;
    if (pitch < 0.1) pitch = 0.1f;
    float speed = (float) mSeekBarSpeed.getProgress() / 50;
    if (speed < 0.1) speed = 0.1f;

    mTTS.setPitch(pitch);
    mTTS.setSpeechRate(speed);

    mTTS.speak(text, TextToSpeech.QUEUE_FLUSH,  params: null);
}

@Override
protected void onDestroy() {
    if (mTTS != null) {
        mTTS.stop();
        mTTS.shutdown();
    }

    super.onDestroy();
}
```

As you can see, we start by defining a new TextToSpeech instance in the onCreate method of our activity. This class receives the context and a TextToSpeech.OnInitListener where we should check the status and in case of success, defined the language for the voice output. Once that's done, we create a speak method to call when the "Say it" button is pressed. In this method, we get the inputted text, the selected pitch and speed values, set them on our TextToSpeech Object and call the speak function from this object to output the voice. No android permissions are needed to use this class.

Now, let's analyze a simple SpeechToText example.



First, we need to define the RECORD_AUDIO permission in the android manifest file.

```
<uses-permission android:name="android.permission.RECORD_AUDIO" />
```

Then, we start by creating a SpeechRecognizer object and a RecognizerIntent in the onCreate method. We set the RecognitionListener and, in this case, only need to fill the onResults method, where we receive the text generated through voice recognition, and display it for the user.

```
checkPermission();

final EditText editText = findViewById(R.id.editText);

final SpeechRecognizer mSpeechRecognizer = SpeechRecognizer.createSpeechRecognizer(this);


final Intent mSpeechRecognizerIntent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
mSpeechRecognizerIntent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,
        RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
mSpeechRecognizerIntent.putExtra(RecognizerIntent.EXTRA_LANGUAGE,
        Locale.getDefault());


mSpeechRecognizer.setRecognitionListener(new RecognitionListener() {

    @Override
    public void onResults(Bundle bundle) {
        //getting all the matches
        ArrayList<String> matches = bundle
                .getStringArrayList(SpeechRecognizer.RESULTS_RECOGNITION);

        //displaying the first match
        if (matches != null)
            editText.setText(matches.get(0));
    }
```

Still in the onCreate method, we set the button to start and stop the SpeechRecognizer listening when the user is pressing it and when he releases it, respectively.

```
findViewById(R.id.button).setOnTouchListener((view, motionEvent) -> {
        switch (motionEvent.getAction()) {
            case MotionEvent.ACTION_UP:
                mSpeechRecognizer.stopListening();
                editText.setHint("You will see input here");
                break;

            case MotionEvent.ACTION_DOWN:
                mSpeechRecognizer.startListening(mSpeechRecognizerIntent);
                editText.setText("");
                editText.setHint("Listening...");
                break;
        }
        return false;
});
```

With this, we can detect what the user says and pass it to text. We can now reuse this logic in our Android apps, utilizing other ways to start and stop the listener, use the text to do online searches, etc. You can even, as I've tested (REPOSITORY HERE), take the generated text and using the TextToSpeech class, have your device "speaking" what you said back to you.

Apart from these libraries, there are Google Cloud Speech APIs, both with free and paid plans, that uses powerful neural network models, recognizes more than 120 languages, can enable voice command-and-control, and even process real-time or prerecorded audio, using machine learning technology. More on these API and how to use it can be found at [1] GOOGLE CLOUD SPEECH API, by clicking the documentation link.

5. Conclusion

In conclusion, voice interaction in Android is mainly monopolized by Google and its specific apps for it, like Google Assistant. However, developers creating Android apps can embed speech-to-text and text-to-speech functionalities in their apps, in a pretty easy way, using them to their own app's needs. It's a growing technology that can still get a lot better, as AI and Machine learning technology grows, leading to easier and less error prone detection of the user's voice, and making natural voice interaction in Android apps a common reality.

6. References

In this document, I mentioned, in my own words, information that I researched for and got from many different sources that I will now mention. I strongly recommend taking a look at these references in order to further understand the topic of Voice Interaction in Android.

1. Google Cloud Speech To Text API - HTTPS://CLOUD.GOOGLE.COM/SPEECH-TO-TEXT
2. A Brief History of Voice Recognition Technology, Chris Kikel - HTTPS://WWW.TOTALVOICETECH.COM/A-BRIEF-HISTORY-OF-VOICE-RECOGNITION-TECHNOLOGY/
3. Android Speech To Text Tutorial, Belal Khan, 11/11/2017- HTTPS://WWW.SIMPLIFIEDCODING.NET/ANDROID-SPEECH-TO-TEXT-TUTORIAL/#ANDROID-SPEECH-TO-TEXT-APP-SOURCE-CODE
4. Android TextToSpeech documentation - HTTPS://DEVELOPER.ANDROID.COM/REFERENCE/ANDROID/SPEECH/TTS/TEXTTOSPEECH
5. Android SpeechRecognizer documentation - HTTPS://DEVELOPER.ANDROID.COM/REFERENCE/ANDROID/SPEECH/SPEECHRECOGNIZER
6. Google Cloud Speech To Text API documentation - HTTPS://CLOUD.GOOGLE.COM/SPEECH-TO-TEXT/DOCS/LIBRARIES
7. Github repository with Android app utilizing both Speech To Text and Text To Speech libraries - HTTPS://GITHUB.COM/ANDREAMARANTE/VOICEINTERACTION