Team Audio Documentation

This document contains parts the team worked on during the project which is not included in the final result, but they led the team to better understand how to make progress and reach the final result.

We still believe it can be of use to people who continue work on the project, so they don't repeat the same mistakes we made or maybe solved they come up with better solutions than us.

1. Speaker Diarization

1.1 Setup - Raspberry pi (NOTE: Not working)

- NOTE: you might have to use pip3 instead of pip, also make sure that your pip is up to date:
  - **pip install --upgrade pip**
  - or **pip3 install --upgrade pip**

First install the librosa package (documentation doesn't mention this but it throws an error if its not installed when you pip the pyannote project)

- **sudo apt install llvm-9**
- **LLVM_CONFIG=llvm-config-9 pip install llvmlite**
- **pip install librosa**

Install pyannote by downloading github repo and pip:

- **git clone https://github.com/pyannote/pyannote-audio.git**
- **cd pyannote-audio**
- **git checkout develop**
- **pip install .**

We will also need Pytorch in order for the pyannote framework to process the sound, due to architectural differences of CPU between pc and raspberry Pi this gets a little more tricky. We can't just pip install torch as we normally would.

- ~~Follow this guide to install pytorch on the pi: https://medium.com/analytics-vidhya/quick-setup-instructions-for-installing-pytorch-and-fastai-on-raspberry-pi-4-5ffbe45e0ae3~~

Install pytorch for arm by:

- **git clone https://github.com/radiodee1/pytorch-arm-builds.git**
- **cd pytorch-arm-builds/**
- **sudo pip3 install torch-1.1.0-cp37-cp37m-linux_armv7l.whl**

(links: https://github.com/radiodee1/pytorch-arm-builds, https://eide.ai/post/2020-06-30-pytorch-raspberry/)

clang
https://solarianprogrammer.com/2018/04/22/raspberry-pi-raspbian-install-clang-compile-cpp-17-programs/

1.2 Setup - Ubuntu Server

- Since we're running with an x86 architecture we avoid all struggle with pytorch and other dependencies (compared to the pi).
- Specs of current server:
  Standard B2s (2 vcpu, 4 GB memory)
- Running in Azure
- The code for the server is published here:
  https://github.com/r8bin/mbox_speaker_diarization

First, make sure python 3 is installed:

- **python3 --version**

Then make sure pip is installed and upgraded:

- **pip --version**

or:

- **pip3 --version**

if the version is < 20.xx

- **pip install --upgrade pip**

or:

- **pip3 install --upgrade pip**

(If pip is not even installed do: **sudo -H pip3 install --upgrade pip**)
Install LLVM and Librosa:

- **sudo apt install llvm-9**
- **LLVM_CONFIG=llvm-config-9 pip install llvmlite**
- **pip install librosa**

install pytorch:

- **pip install torch**

Install pyannote by downloading github repo and pip:

- **git clone https://github.com/pyannote/pyannote-audio.git**
- **cd pyannote-audio**
- **git checkout develop**
- **pip install .**

install required sound library for linux:

- **sudo apt-get install libsndfile1**

The script is already on the server located in /home/mbox/speaker_diarization/scripts
it's taken from the pyannote audio-hub on github:

- https://github.com/pyannote/pyannote-audio-hub

The script is just for testing out on a sample sound recording of two people talking (taken from youtube and then recorded on respeaker).
Some issues was raised:
The script output that multiple people were talking when there were just two, it seems like the script/model doesn't recognize the same person speaking again and then just classifies it as a new person every time. However this script was just taken as a sample from the github page and there are some tweaks that I haven't looked into yet. So there is some investigation to be made in the script/model/framework.

1.3 Setup - Audio recording script and client

This code for this part is published here:
https://github.com/belismau/record_sound

1.3.1 Audio recording script

To send audio from the client to the server, we have managed to modify a script that is built with PyAudio. This script will record in 5 seconds, and automatically send the audio file (.wav-file) to our Azure-server.

The audio script for recording (app.py) is inspired by:
https://stackoverflow.com/questions/40704026/voice-recording-using-pyaudio.

- **pip install pyaudio**

The command above works fine on Raspberry PI, but for some reason I can't run it on my Mac. Instead I used the commands below:

- **brew install portaudio**
- **pip install pyaudio**

Solution:
https://stackoverflow.com/questions/31236194/installing-pyaudio-for-python3-on-osx/43859453#43859453

Also, when running the app.py (recording script) on Raspberry PI, the script starts to record, but it prints several error messages - which, however, do not affect anything...

1.3.2 Client

To send the recorded audio file (that was created in 8.1) from our client app to the Azure-server, we have used ZeroMQ, "an open-source universal messaging library". This allows us to use, for example, TCP protocol. It can talk to the server by connecting to an address and port created by the Azure-server (the address "*localhost"* and port "*5555"* are replaced):

- **socket.connect("tcp://localhost:5555")**

We are also using Push and Pull pattern for sending messages. More about Push/Pull here:
https://learning-0mq-with-pyzmq.readthedocs.io/en/latest/pyzmq/patterns/pushpull.html.

- **pip install pyzmq**

The client app (client.py) is inspired by:
https://zeromq.org/get-started/?language=python&library=pyzmq#

2. DOA (Direction Of Arrival)

2.1 Simple DOA script in python

- Ex: https://www.youtube.com/watch?v=gGVQ-9f7azs
- DOA = Direction Of Arrival, make a script that determines audio source in circular degrees (like in the video above).
- https://github.com/respeaker/mic_array

Download the Respeaker Github-project: https://github.com/respeaker/mic_array,

- **git clone https://github.com/respeaker/mic_array**

If you have installed the drivers correctly according to the Seedstudio webpage, you only need to install numpy (if it's not already installed)

- **pip install numpy**

Then install PyAudio by

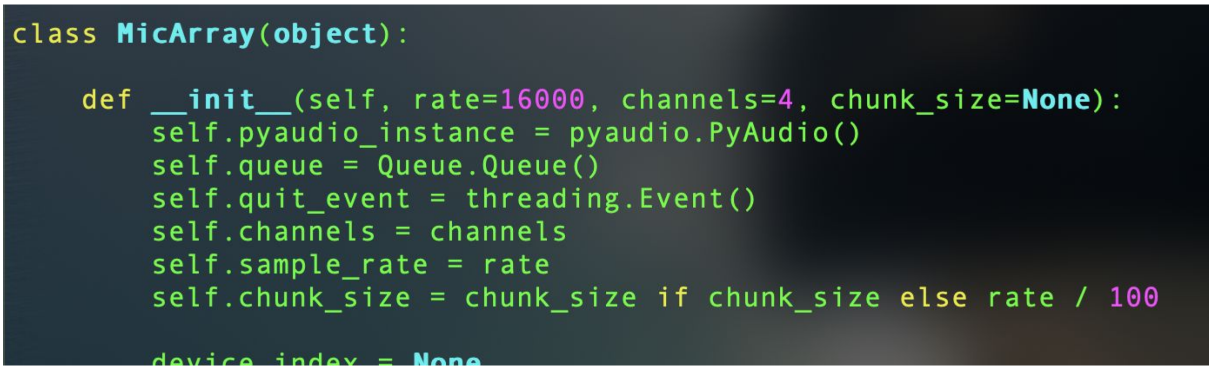
- **pip install pyaudio**

If it doesn't work properly or gives you an error, try following withs sudo apt instead of pip:

- **sudo apt-get install python-pyaudio python3-pyaudio**

Lastly some changes need to be made in the mic_array.py to work properly with the Pihat 4mic array.

- **vim mic_array.py** (or if you use Graphical interface, open with any editor)
- change the "channels" parameter to 4 in the init method:

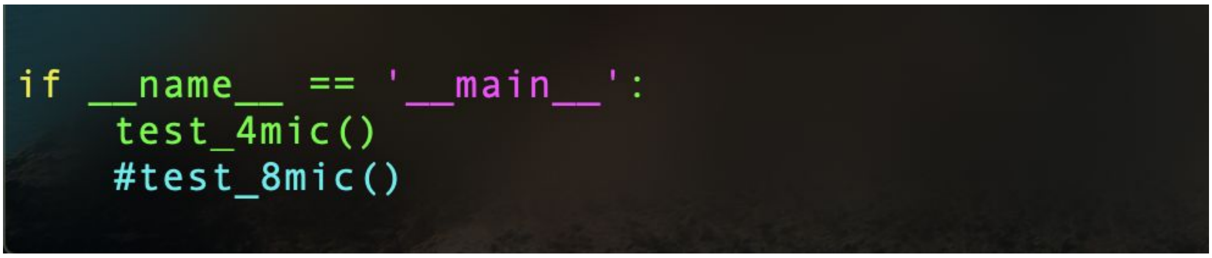


```
class MicArray(object):

    def __init__(self, rate=16000, channels=4, chunk_size=None):
        self.pyaudio_instance = pyaudio.PyAudio()
        self.queue = Queue.Queue()
        self.quit_event = threading.Event()
        self.channels = channels
        self.sample_rate = rate
        self.chunk_size = chunk_size if chunk_size else rate / 100

        device_index = None
```

- comment out test_8mic and uncomment test_4mic (at the bottom of the code):



```
if __name__ == '__main__':
    test_4mic()
    #test_8mic()
```

Finally run the script by entering **python mic_array.py** and try talking around the mic, you will see it constantly printing degrees between 0-360 determining where the audio comes from.

2.2 Thoughts

The script is a good start point, we can reuse a lot of this code in order to not reinvent the wheel again. Then just tweak the script to for instance send the data to our Timeflux server instead, also lower the refresh rate of DOA. Right now its printing out even if no one is speaking, this can be solved by using voice detection for triggering DOA, in the project there is also a script with VAD (Voice Activity Detection) which only calculates DOA when it detects voices, this should be easy to interpret.

3. Explanation of some parts installed and why we install them.

3.1 Pyannote

Neural building blocks for speaker diarization: speech activity detection, speaker change detection, speaker embedding. Highest rating in various pages online, seems to be an good starting point for the project.

3.1.1 LLVM

The LLVM compiler infrastructure project is a set of compiler and toolchain technologies, which can be used to develop a front end for any programming language and a back end for any instruction set architecture.

3.1.2 Librosa

Python library for audio and music analysis. https://librosa.github.io/

3.1.3 PyTorch

Is an open source machine learning framework that accelerates the path from research prototyping to production deployment.

4. ODAS (Open embeddeD Audition System)

A library coded in the C language, for more portability to run on low-cost embedded hardware. With this library we should be able localize sound sources, tracking, separation of different sound sources and post-fitering.

As this library contains the “functions” we need it seems to be a good starting point.

Before we begin make sure you have installed the following libraries:

Prerequisites:

4.1 FFTW [1]:

FFTW is a C subroutine library for computing the discrete Fourier transform (DFT) in one or more dimensions.

```
sudo apt-get install libfftw3-dev
```

4.2 LibConfig [2]:

Libconfig is a simple library for processing structured configuration files. The file format is more compact and more readable than e.g XML.

```
sudo apt-get install libconfig-dev
```

4.3 [3]:

The ALSA library API is the interface to the ALSA drivers. Developers need to use the functions in this API to achieve native ALSA support for their applications.

```
sudo apt-get install libasound2-dev
```

4.4 [4]:

The GNU Compiler Collection includes front ends for C, C++ and several other languages as well as libraries for these other languages.

4.5 [5]:

CMake is an open-source, cross-platform family of tools designed to build, test and package software. I installed (in ubuntu) by writing this command line:

```
sudo snap install cmake --classic
```

(it may also work with only) :

```
sudo snap install cmake
```

Installing process:

Start with cloning the project:

git clone https://github.com/introlab/odas.git

Create a folder to build the project:
cd odas
mkdir build
cd build

Run CMake in the build directory, (in the cd odas directory)

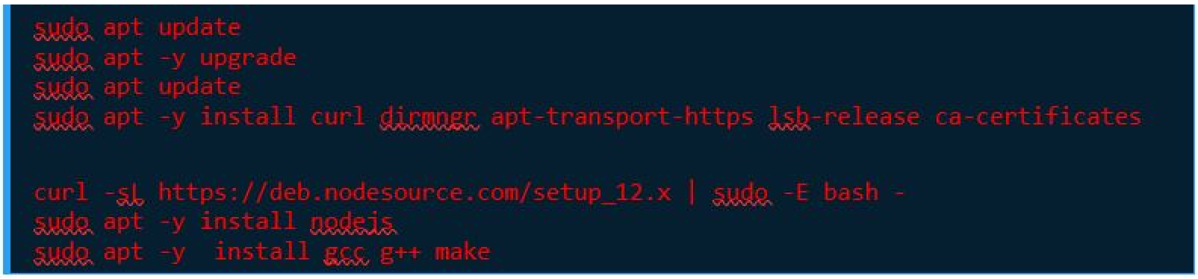
cmake../ (write directory name)

Compile project:

make

Installing process for the Desktop installation (odas_web):

1. **Install Node.js v12 [6]**



```
sudo apt update
sudo apt -y upgrade
sudo apt update
sudo apt -y install curl dirmngr apt-transport-https lsb-release ca-certificates

curl -sL https://deb.nodesource.com/setup_12.x | sudo -E bash -
sudo apt -y install nodejs
sudo apt -y  install gcc g++ make
```

2. **Clone this repository:**

git clone https://github.com/introlab/odas_web.git

3. **Run** npm install **in the cloned repository base folder (the folder which contains main.js)** Heads up, the installation might tell you that something went wrong, try the next command anyway(npm start). It seems the program is installed and works fine.

4. **Run** npm start **in the base folder. And to exit press** ctrl + c

5. **Check Versions (good practise):** node --version **and** npm --version

Important !!!

Please note that even if ODAS Studio can be installed on a Raspberry Pi, it will not perform properly due to CPU limitations. Please see the ODAS configuration section for instructions on how to sink data from a Pi running ODAS to another computer trough sockets [7].

4.6 Conclusion:

Even as we followed the steps and successfully installed the ODAS library, we continued to have problems on the raspberry pi, as the program kept freezing and crashed, making us doubt if it would be sufficient to use together with a camera and other software. It´s nothing we could use in our project, but demonstrated what we should aim at doing with the algorithms we’re building for the ReSpeaker.

4.7 Suggestion:

Keep investigating the possibility to send the data through a server, maybe it can work on a jetsons which has a more powerful cpu.

5. Speaker-Diarization framework

(Not Working)

A Speaker recognition framework based on VGG Speaker recognition, that is a deep CNN [1] based neural speaker verification system, which is trained to map voices to compact embedding spaces. It can then use the distance between vectors in this embedding space to measure the similarity between speakers [2].

Secondly The framework also uses googles UIS-RNN-Api [3]-[4] for speaker diarization and this part of the code was the interesting part. My plan is to use this to investigate the how the speaker diarization works and see if it possible to apply it to the Raspberry pi.

Before we begin make sure you have installed the following libraries:

5.1 Prerequisites:

1. Pytorch

   Pytorch is an open source machine learning framework that accelerates the path from research prototyping to production deployment.

- pip install torch

2. Keras[5]

   Keras is a high-level neural networks API for Python.

- pip install keras

3. Tensorflow [6]

   TensorFlow is an open source software library for high performance numerical computation.

- pip install tensorflow

4. pyaudio

   Bindings for PortAudio v19, the cross-platform audio input/output stream library.

- sudo apt-get install libasound-dev portaudio19-dev libportaudio2 libportaudiocpp0
- pip install pyaudio

You may have problems on Mac if so use : `brew install portaudio` before `pip install pyaudio`

5. You may also need to install libcudart

- sudo apt-get install libcudart10.

6. scikit-learn

For python 2:

```
sudo apt-get install build-essential python-dev python-setuptools \
                     python-numpy python-scipy \
                     libatlas-dev libatlas3gf-base
```

For python 3:

```
sudo apt-get install build-essential python3-dev python3-setuptools \
                     python3-numpy python3-scipy \
                     libatlas-dev libatlas3gf-base
```

You may encounter problems libatlas-dev, if so just run:

```
sudo apt-get install libatlas-base-dev
```

5.2 Installing process:

Start with cloning the project:

```
git clone https://github.com/taylorlu/Speaker-Diarization.git
```

1. Speaker recognition

Enter the folder that generates the embeddings and start the script:

```
cd ghostvlad
```

```
python predict.py
```

OR

```
python3 predict.py
```

Note: It is possible that you can replace read_wav with read_wav_fast in utils.py, but be aware that the result is that the sample rate is not constant. This may be helpful in reading other datasets to, as my teammates have had problems with datasets taking hours to train.

5.3 Conclusion:

I get a problem here following:

```
mgpu = len(keras.backend.tensorflow_backend._get_available_gpus())
```

```
AttributeError: module 'keras.backend' has no attribute 'tensorflow_backend'
```

I have tried to solve with various solutions online, but nothing seem to fix it.

5.4 Speaker diarization.

Start the script to train the model.

```
python train.py
```
OR
```
python3 train.py
```

```
Error occurred : FileNotFoundError: [Errno 2] No such file or directory:
'./ghostvlad/training_data.npz'
```

The error occurs because the training data file is missing. These files are huge and needs to be downloaded separately, because the neither the pi or the ubuntu not have enough memory to handle them.

5.5 Conclusion:

We have to further investigate in how to get a valid npz file with training data. The files that were available for us were minimum size of 10 gb. And hard to get them to work on the Ubuntu or pi because of memory limits.

5.6 Suggestion:

This framework seem to be heavy for our project, but maybe we can use parts of the speaker recognition and diarization later in our project.

6. Training Models With Pyannote-Audio

The group came to the conclusion that models need to be trained to get the speaker diarization to work correctly. Therefore, Pyannote-audio toolkit was used to achieve this. This toolkit has its own data preparation tutorial, and we started with it, before moving on to all the other necessary tutorials (also from Pyannote).

1. **Download the data preparation**
   https://github.com/pyannote/pyannote-audio/tree/master/tutorials/data_preparation
   The tutorial from the link above was followed, and since it takes time and requires downloading a large amount of data (~ 50gb), Google Colab was used. They give GPU options (which is necessary when working with machine learning) and a large amount of free space.

2. **Apply pretrained models on data**
   https://github.com/pyannote/pyannote-audio/tree/master/tutorials/pretrained/model
   When the data preparation was downloaded, steps from the above link were followed to apply pretrained models from Pyannote on the data. Pyannote uses models available on *torch.hub* that were pretrained on AMI training subset. Unfortunately, this did not work, either with our own audio files or the trained ones.

3. **Train the model**
   https://github.com/pyannote/pyannote-audio/tree/master/tutorials/models/speech_activity_detection
   Instead of using pretrained models, a decision to train the models was taken. This was achieved by following the steps available on the link above. When running the command, an error occurred: *"Can't find file MUSAN/background_noise.txt"*. This was solved by placing all files generated from the data preparation step to the pyannote-audio folder (the root folder). After this fix, the training started successfully, but it took a long time. According to Pyannote, it says: *"...will train the network using the training subset of AMI database for 200 epochs"*. It took 5 hours to reach the first epoch. We decided to stop the training due to lack of time.

   One epoch was successfully trained, but gave no results when using our own audio files. Perhaps several epochs need to be trained.

6.1 Conclusion

The pyannote-audio library for training models is huge, and due to its size and well described tutorials, there should be a way to build a speaker diarization module. However, some tutorials are a bit hard to follow, and they gave errors even though the steps were followed. This makes it difficult for the group to make progress. Also, all training parts seemed to take a very long time, and for the group, this can be difficult to handle due to the fact that there is a time limit.

6.2 Suggestion

After trying out the Pyannote library with no clear results, a decision was made to focus on the Google API for speaker diarization called *Cloud Speech-to-Text API* (link below). https://cloud.google.com/speech-to-text/docs/multiple-voices.

7. Resemblyzer

To create a speaker diarization module, a tutorial on https://medium.com/saarthi-ai/who-spoke-when-build-your-own-speaker-diarization-module-from-scratch-e7d725ee279 was followed. The author used a repository called Resemblyzer, which can be found here: https://github.com/resemble-ai/Resemblyzer.

1. **Installation**
   - *git clone https://github.com/resemble-ai/Resemblyzer.git*
   - *pip3 install spectralcluster*
   - *cd Resemblyzer*

2. **Setup**
   After installation, a file named *app.py* was created. The code on the web page was copied and pasted into the python file. This code created a list-variable called *labelling*, which was looped through. All this because the variable contained information about who spoke when.

The Resemblyzer package worked, but gave no correct data back when adding our own audio files (*audio_file_path* = '*my_recording.wav*').

8. Good source for different diarization frameworks and open-source projects.

https://github.com/wq2012/awesome-diarization

https://awesomeopensource.com/projects/speaker-recognition