Universität Hamburg

DER FORSCHUNG | DER LEHRE | DER BILDUNG

Bachelor Thesis

Robust preprocessing for real-time audio-visual speech separation

Signal Processing Group
Faculty of Informatics
University of Hamburg

Berkkan Katirci
Matriculation Number: 7298823

March 17, 2023

First Reviewer: Prof. Dr.-Ing. Timo Gerkmann
Second Reviewer: Tal Peer, M.Sc.
Daily Supervisor: Julius Richter, M.Sc.

Abstract

Including video information is a promising approach to improve speech processing systems. Many proposed works have already provided inspiring results, including methods for audio-visual speech enhancement and audio-visual speech separation. In practice, however, problems often arise when it comes to the real-time capability and robustness of such methods. Therefore, this bachelor thesis addresses the robustness and real-time capability of audio-visual speech processing methods. The focus is the development of a real-time preprocessing module that prepares audio and video data for a deep neural network for audio-visual speech separation. The thesis contains the analysis of the technical requirements for setting up a real-time demonstrator in an acoustic lab and the implementation and testing of the final real-time preprocessing module. This involves the choice of the video camera, the signal transmission across a long distance, the face tracking method, the preparation of the region of interest, the processing of the audio, and finally the combination of both modalities.

Zusammenfassung

Die Einbeziehung von Videoinformationen ist ein vielversprechender Ansatz zur Verbesserung von Sprachverarbeitungssystemen. Viele vorgeschlagene Arbeiten haben bereits inspirierende Ergebnisse geliefert, darunter Methoden zur audio-visuellen Sprachverbesserung und zur audio-visuellen Sprechertrennung. In der Praxis treten jedoch oft Probleme auf, wenn es um die Echtzeitfähigkeit und Robustheit der Methoden geht. In dieser Bachelorarbeit wird daher die Robustheit und Echtzeitfähigkeit von audio-visuellen Sprachverarbeitungsmethoden erläutert. Im Mittelpunkt steht die Entwicklung eines Echtzeit-Vorverarbeitungsmoduls, das Audio- und Videodaten für ein tiefes neuronales Netz für audio-visuelle Sprechertrennung aufbereitet. Die Arbeit beinhaltet die Analyse der technischen Voraussetzungen für den Einsatz eines Echtzeit-Demonstrators in einem Akustiklabor sowie die Echtzeitimplementierung und den Test des fertigen Echtzeit-Vorverarbeitungsmoduls. Dies beinhaltet die Auswahl der Videokamera, die Signalübertragung über eine große Distanz, die Gesichtsverfolgungsmethoden und -verarbeitung, die Verarbeitung der Audiodaten und schließlich die Kombination beider Modalitäten.

Contents

1 Introduction

Hearing and seeing are among the most important senses of humans. The human brain is constantly evaluating these senses together to better navigate the environment. For example, the brain combines the senses of hearing and seeing to better understand a person we are talking to. This phenomenon of selective listening refers to the so-called *cocktail party effect* [1]. The name stems from challenging acoustic situations, e.g. in noisy environments like a cocktail party. The person who tries to understand a speaking person focuses on the speech but also on the lip movement and tries to combine both in an efficient way. The technology sector attempts to adapt such human behaviours with the use of deep neural networks (DNNs). The use of two or more modalities is called *multimodal learning* and aims to combine the used modalities [2]. Examples of multimodal learning for speech processing include *audio-visual speech enhancement* (AVSE) and *audio-visual speech separation* (AVSS) [3].

In particular, audio-visual speech enhancement gained popularity because it provides better enhancement results (which is shown in Section 1.1) than audio-only speech enhancement models [4]. Besides better performance, audio-visual speech separation also solves the permutation problem [5]. However, due to the enormous technical challenge, real-time methods for audio-visual processing are currently less intensively studied than for audio-only approaches [6].

A typical application for audio-visual speech separation is telecommunications. Imagine, for instance, a situation where two employees have an online meeting at the same time and are in the same room. The problem occurs when both employees speak at the same time therefore the microphone captures both speakers. The mixed speech is difficult to understand for the other person on the line. Audio-visual speech separation could be used to separate the speakers.

Another example is hearing aids, which provide relief for hearing loss. Unfortunately, today's hearing aids are limited in their ability to restore speech perception in a noisy environment. The main challenge is that hearing loss is primarily a loss of sensitivity, and hearing aids rely on amplifying sound. However, simply amplifying sound does not adequately address the complex process of speech perception. In particular, speech perception in noisy environments remains a significant problem because hearing aids may not distinguish between relevant and irrelevant sounds, leading to impaired speech understanding. Audio-visual speech separation could help do improve hearing aids in a way to have a better perception of speech [7].

In this bachelor thesis, I analyze the real-time capability of audio-visual models and develop a real-time preprocessing module for preparing the audio and video data. This module provides the input for a DNN used for audio-

visual speech separation. For the DNN model, we used *VisualVoice* because the code and the model is publicly available at GitHub[1].

The bachelor thesis is structured as follows. Section 2 describes related work. After that, I will address the problem and define the goal of this thesis in Section 3. The hardware and the setup of the acoustic lab are described in Section 4. Section 5 is about the audio preprocessing. In Section 6, I present methods for video preprocessing which involve face tracking methods and extraction of the region of interest (ROI). Then, I address in Section 7 the combination of the audio and video. Section 8 is about the problems during the implementation. Finally, a conclusion is drawn in Section 9.

1.1 Importance of video

Recent studies in audio-visual speech enhancement and audio-visual speech separation have shown superior performance over their audio-only counterparts. Adding video information as input effectively improves the intelligibility and speech quality compared to audio-only speech enhancement.

Gao and Grauman [8] showed stunning results in audio-visual speech enhancement and also in audio-visual speech separation. Here, I will show only the results of audio-visual speech separation because that is the focus of this thesis. Table 1 shows the speech separation performance of their proposed method VisualVoice. For evaluation, they used different metrics including signal-to-Distortion Ratio (SDR), Signal-to-Interference Ratio (SIR), and Signal-to-Artifacts Ratio (SAR) [9]. In addition to that, they used two perceptually motivated speech metrics. The first one is Perceptual Evaluation of Speech Quality (PESQ) [10], which measures the general perceptual quality of the separated speech. The second one is Short-Time Objective Intelligibility (STOI) [11], which is correlated with the intelligibility of the signal.

The baseline *Audio-only* uses the same architecture from Gao and Grauman except that no visual feature is used. The second baseline is another method called *AV-Conv* that predicts the phase and magnitude of the spectrogram [4]. Lastly, the audio-visual speech separation model from Gao and Grauman, VisualVoice uses a lip motion and facial attributes analysis network for speech separation. Comparing VisualVoice with its audio-only variant, a significant improvement in all metrics can be seen, justifying the use of the visual modality for audio-visual speech separation. Compared with AV-Conv, VisualVoice achieves better results in all metrics, which could be due to an improved network architecture.

In general, audio-visual speech separation does not only improve the quality but also solves the permutation problem, which audio-only speech separation usually struggles with. The permutation problem is a sequence

[1] https://github.com/facebookresearch/VisualVoice

	SDR	SIR	SAR	PESQ	STOI
Audio-only	7.85	13.7	9.97	2.61	0.82
AV-Conv	8.91	14.8	11.2	2.73	0.84
VisualVoice	10.2	17.2	11.3	2.83	0.87

Table 1: Audio-visual speech separation results on the VoxCeleb2 dataset [8].

problem, related to audio-only methods because audio-only do not know who is speaking in a given time [5]. Audio-visual speech separation solves that problem because the video stream provides significant information about the speakers. For example which speaker is talking in a scene and which is not. Due to the aforementioned reasons, audio-visual speech enhancement and audio-visual speech separation recently gained more popularity in the speech-processing community. However, the use of two modalities increases computation and complexity.

2 Related Work

Selective hearing is a very important part of humans. So that we can focus on the things that we want to hear most and ignore the rest. This effect is commonly seen in a noisy environment. This is also called the *cocktail party effect*. This effect was described by E. Colin Cherry in the 1950s [1]. He showed that visual information plays an important role in separating speakers. There are many works related to audio-visual speech separation. There is one paper from Michels et al. [3] which gives a comprehensive overview of the audio-visual speech separation and enhancement field. The paper also discusses the use of visual information in audio-visual speech separation and enhancement, and the importance of effectively fusing audio and visual modalities.

Gao and Grauman introduce a new audio-visual speech separation method named VisualVoice [8]. The difference between VisualVoice to other state-of-the-art audio-visual speech separation models is that VisualVoice uses facial attributes in addition to the lips as a ROI. They use facial attributes to characterize the speaker to bring them closer together in a Cross-modal embedding space. The method achieved good results, which were mentioned in Section 1.1. The DNN model is also fast and it takes around 200ms to produce an output. Because this model is fast, I have considered and tried adding VisualVoice after the preprocessing module, more on this at the end. All audio-visual speech separation and audio-visual speech enhancement approaches are based on multimodal deep learning because it uses two modalities the audio and video information [2][12].

3 Problem description

The problem with current audio-visual speech separation is, that there is no real-time capable model yet (that I know of), that supports seamless and natural communication between people. Besides that, audio-visual speech separation models require a preprocessed input. For example, the state-of-the-art DNN model VisualVoice [8] needs a preprocessed video as input in order to run properly. This is not suitable for real-time applications, as it would take a separate step to preprocess the video first. This would take several seconds to process a video file and get the desired output. After the preprocessing is done, the model can start with the preprocessed video to separate the voices in the video. This method has performance issues and cannot be used in real-time applications.

I have created a preprocessing module which preprocesses the audio and video in real-time and gives the video and audio information in a stream to a DNN model.

The preprocessing module is illustrated in Fig. 1. On the left is the input of the video and audio device. The two streams are processed separately. The audio preprocessing will be discussed in detail in Section 5, and the video preprocessing in Section 6. Receiving the input of the audio and video must be done with the low latency possible to obtain a real-time capable preprocessing module. After each preprocessing step is done, which is a continuous process, the input must be aligned before passing the two inputs to the DNN model. The alignment and combination will be explained in Section 7. On the right is an audio-visual speech separation model in this case it is VisualVoice [8]. Theoretically, any audio-visual speech separation DNN could be attached behind the preprocessing module. For the DNN model to work properly the parameters of the preprocessing need to be changed to fulfil the requirements of the selected DNN.

The goal of this thesis is to develop a real-time preprocessing module. This module can theoretically be attached before every audio-visual speech separation DNN model, if they are fast enough, to have a real-time experience. My focus lies on the left part surrounded in blue in Fig. 1.

As a bonus, I tried to add VisualVoice after the preprocessing module. One thing in advance, because VisualVoice is not fast enough to reach a real-time experience I added a synthetic delay but more about that at the end.

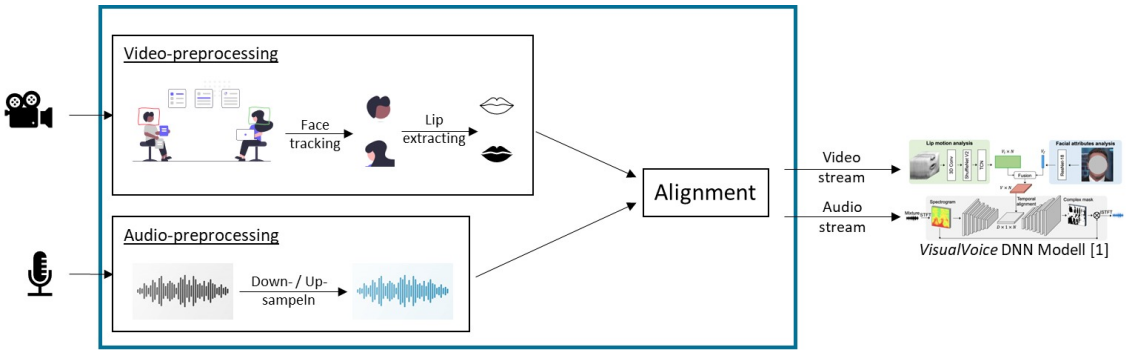

Figure 1. Preprocessing module structure. On the left are the two input streams which are processed separately but in parallel. The top one describes the video preprocessing and the bottom one describes the audio preprocessing. Afterwards, the alignment of the two streams takes place. In the end, the streams are sent to a DNN model to return the output. In this case the DNN model is VisualVoice.

4 Hardware and Setup

This section describes the problems with the hardware and setup and their solution. What hardware I discovered and which one we used at the end in the acoustic lab. I will then explain the setup.

4.1 Hardware

In our acoustic lab, we do not have any video hardware so first, I searched for hardware we would need to buy. While researching for the hardware, I was overwhelmed with cameras and information. To simplify it I categorized and broke it down. First, I will discuss different ways to connect the video device to the computer. In the second part, we will look at different video devices that suit our case the best. To get the video signal as input from the camera, we need a video capture device which I will explain at the end.

4.1.1 Connection

The first problem is, the connection of the camera to the computer because the computer is in another room approximately 15 meters away. A non-scale drawing can be seen in Fig. 2. The microphone is connected through an Serial Digital Interface (SDI) cable. The wire represented with a black dashed line goes through a box on the wall which then connects to the computer (not all details are mentioned). To connect the camera to the computer, we need to connect a 15 meters cable with no or less data loss. There are a lot of options.

First Universal Serial Bus (USB), this connection is fast, especially USB Typ-C 3.2 which has a transmission rate of 48Gbit/s but USB cables are very limited in their length. The max length is 3 meters which is only 1/5 in our case. There are also extensions like USB-Hubs or fibreglass extensions but these extensions are all interposed which increases the latency because we have more connection points till we get to the PC [13][14]. There are also active USB cables which are often used in Virtual Reality (VR). This type of cable has a max length of 30 meters which suits our case. Because there is no interconnection the latency is also small near zero.

The second opinion is High Definition Multimedia Interface (HDMI), which has a pretty good max length from about 5 to 20 meters. It depends on the cable type. There are two types of HDMI cables digital and optical. There are also many different versions of HDMI but in this era version 2.0 and 2.1 is the minimum requirement. HDMI 2.1 has a transmission speed of around 48Gbit/s which will transmit the video input fast to the PC with low latency. HDMI 2.0 transmission rate is 18Gbit/s which is sufficient for this bachelor thesis. HDMI 2.0 thems to be a good option because it covers our 15 meters and it is fast enough [15][14].

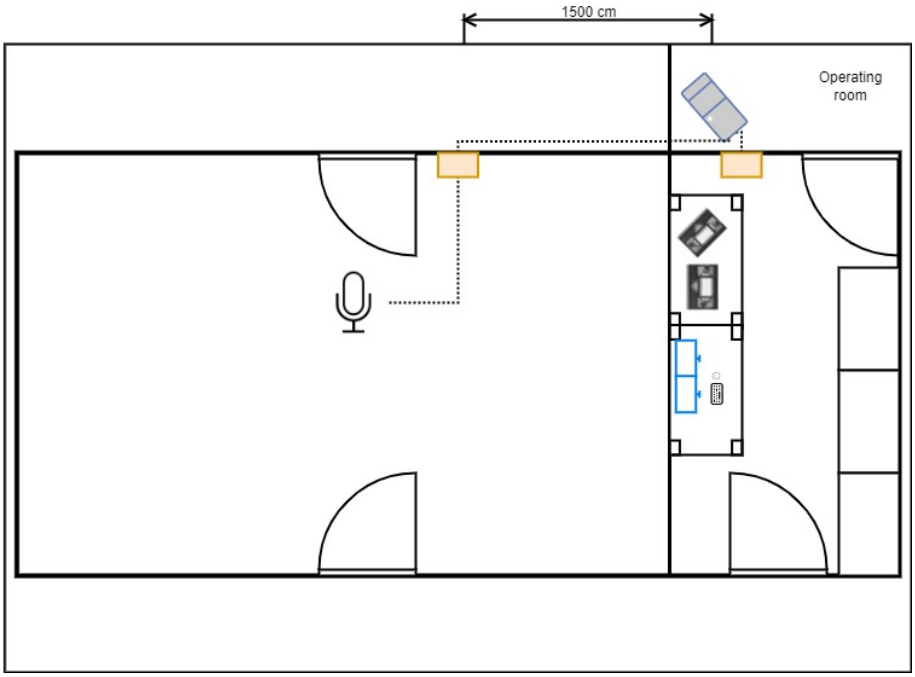

Figure 2. Acoustic lab microphone setup. The microphone is located in the middel of the recording room. The microphone is connected via a SDI cable which is plugin in the wall box represented with a black dotted line. Which then goes to the computer. **(non-scale drawing)**.

Wireless Local Area Network (WiFi) is the third option, which has a nearly unlimited length. But the transmission speed depends on the WiFi speed. If the WiFi is fast, the transmission of the video is also fast and has good quality. However, when the WiFi is slow or overloaded then the transmission is also slow which is bad because it increases the latency. Moreover, WiFi connections are sometimes not stable and likely to disconnect. Because of that reasons, WiFi is not an option for this bachelor thesis.

The last option we have considered is SDI. Those types of cables are used by broadcasters like news channels. Society of Motion Picture and Television Engineers (SMPTE) developed these professional video standers in 1989 and till now SDI cables got six upgrades but the main standards in these years are 3G-SDI, 6G-SDI and 12G-SDI. Fig. 3 shows the different SDI types.

The advantage of SDI cables is that they can detect on recover most of the errors which were caused by noise or interference. This is achieved by the use of the Hamming Code[2]. In this way, we would have nearly no

[2]The Hamming Code was invented by Richard Wesley. The Hamming Code is a linear correcting block code which is used in the signal processing and telecommunication sector for a secure data transmission and data saving. With the *Hamming Code* it is possible to detect two bit and correct one bit errors.

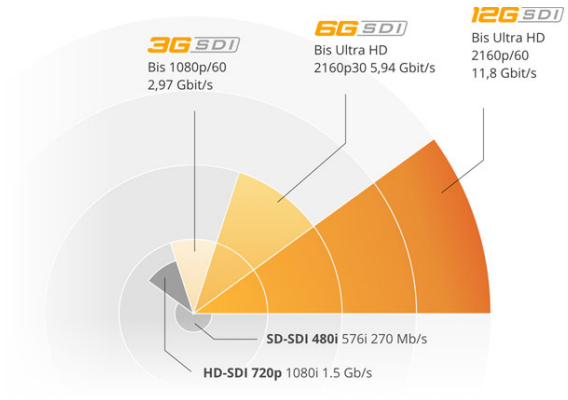

Figure 3. Differents of SDI types [18].

packed loss. Beyond that, the length of SDI cables is up to 100 meters and more [16][17].

To sum it up. The best option to connect the video device for this bachelor thesis would be SDI cables. But HDMI cables are also not a bad choice. It is also important to keep in mind that the choice of the connection cable depends on the choice of the camera as well.

4.1.2 Video device

When it comes to the choice of a camera we must pay attention that the camera has a clean HDMI output. That means that the extra information on the display - like battery status, resolution, blend ... - can be removed [19].

For the video device, there are numerous options.
The first option would be a webcam but this option is disqualified because it can not connect via HDMI only over USB.
The High Dynamic Range (HDR) camera is the second option. HDR cameras are professional cameras and are used by professional photographers. Besides these cameras can connect with USB and also with HDMI. Sony Alpha 6000 is a camera that is used by a lot of twitch[3] streamers and YouTubers. Except that this camera can be connected via HDMI it also has a clean HDMI output. As good as HDR cameras are they have their price. The Sony Alpha 6000 cost around 600€ which is very expensive.
An alternative to HDR cameras are camcorders. Most camcorders have all three, USB, HDMI, and WiFi connectivity so I could theoretically test all three connections. Camcorders take good pictures and videos too and are much cheaper. Their price range is from 100€ to 400€ depending on the extra features and recording resolution. Panasonic HC-V180EG-K is what I found which cost roughly 200€. The camcorder is a good and cheap option

[3]Twitch is a live streaming platform for gaming and interaction with the chat.

and does a good job.
The last option is cameras which are connected via SDI. They often are called streaming cameras or conference cameras. As said these cameras can be connected with SDI which we want to use to connect to the PC. Some of these cameras offers also an HDMI connector so that I can use both - SDI and HDMI - and test them in terms of transmission speed, quality, and packed loss. But they are also expensive. They start from around 400€ and have no real upper limit.
When speaking of the resolution, a camera with FullHD would be enough for our case because the higher the resolution the more space, and calculation power is needed and the algorithm also would take longer.

To wrap it up. The best option would be a streaming camera which also offers an HDMI connector.

4.1.3 Video capture device

But to actually connect the camera via HDMI or SDI and get the video stream as an input we need a video capture card.

A capture card is connected to a computer and takes a video signal as input from another device and encodes it so that the computer can use it for live streaming or recording [20][21].

Here are two types of connectors. The first is a USB connection and the second is a Peripheral Component Interconnect (PCI) which is connected directly to the motherboard. When there is space on the motherboard a PCI capture card is recommended because it would be much faster with lower latency but when there is no space than a USB connection should also do its job.

4.2 Setup

Finally, we used the *Azure Kinect* from Microsoft because this camera was available in the *TAMS*[4] group. We borrowed and tested this camera and found it adequate. Therefore, we decided to buy one for the *signal processing* group as well. The Azure Kinect has a build in depth sensor to measure the depth in the scene. The camera is connected via USB-C. For the cable, we chose an active USB cable, the PureLink FiberX FX-1530-015. Currently, there is no possible way to connect the camera to the computer like the microphone. Because in the wall box is no plug for USB-C to plug the active USB cable in. But there is a hole in the wall where the cable is led through. The full setup with the microphone and the camera is shown in Fig. 4.

[4] https://tams.informatik.uni-hamburg.de

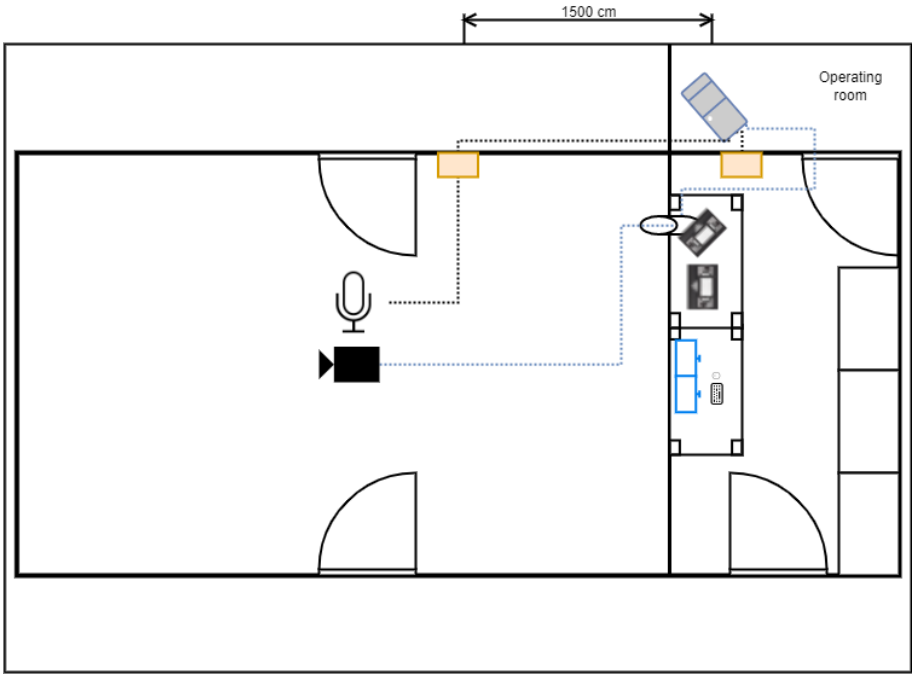

Figure 4. Acoustic lab microphone and camera setup. The microphone has the same connection in Fig. 2. The camera is connected to the computer with a USB-C cable which is led throug a hole connecting the recording room with the computer room represented with a blue dotted line. **(non-scale drawing)**.

5 Audio preprocessing

In this section, I will describe how to acquire the audio stream as an input and how to preprocess it. To this end, I will discuss the various processes involved in processing the audio data.

For the audio preprocessing, I use the *JACK* [22] library. JACK stands for *Jack Audio Connection Kit* and is a professional solution for real-time and low-latency connection for applications. Which is also an open-source library written in C. The Jack library helps to get the audio stream from the microphone to the PC as an input. The Jack library has a callback function called `process(frame)` [22]. The callback is called every 8 ms. The 8 ms comes from the hardware settings. The hardware in our acoustic lab has a sample rate of $f_{s,\mathrm{H}} = 32\,\mathrm{kHz}$ and an audio buffer size of $N_{\mathrm{buffer,H}} = 256$. Therefore,

$$T_{\mathrm{buffer}} = \frac{N_{\mathrm{buffer,H}}}{f_{s,\mathrm{H}}} = \frac{256}{32000\,\mathrm{Hz}} = \frac{1}{125}\,\mathrm{sec} = 8\,\mathrm{ms} \tag{1}$$

every 8 ms there is a new audio frame. The new audio frame can then be accessed in the callback function where the required preprocessing is done and sent back as an output.

For the preprocessing, I have a down and upsampling part. This is required because every audio-visual speech separation model has different sample rates. First, I downsampled the audio by 2. The factor is variable. I choose this downsampling factor 2 because the requirement of the VisualVoice model is $f_{s,VV} = 16\,\mathrm{kHz}$ [8]. After the downsampling is done the DNN can be called in a different process. More about the processes and the DNN call will be mentioned later. After that, the result of the DNN, which is also at a sample rate of 16 kHz must be upsampled again to 32 kHz to match our hardware requirements. After the upsampling is done the results can be heard over the output in our case the speakers.

6 Video preprocessing

Besides the audio, it is also necessary to get the video input as well and also preprocess the video. In this section, I will talk about how I received the video stream as input. And how I preprocessed the video. For the face tracking, I have chosen two different models provided by Google which are available in the *MediaPipe*[5] library. The two models are *Face Detection* and *Face Mesh* which I will explain in detail in the next sections. The face tracking step is necessary because we are only interested in the ROI. Lastly, I will describe my implementation of the preprocessing of the video frames in order to pass it to the DNN model of audio-visual speech separation.

6.1 Face detection method

MediaPipe provides a face tracking model called Face Detection [23]. This model is a fast and lightweight model, which accurately tracks faces and detects also 6 key points. The 6 key points are the ears, eyes, mouth and nose. The Face Detection model is based on the *BlaceFace* [24] model. This model is specially made for mobile Graphics processing units (GPUs). That is why the BlaceFace is a very lightweight face detection model. However, the BlaceFace model has its inspiration from the *MobilNetV1* and *MobilNetV2* [25] model. As the name suggests this visual neural network model is designed to run on mobile devices [23].

The result of the Face Detection model is shown in Fig. 5

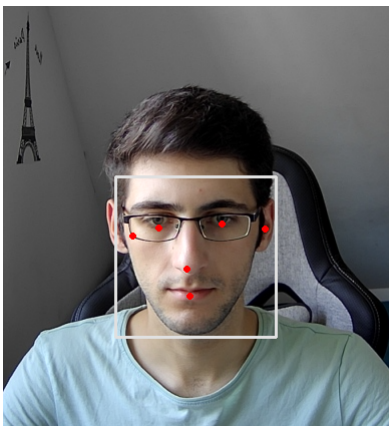

Figure 5. Face Detection model with 6 key points in red and a white rectangle around the face.

[5] https://mediapipe.dev

6.2 Face mesh method

Face Mesh [26] is a machine learning (ML) model from MediaPipe. This model can detect 468 3D landmarks in real-time on faces. There is no need for an extra camera with a depth sensor. A normal camera e.g. a smartphone camera is enough. The Face Mesh model is a lightweight architecture which means that this model also can run on a regular smartphone with no performance issues.

The Face Mesh model consists of two ML models.

- The first model is a face detection model. This model detects the faces in the given scene. The model that they use is the BlazeFace [24] model. The same model that they used in the Face Detection in Section 6.1. They use this model because it is a lightweight model which is also fast. The model returns at the end the detected faces.
- The second ML model is applied to the faces which were detected by the first ML model. That Face Mesh model estimates 468 landmarks on the faces. But not only the 2D coordinates. Afterwards it approximates the depth of that coordinate. So a landmark consists of an x, y and a z coordinate. All 468 landmarks can be projected on the face. An example is shown in Fig. 6. As said the landmarks are in a 3D space. It would look like Fig. 7 if plotted in a 3D space.

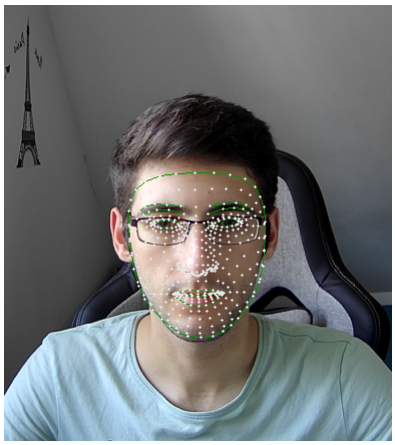

Figure 6. Face Mesh with 468 landmarks. The landmarks on the eyes, eyebrows, mouth and the face are connected by a green line.

The Face Mesh model uses at first the BlazeFace model to detect the face. This is done to improve the performance of the face mesh model. Because then the face mesh algorithm does not have to work on the whole scene, which would be more computationally intense and therefore could

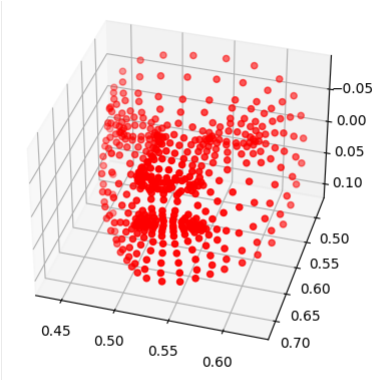

Figure 7. FaceMesh in a 3D space. Each red dot is a landmark.

increase the calculation and it could be that the model then can not fulfil the real-time criteria. That is why the first ML model is a face detection model so the landmarks can be calculated only on the face to predict the landmarks. Another advantage is that the Face Mesh model does not need to care about a lot of noise or scale, transformation etc. problems. This model can also handle partial occlusions or grimaces and predicted the landmarks pretty well. Two examples with occlusions are shown in Fig. 8 and Fig. 9 [26].

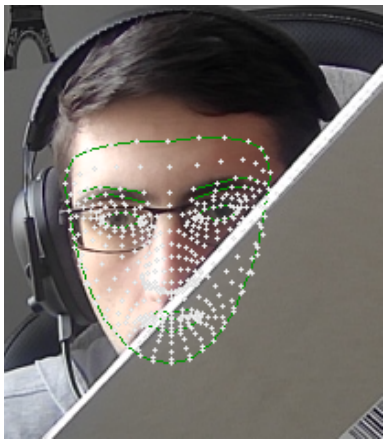

Figure 8. FaceMesh landmark prediction with partial occlusion.

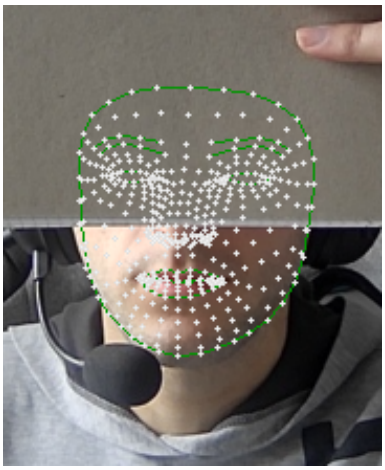

Figure 9. FaceMesh landmark prediction with partial occlusion.

6.3 Implementation

As seen in the two sections before both models of MediaPipe are two good choices. However, I decided to choose Face Mesh because it provides also the depth information of the landmarks. The depth information of the landmarks could be used to transform the lip position in space. This way the lips could be transformed, which are captured from different angles in a front-view perspective. The benefit of doing such a transformation is that the DNN model has a clear front view of the lips. For the depth information is no need for a specific camera because the depth is approximated. However, the approximated depth is not necessary anymore because of the use of the Azure Kinnect camera, mentioned in Section 4.2 which already has a built-in depth sensor. The depth sensor can be used with the *Sensor-SDK*[6].

For the implementation, I used the OpenCV library which I used to get the video frame of the camera in order to manipulate or display it. For the face tracking, I used mediapipe with Face Mesh and for the manipulation of the video frame I used numpy. For the face mesh to work, I need the video frame in the RGB color space. After that, I initialized the key parameter of the Face Mesh model. The key parameter are how many faces I want to detect and how accurate the detection should be. To get the landmarks of the faces I called the Face Mesh model. The model call returns me a list of a length of two where in each sublist the landmarks of the faces are. The list has a length of two because I set the key parameter of the faces that I want to track to two. The landmarks of the two faces are then drawn on the face which is shown in Fig. 10.

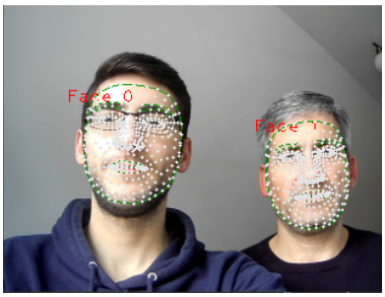

Figure 10. Landmarks on two face with the corresponding face number.

To extract the ROI of each face the function `cropLips()` is called. In order to crop the lips I loop over the lip landmarks and took the minimum and maximum values in the x and y coordinates to get a rectangle around the lips. The rectangle is visualized in Fig. 11 and Fig. 12. After this operation, the ROI is crop into an 88 x 88 sized frame and save it in a separate frame and also put the result into a queue. For each face, there is one queue. More

[6] https://learn.microsoft.com/de-de/azure/kinect-dk/sensor-sdk-download

about queues in a bit. I choose the frame size to be 88x88 because the input of the ROI in VisualVoice is 88 x 88. But the values can be customized to the need of the DNN model. The frame of the cropped lips is displayed in Fig. 13 and Fig. 14.

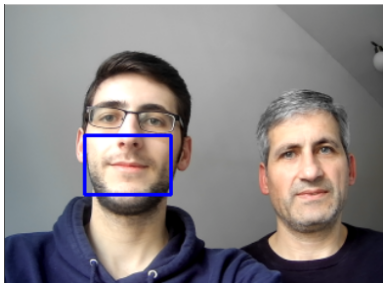

Figure 11. Rectangle in blue around the lips of face 1.

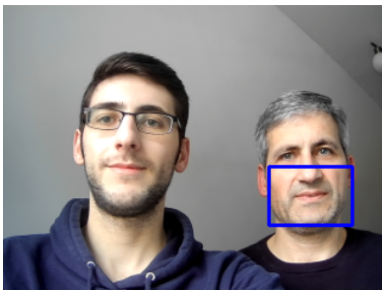

Figure 12. Rectangle in blue around the lips of face 2.

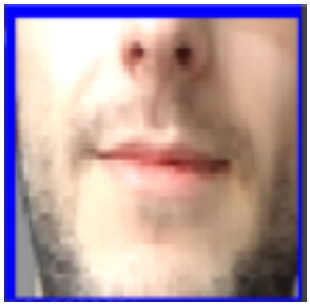

Figure 13. Cropped lips of face 1 with a size of 88 x 88

Figure 14. Cropped lips of face 2 with a size of 88 x 88

7 Combining the two preprocessing steps

In this section, I am going to discuss how the two modalities, audio and video, discussed in Section 5 and 6, are combined for multimodal processing. This is important because we have two continuous data streams. The acquisition and processing of both streams must run in parallel so there is no loss of data. First, I will describe the possible ways to run multiple tasks in parallel in Python. Then, I will explain the alignment of the audio and video in order to pass it to the DNN model.

7.1 Multithreading vs. Multiprocessing

During my research, I came across the concepts of *multithreading* and *multiprocessing*. Both techniques are used in computer programming to achieve parallelism and improve the efficiency of a program's execution. The following subsections outline the basics of both methods and their respective advantages and disadvantages.

7.1.1 Multithreading

A *Thread* is a sequence of instructions of a running program. Multithreading is a technique to run multiple threads concurrently within a single process. Multithreading has a shared memory and can easily communicate with each other because they are in one process [27]. Fig. 15 shows an illustration of how multithreading can be understood.

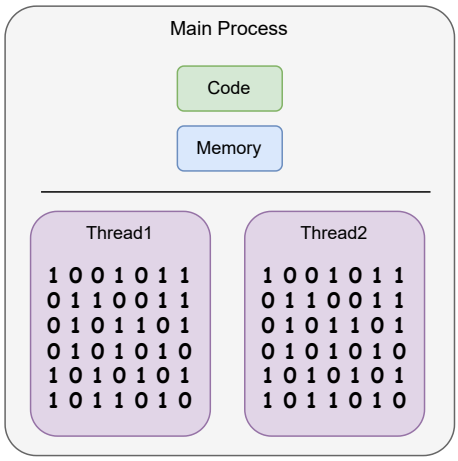

Figure 15. Multithreading illustration. The gray rectangle is the process which has a code and a memory. In this case, there are two threads illustrated in two purple rectangles.

In Python, multithreading can be implemented using the threading

module[7]. Threads in multithreading run simultaneously, however, the threads do not run parallel. When a function is called in a thread, the other thread must wait for the call to finish before continuing the execution. To prevent that a thread blocks all other threads at all times the Operation System (OS) switches the threads after a short period of time, as illustrated in Fig. 16. That means that in a thread a small part of the code is executed and then it switches to another thread. That way each thread has a chance to execute.

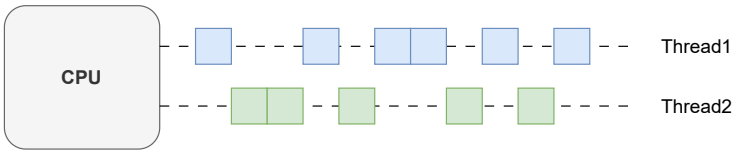

Figure 16. Execution of multithreading. The Central processing unit (CPU) in this case has one process. The process running on the CPU has two threads the top one in blue and the second one on the bottom in green. The rectangles on the dotted line represent the execution process of the thread.

Of course, it could be possible to run multithreading in parallel. This happens when the threads are on different processes which would be the case if the computer has a multicore architecture. But Python uses a Global Interpreter Lock (GIL) to prevent that more than one thread is running in parallel [28].

Multithreading is often used on servers to balance the traffic on the servers. But also in Graphical User Interface (GUI) applications and I/O tasks. Multithreading allows the application to listen for user interactions with the User Interface (UI) [28][29][30].

7.1.2 Multiprocessing

The approach of multiprocessing is to run multiple processes on separate processors. Fig. 17 shows how the execution of multiprocessing is performed. Multiprocessing in Python can be implemented using the multiprocessing module[8] The advantage of this approach is that multiple processes can run in parallel at the same time without interfering with each other. A disadvantage of multiprocessing is that they have no shared memory like multithreading. This is because the processes run on different processors and they have separate memories. An illustration of multiprocessing is shown in Fig. 18. However, there are ways to share data between processes. *Queues*, *Pipes* and *Arrays* are just a few example. In this project, I use Queues because it was easy to implement and there was no restriction in the data type or size like in Array [27][30].

[7] https://docs.python.org/3/library/threading.html
[8] https://docs.python.org/3/library/multiprocessing.html

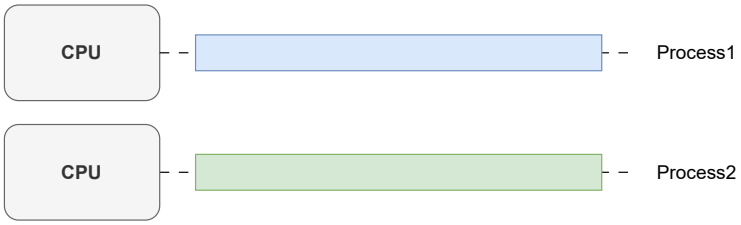

Figure 17. Execution of multiprocessing. There are two CPUs, and each of them has one process. The top one is in blue and the second one on the bottom in green. The rectangle on the dotted line represents the execution processing of the processes.

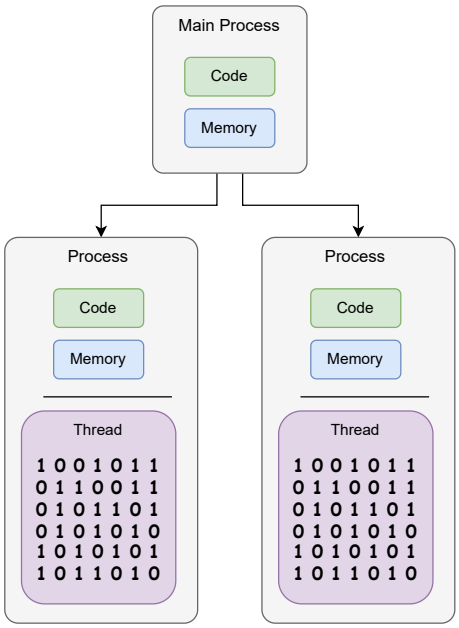

Figure 18. Multiprocessing illustration. The *Main Process* on the top creates two new processes which are below. In each process the is the code (green) and a memory (blue) and the corresponding thread (purple) which runs in the process. Between the two processes on the bottom are two arrows which represent the communication between the two processes.

7.2 Combination of both modalities to run in parallel

To combine the two modalities, I use multiprocessing because it allows the execution of the preprocessing steps in parallel. Multiprocessing has no shared memory like multithreading. So in order to communicate with the DNN process, I leverage the use of *Queues*. In total there are four queues. The first one is the *audioInQueue*. The second one is the *audioOutQueue* and the last two are *videoFrameFace1* and *videoFrameFace2* which are used for the two detected faces. The two preprocessing steps of the audio and video work in real-time. In order to run the preprocessing steps in parallel, I have one process for the video preprocessing where the faces are detected and the ROI is cropped. The second process is audio preprocessing where the audio is read in from the input device and downsampled for the DNN model to process and upsampled again to match the hardware requirements of our acoustic lab. There is another process for the DNN call of VisualVoice. This process collects the audio samples and video frames to start the DNN model and delivers an output. Unfortunately, VisualVoice is not real-time capable, because the DNN model takes about 100 - 200 ms to execute and deliver an output. To achieve a near real-time experience, I added a synthetic delay. So the audio is buffered and processed and the result is played with a delay of less than one second. In order to add a synthetic delay, I use two buffers one buffer is in the DNN call. This buffer is called *audioBufferIn*. audioBufferIn has a size of $N_{\text{audioBufferIn}} = 40800$ audio samples initialized with zeros. The buffer has a size of 40800 audio samples because VisualVoice requires an input of $T_{in,\text{VV}} = 2.55$ seconds. Since the operating sampling rate of VisualVoice is $f_{s,\text{VV}} = 16\,\text{kHz}$, we have

$$\begin{aligned} N_{\text{audioBufferIn}} &= f_{s,\text{VV}} \cdot T_{\text{in,VV}} && (2)\\ &= 16000\,\text{Hz} \cdot 2.55\,\text{sec} && (3)\\ &= 40800\,. && (4) \end{aligned}$$

The audioBufferIn is illustrated in Fig. 19. Each rectangle represents an audio frame.

The second buffer is called *audioBufferOut* which is located in the audio preprocessing. This buffer collects the output delivered by the DNN model to play the audio. The size of this buffer plays a significant role in the synthetic delay. This buffer is shown in Fig. 20 and is not any different from the audioBufferIn. It is also initialized with zeros.

In the audio preprocessing, a new audio frame is received by the Jack callback and downsampled before being put into the audioInQueue. In the DNN process, this audio frame is then appended to the audioBufferIn. More precisely, the first audio frame is dequeued (removed) from the audioBufferIn and the new audio frame is enqueued (added) at the back of the buffer. This

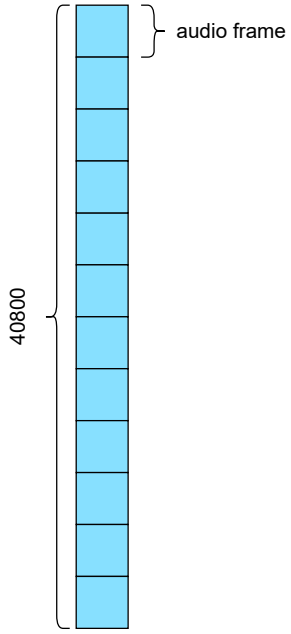

Figure 19. audioBufferIn with a size of 40800 audio samples. Each rectangle represents an audio frame (not scaled).

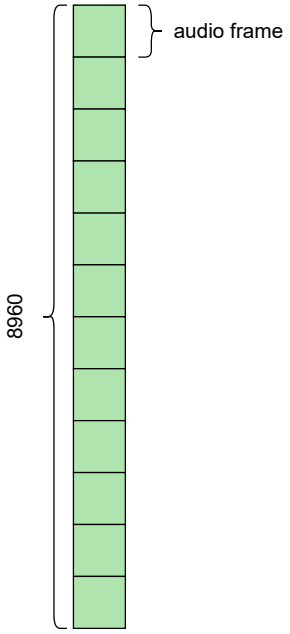

Figure 20. audioBufferOut with a size of 8960 audio samples. Each rectangle represents an audio frame (not scaled).

step continues till we reached a specific amount of new audio samples. This buffering procedure is introduced because the execution of the DNN takes longer than the frame period $T_{\text{buffer}} = 8\,\text{ms}$. The number of buffered frames needed before the DNN is executed is called *count* and is determined by the maximum time $T_{\text{DNN,max}}$ the DNN model takes to finish the process and an extra buffer time T_{extra} and the frame period T_{buffer}:

$$count = \left\lceil \frac{T_{\text{DNN,max}} + T_{\text{extra}}}{T_{\text{buffer}}} \right\rceil \tag{5}$$

$$= \left\lceil \frac{200\,\text{ms} + 80\,\text{ms}}{8\,\text{ms}} \right\rceil \tag{6}$$

$$= 35 \tag{7}$$

If the amount is reached the DNN model call starts. Empirical measurements revealed that the execution of the DNN takes approximately 100 - 200 ms. The new audio frames are continuously put in the audioInQueue. After the DNN call is finished the audioBufferIn is again getting the audio frames in the audioInQueue and shifts the new audio frames in the audioBufferIn. The step is illustrated in Fig. 21.

After a DNN call is finished. The last 35 audio frames are taken and put in the audioOutQueue so the audio preprocessing can get them and sample

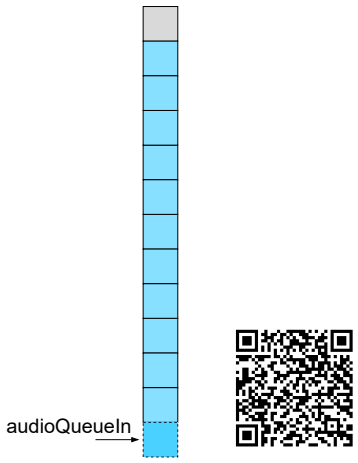

Figure 21. Shifting of the new audio frames. The new audio frame comes from the audioQueueIn and is added at the end on the bottom. After that the first audio frame is removed shown in gray (not scaled). Link to the animated image[9].

them up and add them to the audioBufferOut. We only take the last 35 audio frames because that are also the new audio frames that are processed from the DNN model the ones before are already been processed. In the next step the new audio frames are added to the audioBufferIn and the old ones are removed. Then the DNN call returns the result back and only the 35 audio frames or 4480 audio samples are new so we only need them.

During the whole process in the DNN call the audio in the audio processing is outputted continues. For that, there is the audioBufferOut. This buffer gets filled every time after the DNN call finishes. The result of the DNN call is then added at the end of the audioBufferOut. The initial size of the audioBufferOut $N_{\text{audioBufferOut}}$ has the same amount of audio frame as new audio frames get in. The DNN call takes approximately 200 ms to return a result. We ensure that the DNN call is finished before all audio frames in the audioBufferOut are outputted because there is an 80 ms extra time added. Therefore,

$$N_{\text{audioBufferOut}} = count \cdot N_{\text{buffer,H}} \tag{8}$$
$$= 35 \cdot 256 \tag{9}$$
$$= 8960\,. \tag{10}$$

So the audioBufferOut has a size of 35 audio frames or 8960 audio samples. If we would choose the audioBufferOut too small the code will fail. Because the DNN model is not finished yet but the audio preprocessing is continuously reading the audio frames from the audioBufferOut. If the audioBufferOut is

[9] https://github.com/berkkan22/audio-visual-speech-separation/blob/main/images/shifting.mp4

too big the delay would be much longer than necessary because the audio preprocessing needs to read all zeros that are initially in the audioBufferOut. So we would have several new audio frames coming from the DNN and added to the audioBufferOut but not all the zeros are outputted yet.

So it is important to find the right size for the audioBufferOut. So that the buffer is not too big to have an unnecessary delay but also not too small to run into an error.

The whole process of the buffers is illustrated in Fig. 22. On the left is the audioBufferIn which gets the new audio frames. After a specific size is reached the DNN call starts and delivers an output. The output is then added at the end of the audioBufferOut.

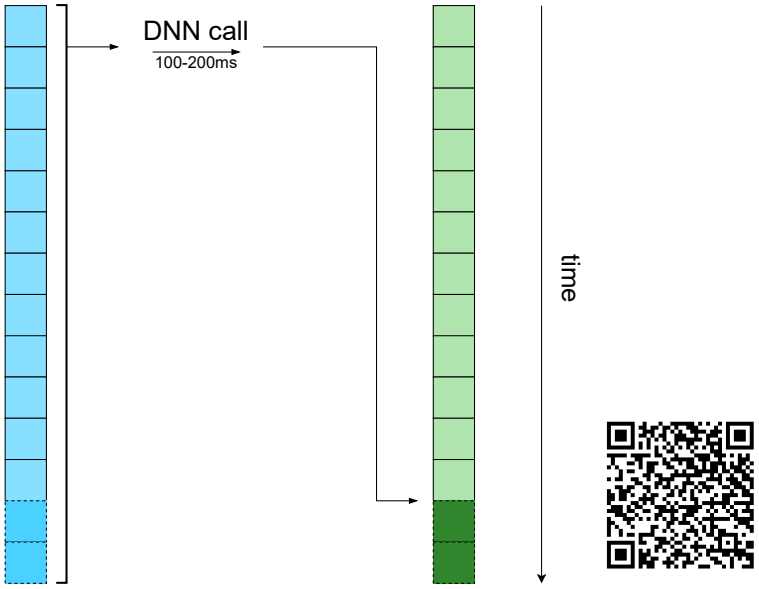

Figure 22. Getting the audio frames and adding them to the audioBufferOut. The DNN call take the whole audioBufferIn and starts the processing. After the DNN model is done the output is added at the end on the bottom to the audioBufferOut (not scaled). Link to the animated image[10].

[10] https://github.com/berkkan22/audio-visual-speech-separation/blob/main/images/dnn_call.mp4

8 Limitations

Throughout the implementation process, various challenges were encountered, and while certain issues were successfully addressed, others proved to be difficult to overcome.

The VisualVoice DNN model is successfully integrated and also called to process the input data. The results are also send back to the audio preprocessing to play. However, the DNN separation estimates leave the mixture audio unchanged. This phenomenon could be attributed to a discrepancy between the training and testing phases, a concern that could be remedied in subsequent research endeavors.

The second problem is that the Queues are not always communicating properly with each other. The problem appeared after saying that the process start method should be `spawn`. Although the script may eventually function after a few runs, a resolution to this problem remains elusive.

The last problem is that there is an audio loss in the input audio. A recording can be heared here[11]. However, this problem was solved. The problem was that the audio frames inserted into the queue were too large. Because at first I collected all 40800 audio samples in the audio preprocessing and send everything at once which was apparently too memory-consuming. The solution was to put smaller audio samples and send them. So I put every audio frame which is 128 audio samples big in the Queue and collected them in the audioBufferIn in the DNN call (see Section 7.2). After fixing the problem, no audio loss occurs.

[11] https://github.com/berkkan22/audio-visual-speech-separation/blob/main/images/audio_drops_problem.aac

9 Conclusion

In this bachelor thesis, I addressed the problem of the real-time capability of audio-visual speech separation. I have first motivated that by including the video information the separation of speakers achieved better results compared to audio-only speech separation. In addition, audio-visual speech separation also solves the permutation problem.

The main focus of this bachelor thesis was to develop a real-time preprocessing module for audio-visual speech separation. I have shown how the two sensory data audio and video are read in. Furthermore, I have discussed how the two modalities need to be processed to form the input of an audio-visual speech separation model. Specifically, an issue arose from the interaction between the two modalities. In particular, the simultaneous processing of sensory data posed a challenge during the developmental stage. Nonetheless, I was successfully solving that by using Python's multiprocessing library to run the two processing steps in parallel. As a result, the preprocessing module works in real-time. However, to the best of my knowledge, there is currently no publicly available code for a real-time capable audio-visual speech separation model. To get an approximate real-time experience, I added a synthetic delay to give the DNN model time to process the input and deliver the output with a delay of less than one second.

In the end, I was able to develop a preprocessing module that works in real-time. The state-of-the-art DNN model VisualVoice is connected after the preprocessing module and starts processing the inputs.

The working code can be found in the repository *audio-visual-speech-separation*[12] on my GitHub.

9.1 Future work

The code is working but there is still improvement to make. One improvement is to use the Azure Kinect SDK. This SDK then can be used to get the depth information from the camera to transform the lips in space to get a front view. Furthermore, the problems mentioned in Section 8 should be solved to make the script more robust.

Furthermore, other audio-visual speech separation DNN models could be used to decrease the introduced synthetic delay and to compare their respective separation performances.

[12] https://github.com/berkkan22/audio-visual-speech-separation

References

[1] E Colin Cherry. Some experiments on the recognition of speech, with one and with two ears. *J. Acoust. Soc. Am.*, 25(5):975–979, 1953.

[2] Jiquan Ngiam, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee, and Andrew Y Ng. Multimodal deep learning. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 689–696, 2011.

[3] Daniel Michelsanti, Zheng-Hua Tan, Shi-Xiong Zhang, Yong Xu, Meng Yu, Dong Yu, and Jesper Jensen. An overview of deep-learning-based audio-visual speech enhancement and separation. *IEEE ACM Trans. Audio Speech Lang. Process.*, 29:1368–1396, 2021.

[4] Triantafyllos Afouras, Joon Son Chung, and Andrew Zisserman. The conversation: Deep audio-visual speech enhancement. *Proc. Interspeech 2018*, pages 3244–3248, 2018.

[5] Dong Yu, Morten Kolbæk, Zheng-Hua Tan, and Jesper Jensen. Permutation invariant training of deep models for speaker-independent multi-talker speech separation. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 241–245. IEEE, 2017.

[6] Mandar Gogate, Kia Dashtipour, and Amir Hussain. Towards robust real-time audio-visual speech enhancement. *arXiv preprint arXiv:2112.09060*, 2021.

[7] Nicholas A Lesica. Why do hearing aids fail to restore normal auditory perception? *Trends Neurosci.*, 41(4):174–185, 2018.

[8] Ruohan Gao and Kristen Grauman. Visualvoice: Audio-visual speech separation with cross-modal consistency. In *CVPR*, 2021.

[9] Emmanuel Vincent, Rémi Gribonval, and Cédric Févotte. Performance measurement in blind audio source separation. *IEEE transactions on audio, speech, and language processing*, 14(4):1462–1469, 2006.

[10] A.W. Rix, J.G. Beerends, M.P. Hollier, and A.P. Hekstra. Perceptual evaluation of speech quality (PESQ)-a new method for speech quality assessment of telephone networks and codecs. In *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.01CH37221)*, volume 2, pages 749–752 vol.2, 2001.

[11] Cees H Taal, Richard C Hendriks, Richard Heusdens, and Jesper Jensen. An algorithm for intelligibility prediction of time–frequency weighted

noisy speech. *IEEE Transactions on Audio, Speech, and Language Processing*, 19(7):2125–2136, 2011.

[12] Cem Akkus, Luyang Chu, Vladana Djakovic, Steffen Jauch-Walser, Philipp Koch, Giacomo Loss, Christopher Marquardt, Marco Moldovan, Nadja Sauter, Maximilian Schneider, et al. Multimodal deep learning. *arXiv preprint arXiv:2301.04856*, 2023.

[13] Meilhaus Electronic GmbH. Erweitern der USB 3.0 Übertragungsdistanz. https://www.meilhaus.de/news-aktionen/blog/blog-icron-usb30/?b2b=0. Accessed: 2022-8-23.

[14] Paul Jensen. Zulässige Kabellängen für unterschiedliche Schnittstellentypen. https://wiki.inonet.com/kabel-schnittstellen. Accessed: 2022-8-23.

[15] HDMI Kabel – 4K / HDMI 2.1 – wie lang kann ein HDMI Kabel sein? https://www.heimkino-boutique.de/allgemein/hdmi-kabel-4k-hdmi-2-1-wie-lang-kann-ein-hdmi-kabel-sein/. Accessed: 2022-8-23.

[16] Andrew Froehlich. What is serial digital interface? https://www.techtarget.com/searchnetworking/definition/Serial-Digital-Interface, September 2021. Accessed: 2022-8-23.

[17] Lisa. SDI cable: All you need to know how to choose the cable you need. https://www.wiringo.com/sdi-cable-all-you-need-to-know-how-to-choose-the-cable-you-need.html, October 2019. Accessed: 2022-8-23.

[18] Blackmagic Design. DeckLink. https://www.blackmagicdesign.com/at/products/decklink. Accessed: 2023-3-15.

[19] Christian Karasiewicz. What is clean HDMI? why does it matter for my live stream? https://streamyard.com/blog/live-streaming-equipment/clean-hdmi-for-live-streams/, July 2021. Accessed: 2022-8-23.

[20] John Awa-abuon. What is a capture card and how does it work? https://www.makeuseof.com/what-is-a-capture-card-how-does-it-work/, June 2022. Accessed: 2022-8-23.

[21] Chris Heckmann. What is a capture card for streaming and how to use one. https://www.studiobinder.com/blog/what-is-a-capture-card-for-streaming/, November 2020. Accessed: 2022-8-23.

[22] Jackaudio. https://jackaudio.org. Accessed: 2023-3-10.

[23] Valentin Bazarevsky, Yury Kartynnik, Andrey Vakunov, Karthik Raveendran, and Matthias Grundmann. Blazeface: Sub-millisecond neural face detection on mobile gpus, 2019.

[24] Valentin Bazarevsky, Yury Kartynnik, Andrey Vakunov, Karthik Raveendran, and Matthias Grundmann. Blazeface: Sub-millisecond neural face detection on mobile gpus, 2019.

[25] MobileNetV2: The next generation of on-device computer vision networks. https://ai.googleblog.com/2018/04/mobilenetv2-next-generation-of-on.html. Accessed: 2023-3-9.

[26] Yury Kartynnik, Artsiom Ablavatski, Ivan Grishchenko, and Matthias Grundmann. Real-time facial surface geometry from monocular video on mobile gpus, 2019.

[27] Molly Clancy. What's the diff: Programs, processes, and threads. https://www.backblaze.com/blog/whats-the-diff-programs-processes-and-threads/, November 2022. Accessed: 2023-3-6.

[28] Michael Weigend. *Python 3: Lernen und professionell anwenden. Das umfassende Praxisbuch.* 2019.

[29] John Goerzen, Tim Bower, and Brandon Rhodes. *Foundations of Python Network Programming: The comprehensive guide to building network applications with Python.* APress, Berlin, 2 edition, 2010.

[30] Kay Jan Wong. Multithreading and multiprocessing in 10 minutes. https://towardsdatascience.com/multithreading-and-multiprocessing-in-10-minutes-20d9b3c6a867, March 2022. Accessed: 2023-3-6.

Eidesstattliche Erklärung

7298823
Martikelnummer

Katirci, Berkkan
Name, Vorname

Hiermit versichere ich an Eides statt, dass ich die Arbeit eigenständig verfasst habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen und die eingereichte schriftliche Fassung entspricht der auf dem elektronischen Speichermedium.

Hamburg, den 17.03.2023
Ort, Datum

B. Katirci
Unterschrift