Talk by the Light

Ambient Intelligence
Group A11

92456 - Duarte Bento
92560 - Susana Monteiro

2021/2022

1. Problem

There are situations where it is important to maintain an equilibrium during a conversation, speak but let the others speak in equal amounts, for a debate that is crucial but also in peer to peer conversation you do not want to later find out that you were monopolizing the whole conversation without being aware. Those who know that tend to speak would greatly benefit from a device that could easily indicate, maybe even visually, in real time that he/she is talking more than the other interlocutor. It could also be beneficial to this second interlocutor, who might be a bit shy for example, and can use this device as a reminder to participate more in the conversation.
For debates this could be a “moderator in a box” effectively balancing the time each person speaks with little setup overhead.

Some possible uses for this device could be:

- Just casual use (2 or more people in a conversation)
- Debate moderator
- Project group discussions or presentations
- Brainstorming - in a company usually there is no control. However, in a brainstorming session there is benefit in letting everyone have an opportunity to expose their ideas. Using a device like this could be a non-invasive way to moderate: everyone can look at the light or at the statistic instead of interrupting each other..
- Permanent installation in meeting room

1.1. Assumptions

We assume that the user has the ability to speak and to carry out a conversation, as well as the ability to see and distinguish colors. We also assume that the room is quiet aside from the participants.

In terms of technology, we assume there is at least one Android smartphone with Internet connection, Bluetooth capabilities and a working microphone.

1.2. Requirements

- Identify which user is speaking, possibly by voice recognition using artificial intelligence;
- Display visual cue based on spoken time to act as a moderator;
- Be able to adapt to the inversion of user speaking times: this is when the speaker who spoke the less becomes the one who is speaking more;
- Be easy and simple to set up and also non intrusive for comfortable use in casual conversations;
- Know exactly how long did each user spoke: conversation statistics;
- Create an Android application that is able to connect to the light, to use the smartphone’s microphone for sound input and to display some information about the conversation.

2. Proposed Solution

2.1. Overview

We are proposing a device that can monitor, measure and store how long each interlocutor is speaking. When starting a conversation, each interlocutor would be assigned a color. By identifying who is speaking, we would gradually change the color of a LED to the color of the speaker who has spoken the most until that moment. Additionally, we also want to display the statistics of the conversation on the screen, in real time.

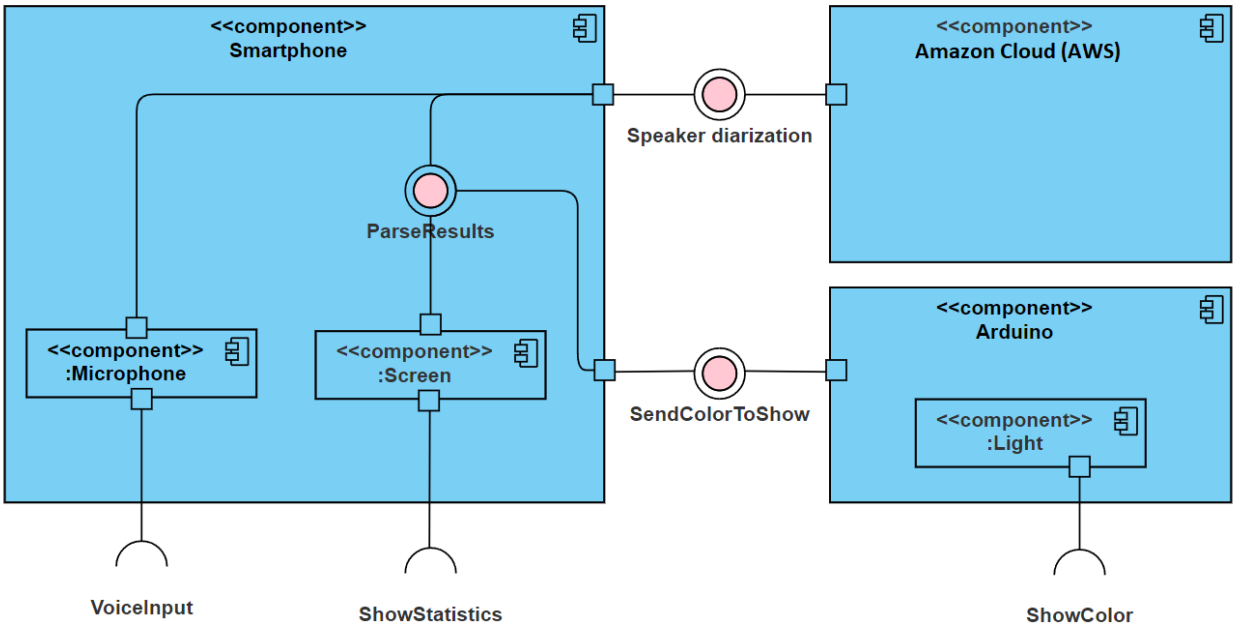


Diagram 1 - Components UML

For example, let's imagine user1 is assigned the color blue and user2 is assigned the color red. User1 starts the conversation and the light is white. After some minutes of continuously speaking, the light is becoming increasingly bluer. Then the user2 starts speaking and the light gradually becomes white again, until he/she has talked more than user1. At this moment, you see the light turning to shades of red.

2.2. Platform

- Arduino IDE to program Arduino nano in C++ programming language.
- Android studio, to develop an android application to read microphone input and send it to the cloud for processing, show the conversation statistics and transcription, to communicate with the light device via Bluetooth and to setup/configure the conversation settings. Language for native development in Android studio will be Kotlin and Java.

2.3. Functionalities to develop

- Regulate light via a microcontroller;
- Connect microcontroller to Bluetooth communication for receiving light values;
- Establish a connection and send messages between the smartphone and the Bluetooth Module;
- Use the smartphone to receive sound input, using its microphone;
- Connect and stream microphone output to Amazon cloud servers;
- Ability to stop the connection to Amazon AWS and restart the conversation with different configurations, without exiting the app;
- Distinguish and identify different interlocutors in a real time conversation;
- Measure and store speaking times;
- Compute a color value based on spoken time to act as visual cue;
- Use a smartphone to display more information and statistics about the current conversation;

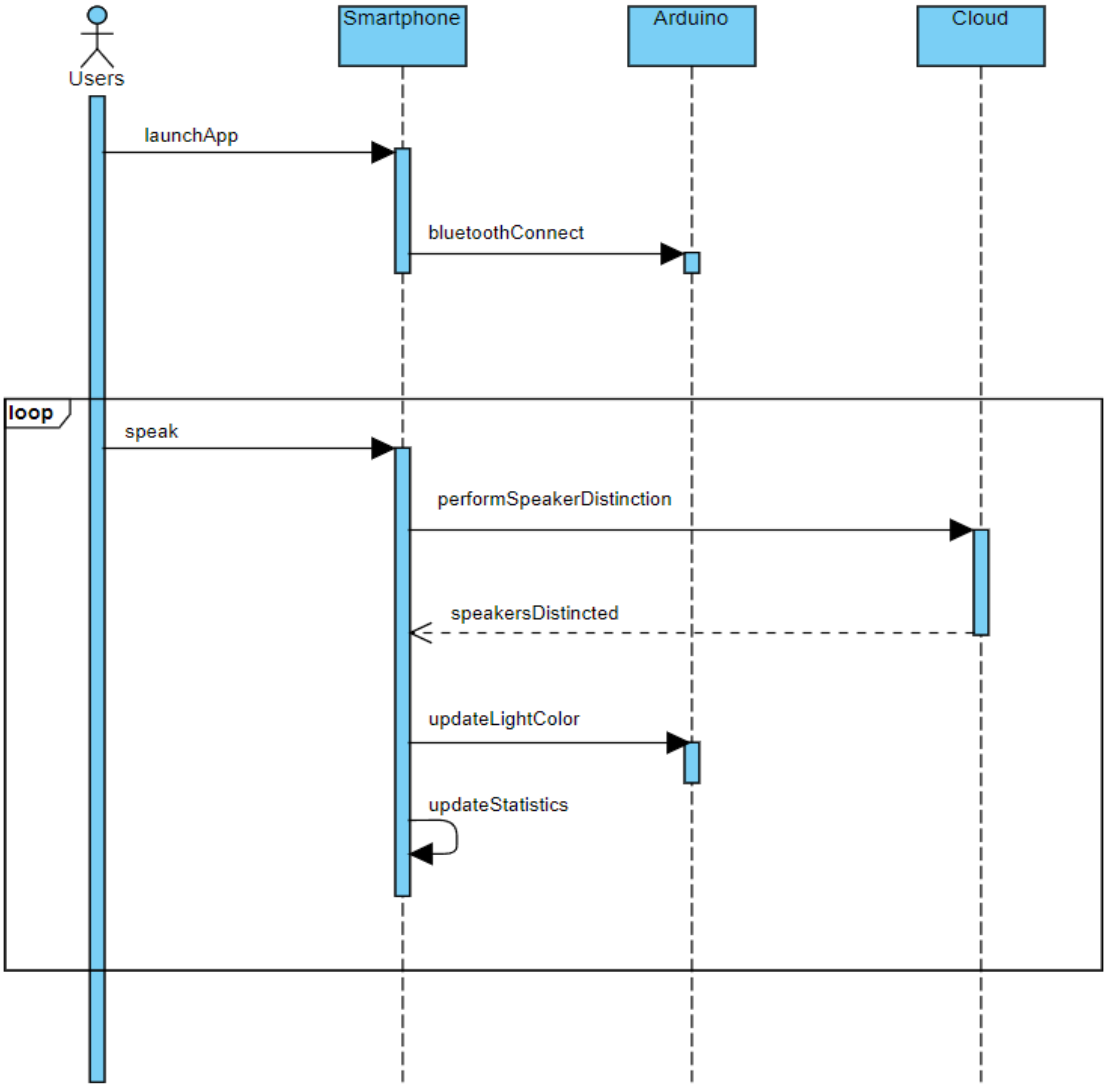


Diagram 2 - Sequence diagram

2.4. Communication channel(s)

The smartphone will communicate with the cloud via the Internet, using AWS SDK for Java 2.x (internally using HTTP/2), and with the light device's microcontroller via Bluetooth.

2.5. Deployment

Android smartphone:

- Equipped with an application developed for our purposes;
- This device will act as microphone to record the conversation;
- Screen to check real time status;
- Establish communication with the microcontroller, for that our device needs Bluetooth capabilities which is standard in most smartphones and a library for Bluetooth serial communication;
- Make a request to a server in the cloud, for that it will need specific libraries from the cloud provider and working internet connection, either through WiFi or cellular data;

Microcontroller in light device:

- Powerful enough to drive and controller the chosen LED;
- Integrate with Bluetooth module;
- Be able to be battery powered;

3. Results

3.1. Light assembly - Microcontroller

We created our own smart light by connecting a RGB LED to a microcontroller that receives update messages defining the color to be displayed.
The communication relies on Bluetooth with our own message protocol with the following format: {#red,#green,#blue}, where #color is a value between 0 and 255.
After receiving a message the microcontroller forwards the color value to the specific PWM (Pulse Width Modulation) pin connected to the corresponding pin (of the same color) in the LED. With the maximum value of 255 for the PWM pin, since the microcontroller we worked with uses 5V internally, the value sent through the output pin would be 5V. To account for that and the different working voltage of the LED we had to use resistors:

Red working voltage is 1.8..2.6V so we used a 120 Ohm resistor;
Green working voltage is 2.9..3.6V so we used a 80 Ohm resistor;
Blue working voltage is 2.9..3.6V so we used a 80 Ohm resistor;

Our Bluetooth module, referenced in the diagram as "HC-06", is basically a Bluetooth to serial module, Bluetooth SPP (Serial Port Protocol), that only supports slave mode, which is enough in our project, sending the messages received through Bluetooth to our microcontroller using the well known 8 n 1 serial protocol, the same used to communicate

with the Arduino serial, this means that for every 8 bits of data sent there is also sent one stop bit but no parity bit. This means that we can connect our Bluetooth module to the digital pins of our microcontroller and use a well known library for Arduino called SoftwareSerial. However, by checking our module datasheet and several guides[1] we realize that even though our module works with a 5V power supply, the input of the module (pin RX) is sensitive and might get damaged by directly using the 5V from our microcontroller. To mitigate this and as suggested by a guide we protected the connection via a voltage divider, as seen in the diagram below, effectively using resistors to reduce the input voltage.

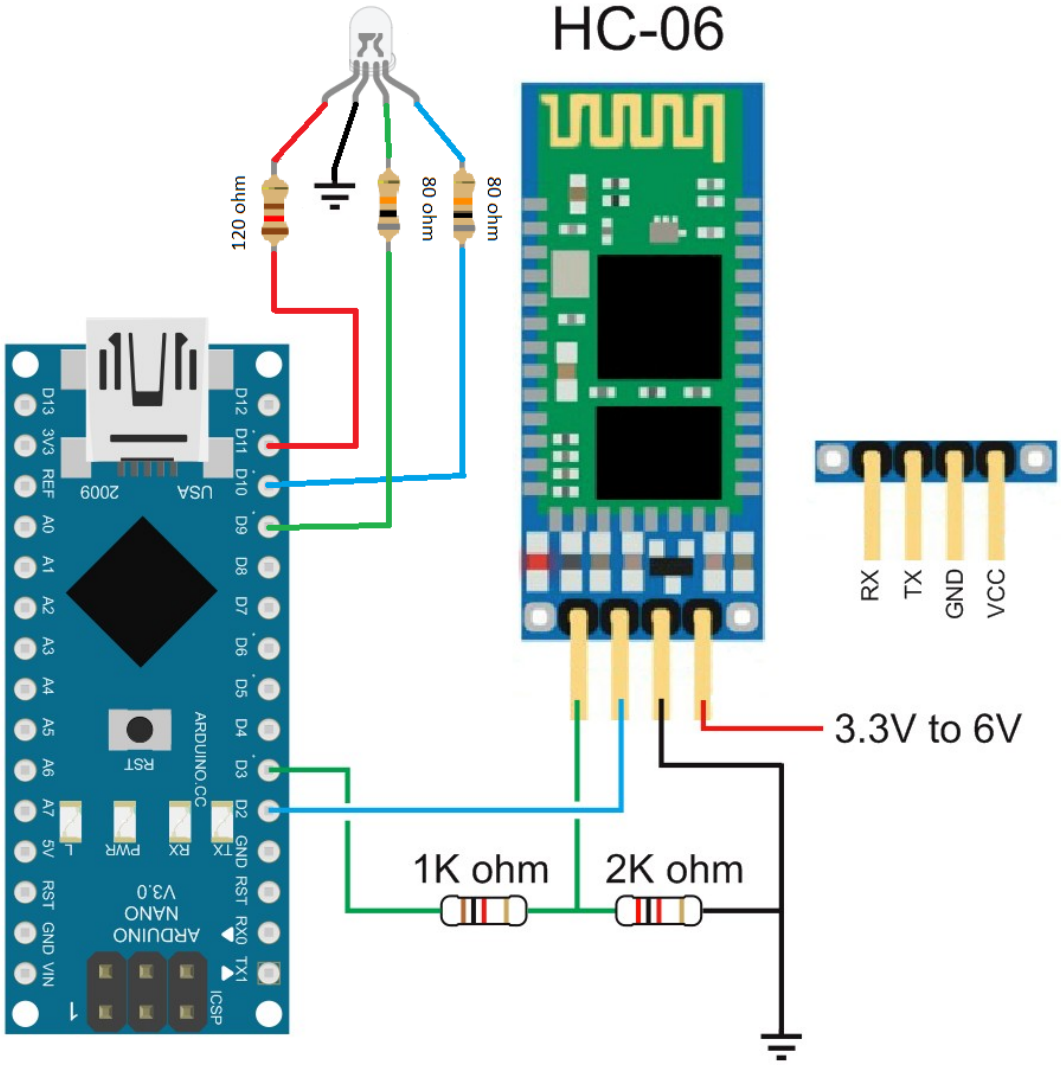
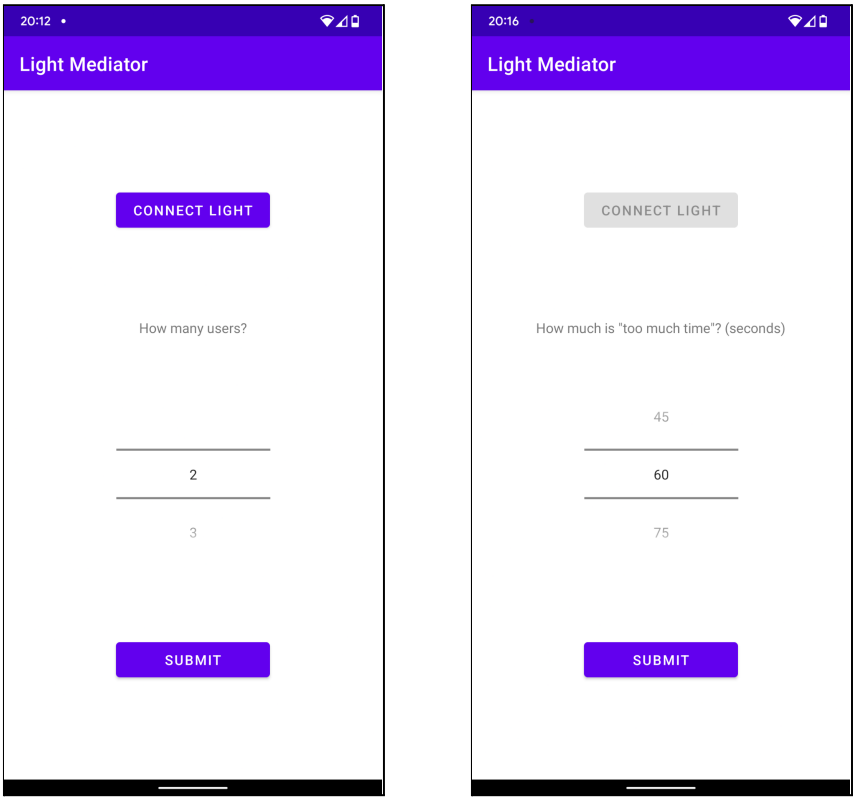


Diagram 3 - Light assembly

After assembling we had a functional Bluetooth smart light whose color we could change directly via any bluetooth serial terminal application. Now we only had to connect to it the device that was going to send the computed color values. We do that in the Android application we developed and is analyzed in the next section.

3.2. Android App

We developed an Android App that works as a bridge between the user, the light and the cloud service used for the speaker diarization.



Images 1 & 2
Initial screen - Before and after connecting to light and choosing users

To start a conversation, an initial screen (*see Images 1 & 2*) is shown to the user, where it is possible to connect the smartphone to the light ("connect light" button). Note that connecting to the light is optional and the application itself will still function properly if you choose not to connect. In that same screen, it is possible to choose the number of speakers and how much time should be considered "too long" for a speaker to hold above the others.
The number of speakers is restricted between 2 and 5, since these are the values recommended by AWS Transcribe, which is the service used to separate each speakers' speech.
The second input we ask for (time cap) is used for calculating the saturation of the light. We calculate the saturation of the light based on the difference between the maximum and minimum spoken times. So, let's suppose the time range chosen is 1 minute. Initially, the spoken times are balanced, the light will be white (255, 255, 255). If the blue speaker talks for 30 seconds (half of the time range), the light will be half in between blue and white, so approximately (127, 127, 255) in RGB. If this blue speaker keeps on speaking until he/she reaches 1 minute, the light is fully blue, so (0, 0, 255) and will have this value until another speaker contributes to the conversation. When this finally happens, the difference between

the maximum and minimum times decreases, and therefore the light becomes whiter and whiter. After some time, this second speaker (whose color we suppose it is red) might reach a total spoken time higher than the first one and the light becomes redder. Thus, if the user chooses a smaller time range, then the light will change colors faster because the difference of times spoken will reach that value faster. We ask the user for this time input so that our application can be used in a variety of situations: longer vs shorter conversations, conversations where balance is crucial vs conversations where it is reasonable if the speakers are not totally balanced, etc.

The second screen of our application (*see Image 3*) shows the color of each user, as well as a text input, where the user can type his/her name if desired (or the name will be "User #" as default for the speaker number #). It is also asked for the user to say the sentence you can see in the image. This is necessary for us to associate the voice of the speaker to its user, so that we can recognize it later. AWS Transcribe only returns the speech separated by speakers (each one identified by a generic speaker tag), but does not support voice training itself, so we had to implement a way to associate a speaker tag to each user ourselves.

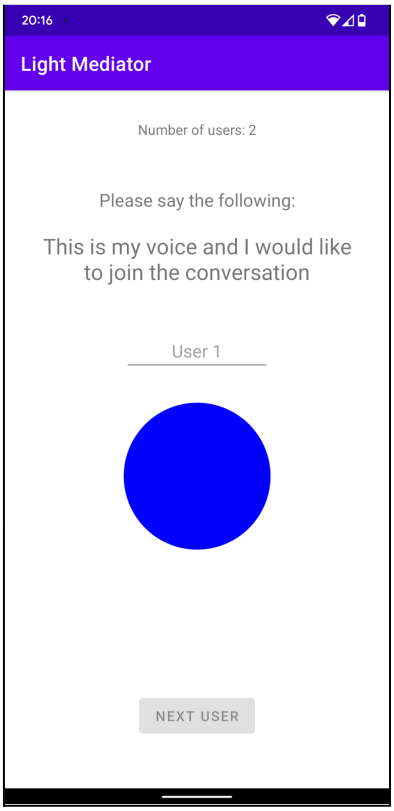


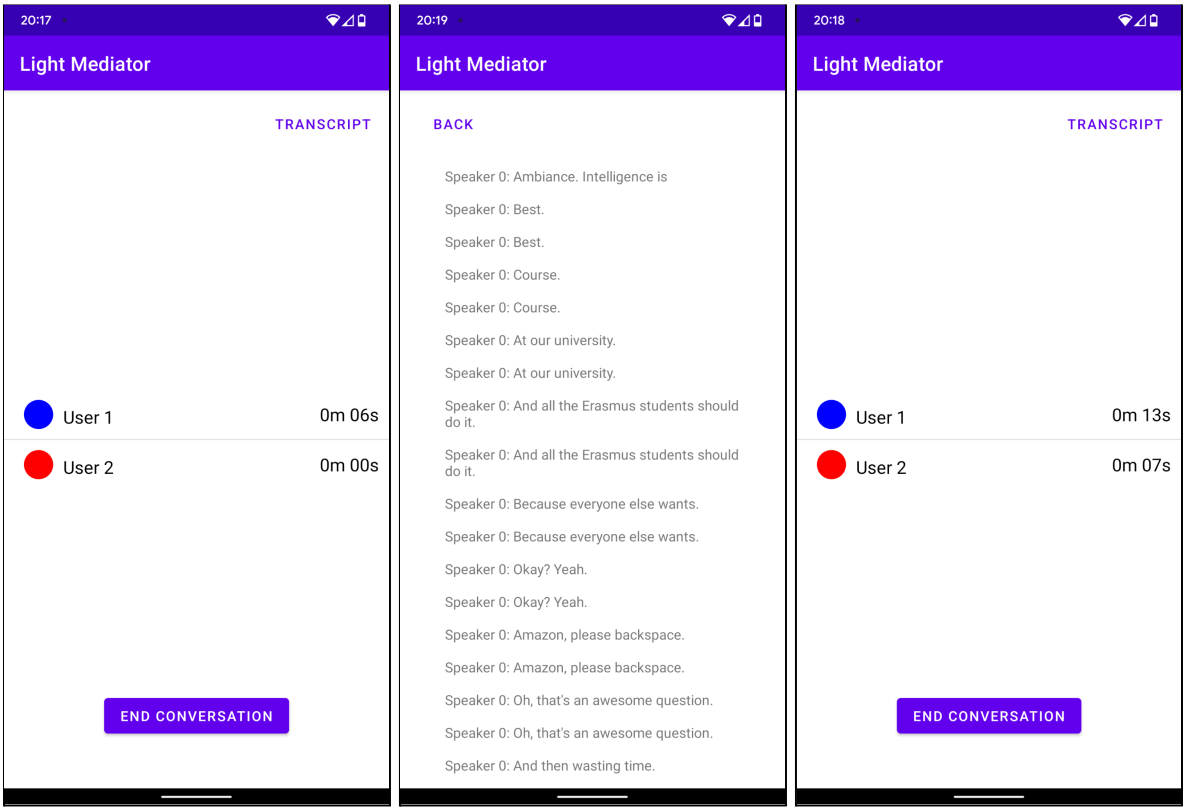
Image 3
Second screen - voice match

This step of the implementation is not trivial. We ask the user to say something specific because we need to know if the response received from AWS Transcribe is actually the one we are looking for and not some other voice recognized from background noise, for instance. Therefore, we need to check if the text received equals the one the user is supposed to have read, which means that the text should be something easy enough to say and easily recognizable, even if the user's pronunciation is not the best. If not, the user could be repeating the text over and over again without being successful in saying the text correctly. The text should also have a significant size - if it was too short, it would be hard for the voice to be learnt by AWS Transcribe. However, the longer the text, the harder it is to get every single word correct. So, how do we create a text that is big enough and at the same time easy to say and be recognized?

Well, our solution was to ask the user to say a rather long text, but only recognize the first few words, so that we can identify the correct response by comparing the first few words and AWS Transcribe still has a longer text to help with the recognition. So, we compare the part "This is my voice", which is rather easy to recognize and the rest is only to increase the text size.

Finally, we also check if the speaker label returned by AWS Transcribe was already attributed to another user, since we do not want 2 users to be recognized as the same.

By the time we get a response from AWS and in case that response is correct we can finally proceed to train the next speaker.



Images 4, 5 & 6
Conversation screen - Live statistics and transcription

Next, the actual conversation can start. The time spoken by each user is displayed on the screen of the device (*see Image 4*) and the light starts to be updated. There is also a button on the upper right corner saying “transcription” which gives us access to the full conversation, separated by speakers. This was used as a debug feature during the development of the project, but could be very helpful for multiple situations where recording the conversation in text might be interesting. Note, however, that the transcription is not 100% accurate (perhaps also because we are not native english speakers) and therefore we left it in the application as an extra feature and because it is interesting to see what is being recognized nonetheless.

Additionally, since we wanted to support more than two speakers we had to take into consideration how to manage conflicts between changing the colors for, let's say three speakers. One idea we had was to change from a user color to another user color, for example from blue to yellow we would travel through the colors in between but if we had a third color it would be harder for the users to differentiate the intermediate colors and even harder for four or five. In the end we chose to implement it in a similar way from what we were doing for only two users, going through white as the origin color and each user’s color as the top speaker.

3.3. Applying our project to a real debate

To methodically test our project we chose to run a video of a recorded debate between two American candidates: Trump (T) and Biden (B). The debate is being moderated by Kristen (K). Here is the video, in case you want to follow the debate:

(C-Span. Second 2020 Presidential Debate between Donald Trump and Joe Biden. https://www.youtube.com/watch?v=bPiofmZGb8o)

We played the video from 25m 20s (the effective start of the debate) until 30m 36s.

While our app was processing the audio from the debate, we manually counted the time each speaker used from those 5 minutes and 16 seconds. Keep in mind this time includes claps and silences, corresponding to changes between speakers, which are being ignored from the time spoken. This means that the total measure both manually and with the application is expected to be lower.

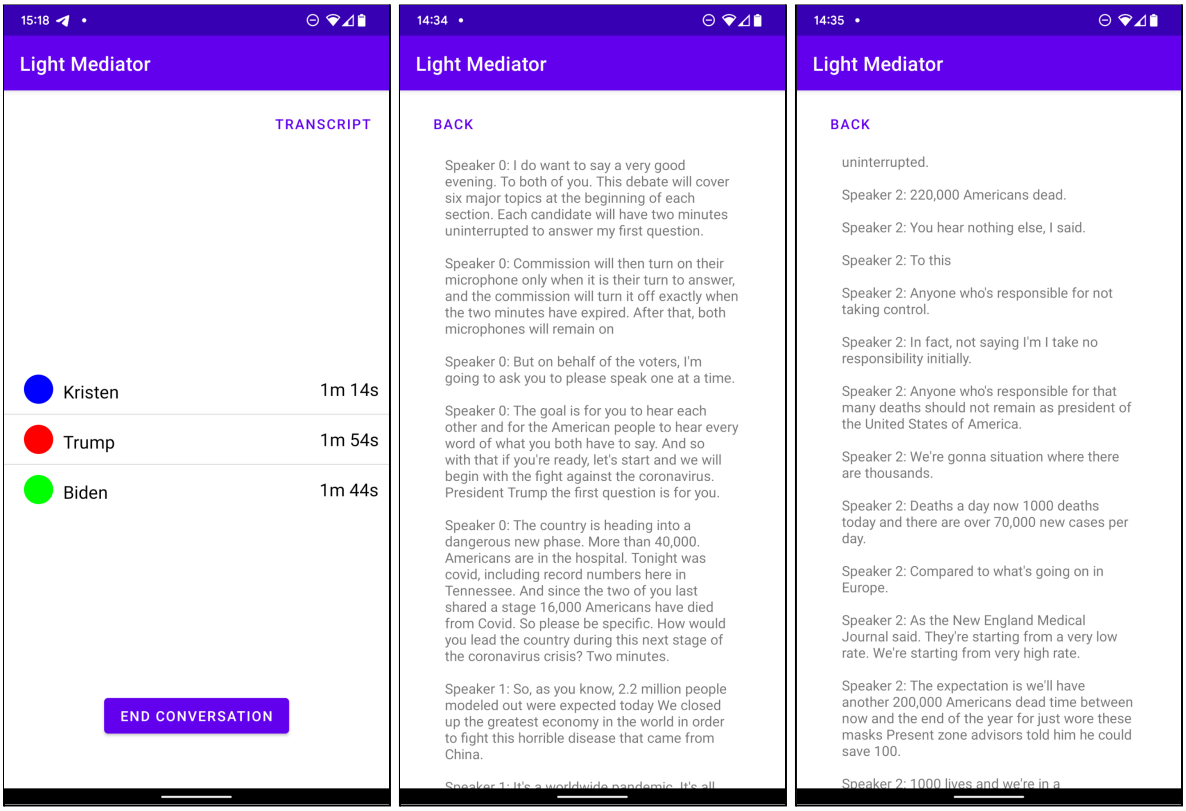
Note that our app's microphone is recording sound coming from the computer speakers, so note that the quality of the recording is not the best. Also, in order to use our app with a pre-recorded video/audio, we need to skip the voice setup step (where we ask the user to say a phrase in order to associate his/her voice to his/her user).

The results were very satisfactory by comparing with the original times, managed by a human being, as we can see in *Table 1*.

	Application	Manually	Difference
Kristen	1m 14s	1m 13s	+1s
Trump	1m 54s	1m 58s	-4s
Biden	1m 44s	1m 48s	-4s
Total Time	4m 52s	4m 59s	-7s

Table 1 - Test results

Below is also a screenshot from the times captured by the app as well as some parts of the transcription, so you can compare whilst watching the video. Note that the transcription was intended as a debug feature, so all of the speakers identified by AWS Transcribe will be displayed with the speaker label generated by it, instead of the names assigned to the users in the application. In this case, Kristen will be "Speaker 0" as she was the first one to speak, then Trump will be "Speaker 1" and Biden "Speaker 2".



Images 7,8 & 9
Screenshot of recorded debate. The spoken time for each participant and a transcript of the conversation.

3.4. Speaker Distinction (choice of Cloud Service)

In order to calculate how much time each speaker spent talking, it was necessary to perform speaker separation. Our initial approach was to use multiple directional microphones (one for each user) and the one receiving the stronger input would be considered the microphone of the user who is speaking.

This solution could have worked, but would require buying multiple microphones, would be extremely susceptible to background noise and it seemed more interesting as well as more effective to try to perform voice recognition to achieve our goal.

Regarding voice recognition there were two options. The first option would be implementing it ourselves (which would require data processing and augmentation, building a neural network and training its model, etc. [2] The second option would be to use a service already available on the Internet that would suit our needs. After some research, we had found multiple services that appeared to be satisfactory.

The first service we tried was “speech-to-text” provided by Google Cloud [3][4][5]. This is an experimental feature of Google (still in Beta) but we still decided to give it a try. We were

able to recognize multiple voices if we sent a pre-recorded audio file. However, in order to maintain context of the users' voices, we needed to use a continuous real time stream. Eventually we also got it to work with streams [6], but the performance was very poor - most words were wrongly recognized or not recognized at all.

Then we looked into Azure, which also provided a similar service to Google. However, this API requires a 7-array mic for real time speaker recognition, which we do not own and is quite expensive.[7] They have other similar services, like speaker diarization from files, which we tried before realizing that we could not implement the only real time speaker recognition service available, and it works much better than Google Cloud's speech-to-text.
Next we looked into AWS Transcribe services. There is a demo application to try their service and it works very well, with a performance similar to Azure's. However, they have a huge lack of documentation, which makes it very hard to work with. We ended up trying other options before actually spending time trying to figure out how to work with AWS Transcribe.

We tried IBM [8], which also has a demo app. It performs better than Google Cloud's speech-to-text, but lacks behind Azure and AWS services. It is also not great on documentation and examples. We tried AssemblyAI [9], but it is paid and does not support audio streaming (just sending audio files). Finally we also tried some less well known options like Pyannote[10] and Kaldi[11], which are open-source toolkits for speaker diarization and speaker recognition. However, neither of them served our needs once again missing a feature where we could stream our microphone output directly as a stream.

Back to AWS Transcribe. We ended up using AWS SDK (software development kit) for Java because it was compatible with our Android application and provided an easier way to connect with the service.
If we take a closer look at AWS Transcribe service, we are effectively first creating a connection using the previously generated credentials, starting an HTTP/2 connection with the servers and then enabling the necessary properties for speaker differentiation by setting the flag "showSpeakerLabel" to true. After this is done we forward our microphone input as a stream directly to AWS using the provided library which tries to break it into segments such as a change in speaker or a pause in the audio [12]. From now on we receive a speaker identifier, the words spoken and time interval of each spoken word. Additionally we could receive intermediate results, and those would be faster to receive, but we would only be getting speaker differentiation on our final results, so we changed the flag called "isPartial" to false and prevented the flow of unnecessary messages, effectively only receiving complete results.
AWS Transcribe does have a problem: it does not allow us to specify the number of speakers participating in the conversation. The way we overcame this problem was to use the tags we had identified when configuring the users and just discard speaker tags that were not recognized, as previously explained in section **3.2**. Overall it worked and met our specifications. One thing we would have liked to be better is the intrinsic delay of the real time computation and network latency which is present in all the different cloud solutions we tried. In the end we deemed it acceptable to have an average two to three seconds of delay

between updates on our spoken time since in a longer conversation it would not matter much.

3.5. Summary

We ended up achieving what we described in the proposal: perform speaker separation, compute times spoken and show gradually, using a RGB light, the color of the speaker who has talked the most.
We are especially happy with the communication between the smartphone and the light - we can send messages at a decent rate and make the light change colors with unnoticeable delay.
Has mentioned previously, the delay in updating spoken times of each user is caused by the responses coming from AWS Transcribe: we wait for a full transcription, which might take some seconds to be sent to us, and thus the times are updated only every couple of seconds, not being as smooth as we would like. However, this is the service we chose to work with and we cannot change the response times or how the service handles the speaker diarization.
The last thing to be mentioned is the effectiveness of the speaker diarization: its performance depends a lot on the background noise. We have done some experiments and in fact, if we use our device in a quiet environment, the speaker diarization is almost flawless: the voices are correctly identified and the transcription is almost perfect. On the other hand, if we try to use it in a noisier environment, the performance decreases both in the quality of the transcription and in the voice recognition. It starts recognizing background voices, including some words said by non-participants as if said by one of the participants, and sometimes even mixes the voices of the participants, when their voices are not so different.

4. References

(1) http://www.martyncurrey.com/arduino-and-hc-06-zs-040/
(2) https://towardsdatascience.com/audio-deep-learning-made-simple-automatic-speech-recognition-asr-how-it-works-716cfce4c706
(3) https://cloud.google.com/speech-to-text/docs/multiple-voices
(4) https://cloud.google.com/speech-to-text/docs/streaming-recognize
(5) https://cloud.google.com/speech-to-text/docs/endless-streaming-tutorial
(6) https://developers.google.com/resources/api-libraries/documentation/speech/v1/python/latest/speech_v1.speech.html
(7) https://rposbo.github.io/speaker-recognition-api/
(8) https://medium.com/vmacwrites/track-whos-speaking-with-speaker-diarization-2e3eac2de2c3
(9) https://www.assemblyai.com/

(10) https://github.com/pyannote/pyannote-audio
(11) https://github.com/kaldi-asr/kaldi
(12) https://docs.aws.amazon.com/transcribe/latest/dg/streaming.html