EE434 Project Design and Outcomes

Matt Baseheart, Yuzhou He, Michael Pozzi

May 9, 2021

1 Abstract

The goal of this project is to use visual and audio information to perform real time processing in order to improve the quality of the audio with respect to a speaker's position in a limited environment. The device is designed to be set up at a personal workstation in a room, office, or study space and will utilize a camera and microphone system to collect data to inform audio corrections. The device will perform ambient noise reduction to limit noise above and below the human voice range. The device will also employ an equalization filter in order to improve the quality of the audio coming from the microphone. As for video, the camera will perform facial detection, tracking, distance detection to use for audio correction, pan/tilt to follow the subject's face, and a digital zoom to keep the subject's face size more constant.

2 System Design

2.1 Modules

The software consists of both an audio processing chain and a video processing chain. The audio chain includes a voice activity detection (VAD) module, a denoiser module, and an equalizer (EQ) module. The video chain consists of a facial detection module, feature location processing module, pan/tilt adjustment module, digital zoom module, and distance detection module. The distance information will be sent to the audio DSP chain to control the volume of the audio output. The back end of the software is coded in Python, and the control panel is a web page held by Flask server. Depending on the operation scenario, the software can be used for different purposes. The user can: 1) host the Flask server locally on devices like Jetson Nano or personal computers to do all the processing (See Figure 1 below), or 2) host the Flask server on a separate server (preferably GPU server), access the webpage using browser, use WebSocket to communicate data and do all the processing remotely (See Figure 2 below). Mode 2 also allows multiple people to access the application simultaneously through browsers, as long as the computation power of the cloud server allows.

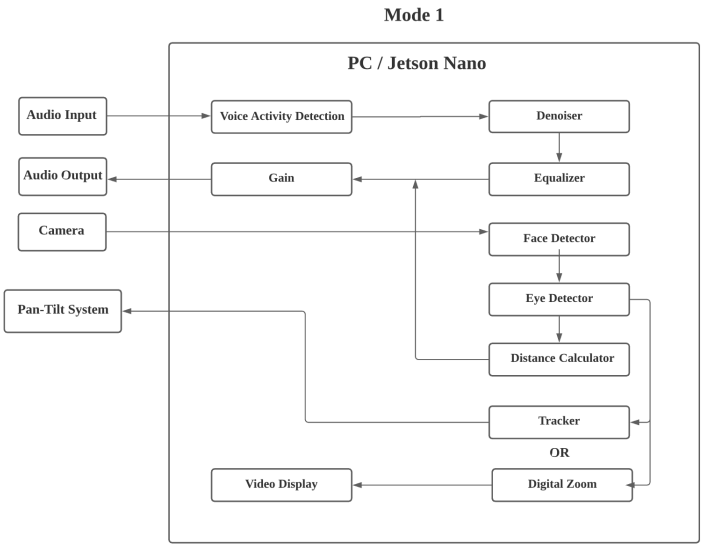

Figure 1: Mode 1: All processing and display done on the Jetson

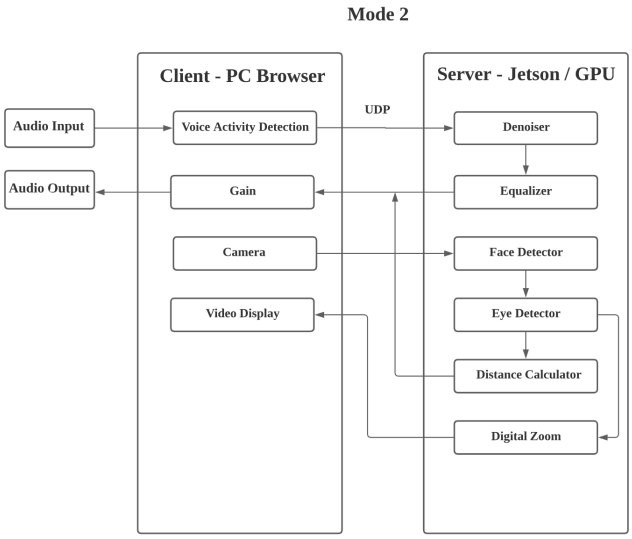

Figure 2: Mode 2: Processing done on Jetson and sent to separate device in order to display.

The user can change the input and output devices in the application, while the default audio input and output are a microphone and speakers attached to local devices. If the user is using a meeting software, they can switch the audio output device to audio feedback interfaces like Soundflower. Python can stream NumPy arrays into feedback interfaces and feed them into meeting software, like Zoom. The video input can be a pan-tilt camera, a webcam, or simply the computer camera. If a pan-tilt camera is used, it will track the user's face and keep it at the center of the frame and use digital zoom. If not using the pan-tilt camera, then no digital zoom will be implemented. If the application is run in a remote mode, the user input can be switched to microphone, camera, and speaker of the browser. If the user would like to run the software in non-real time, they can save the audio and video output into files instead.

The control panel allows the user to control the VAD module, denoiser type, denoiser level, on/off of the denoiser, equalizer parameter, and display the current EQ response in real time. Figure 3 below shows the content in the project's website prototype.

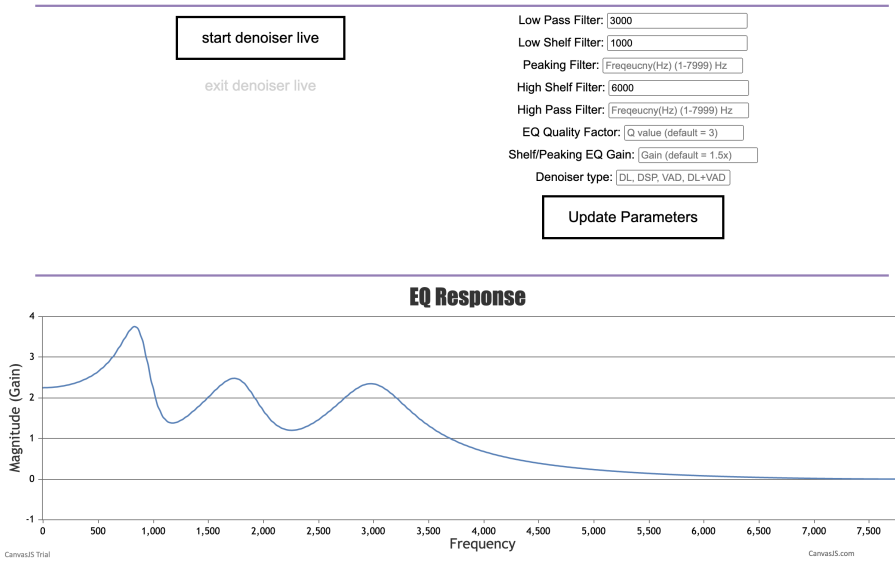

Figure 3: The interface on the Flask server, which can be controlled via web browser.

2.2 Hardware

2.2.1 Nvidia Jetson

The cornerstone of the project is the Nvidia Jetson Nano. The Jetson Nano is designed to compute performance in order to run modern AI workloads. It is supported byNVIDIA JetPack, which includes a board support package, Linux

OS, NVIDIA CUDA, cuDNN, and TensorRT software libraries for deep learning, computer vision, GPU computing, multimedia processing, etc. The version of the Jetson that the team develped on was the 4GB version.

2.2.2 Pan Tilt Camera

Due to the constraints of the project team's locations and discrepancies in members' locations, a consistent camera was not able to be used for testing and implementation. The camera that is connected to the Jetson is part of an Arudcam Pan Tilt Platform (camera) [2]. (See Figures 4 and 5). The camera included in the Pan Tilt Platform has an IMX219 sensor with a field of view of 62.2 degrees [12]. The system displays video at 24 frames per second post-processing. The Pan Tilt system includes two GH-S37D digital servos and a PWM Control Board that is I2C controlled [2].

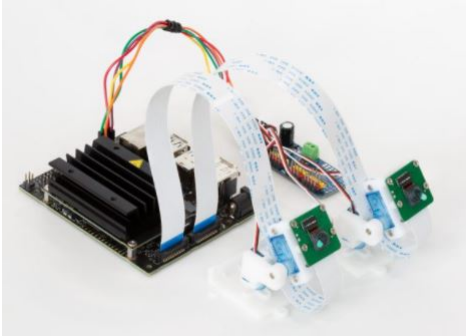

Figure 4: Arducam Pan/Tilt System set up with Nvidia Jetson Nano in Dual Camera Format. The team's project only utilized a single camera and Pan-Tilt system along with the given camera.

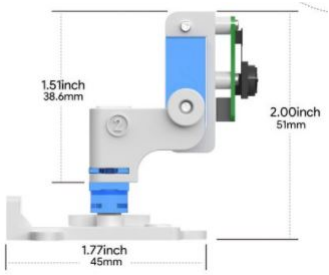

Figure 5: Closer look at the Pan-Tilt system along with dual servos for tracking the facial features of the subject and keeping the subject's face in the center of the frame.

2.2.3 Microphone

The microphone chosen for the project is the Samson Go USB Microphone. It has a switchable directivity pattern which can be chosen between omnidirectional and cardiod. For our purpose and environment, we implemented the microphone with a cardiod pattern. The microphone samples with 16 bits and at 44.1kHz resolution. It has a smooth, flat frequency response between the frequencies of 20Hz and 18 kHz [10]. (See Figure 6).

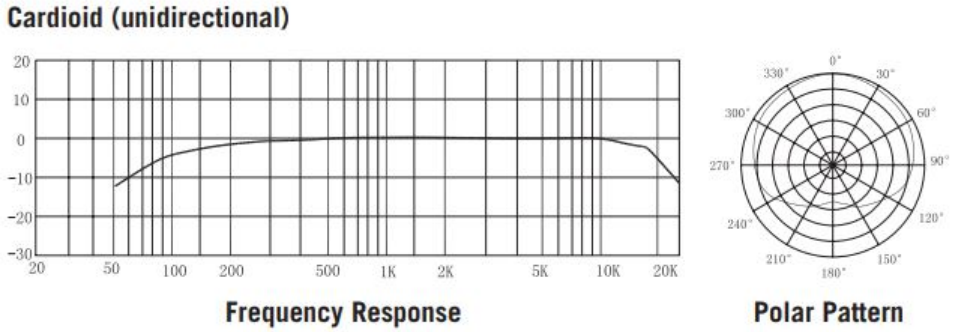

Figure 6: Samson Go USB Microphone frequency response as given by microphone documentation [10]

2.3 Algorithms

2.3.1 Voice Actvity Detection

Our application includes a Voice Activity Detection (VAD) module to control the audio DSP activity. It is the first component in the audio DSP chain after audio is collected from the microphone. When speech activity is detected, the VAD module will pass the input to the rest of the audio DSP chain. If it does not recognize speech from input audio, the input audio will be ignored. It is important to the project team to have it because the deep learning denoiser, which will be introduced in the next section, has a relatively high energy consumption and GPU usage. Because users might use the software for long meetings, this module can close the audio DSP chain when no speech is detected to save computation power. Users can remove VAD modules from the audio DSP chain if the deep learning denoiser is not used, because the designed VAD module itself also requires computation resources and introduces a delay of about 100ms.

The deep learning VAD network consists of a Long Short Term Memory (LSTM) layer (input size = 12, hidden size = 20), a ReLU layer for non-linearity, 2 fully connected layers, and a Softmax layer for activation. The network structure can be seen in the table shown in Figure 7. The input data is firstly sliced

into frames of about 30ms, and a 12-parameter MFCC feature is extracted. After MFCCs are extracted, the 12 input parameters are passed to the LSTM layer, ReLU, fully connected layer, and Softmax layer in order.

Modules	Parameters
rnn.weight_ih_l0	960
rnn.weight_hh_l0	1600
rnn.bias_ih_l0	80
rnn.bias_hh_l0	80
lin1.weight	10400
lin1.bias	26
lin2.weight	52
lin2.bias	2

Total Trainable Params: 13200

Figure 7: LSTM layers and parameters.

The project team trained the LSTM with LibriSpeech ASR corpus (100 hours of clean speech) as speech dataset and QUT-NOISE (including around 80 hours of noises, including café, car, home, reverb, and street as noise dataset [16]. The noise and sound is mixed, and voices are labeled on clean speech with 30ms intervals using a Python VAD library webrtcvad (Google WebRTC project) [17]. Part of the clean speech is mixed with noises, so the VAD works in a noisy environment. The label of the mixed sound is used as the true output of the neural network in the training phase, with MFCCs of input frame as input.

2.3.2 Denoiser

The denoiser is one of the most important components in the audio DSP chain. Depending on the quality of the microphone and noise environment of the meeting, there could be many different kinds of noises. The project's application includes three types of denoisers: two are based on deep neural networks and the other one is based on modified Improved Minimal Controlled Recursive Averaging (IMCRA) method and Optimal Modified Minimum Mean-Square Error Log-Spectral Amplitude (OMLSA) method (OMLSA + IMCRA). All three types provide reliable noise reduction in both stationary and non-stationary noise environments, but they all suit different environments. Depending on

the computation power of the device and denoising level requirement, different denoisers can be chosen. The first deep learning denoiser uses Demucs, while the second uses Deep Complex Convolution Recurrent Network for Phase-Aware Speech Enhancement (DCCRN). The Demucs network operates in time domain, while the DCCRN network operates in frequency domain. Both deep learning denoisers work in real time on a 4-core i5 CPU or Tesla T4 GPU, but they failed to reach real time processing speed on a Jetson Nano GPU. The OMLSA denoiser, on the other hand, can be used in all cases with much less computation and CPU usage.

The Demucs network uses U-Net structure, with skip connections, 5 encoder layers, 5 decoder layers, and a bi-directional LSTM network [7]. (See Figure 8). As a time-domain denoiser, it receives raw audio input data with size (Length, 1) and output processed audio data with size (Length, 1). The input is fed directly into the encoder layer without STFT operation. The encoder is used to downsample the input audio to obtain larger scale features, while the decoder upsamples them back. The encoder consists of two 1D convolution layers with a ReLU in between, and the decoder consists of a 1D convolution layer and a 1D deconvolution layer. Between each frequency scale, there are skip connections that concatenate decoder input with corresponding encoder input and increase the number of input channels, which provide the decoder more features from different scales. The bi-directionoal LSTM layer in the middle helps extract temporal information from the input speech. The Demucs network used in the project is trained on both the DNS dataset and Valentini dataset, and the hidden size is 48 (H=48).

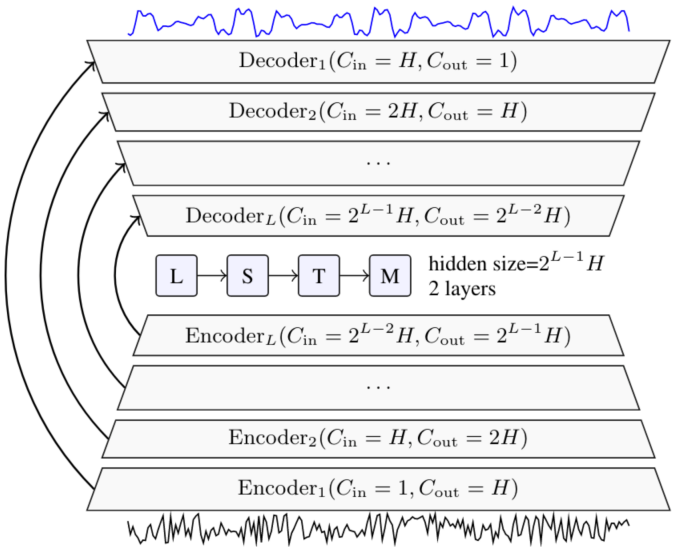

Figure 8: Demucs Structure. [7]

The DCCRN network, on the other hand, operates in frequency domain [11]. (See Figures 9 and 10). The input is fed to a STFT layer first to generate spectrograms. A 1D convolution layer is put at the beginning and the end of the model to compute STFT and iSTFT through convolution. The complex encoder, decoder, and LSTM blocks compute both the real and imaginary parts of the input, and concatenate them in the output as [real, imag]. The complex encoder consists of a complex 2D convolution layer, a complex batch normalization layer and a PReLU. The complex decoder uses a complex transposed 2D convolution instead. The complex LSTM layer used is a combination of two LSTMs; one computes the real part and one computes the imaginary part. After the LSTM layer, there is a dense fully connected layer. Like Demucs, DCCRN also includes skip connections to increase the number of feature channels. Compared with other time-frequency domain implementations, DCCRN computes the complex mask of the input signal, which takes the phase information into account. The DCCRN network is trained on LibriSpeech ASR corpus and WSJ0 Hipster Ambient Mixtures dataset.

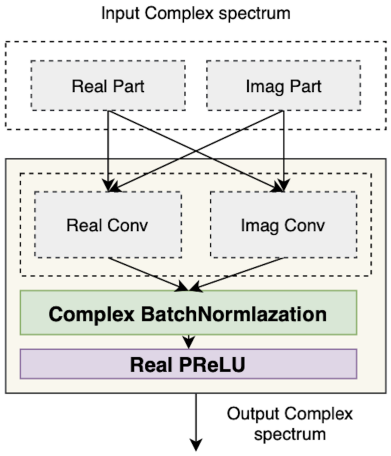

Figure 9: Complex Encoder used in DCCRN structure. Real and imaginary parts of the spectrum are used in convolution. [11]

The team also included an OMLSA + IMCRA denoiser. It is a combination of Optimally Modified Log-Spectral Amplitude estimator (OMLSA) and Improved Minima Controlled Recursive Averaging algorithm (IMCRA). OMLSA estimator minimizes the log spectral differences between clean speech and denoised speech [5]. To produce estimated clean speech, OMLSA estimator requires the calculation of the spectrum of the noise, so the IMCRA algorithm will be used. Firstly, the speech presence probability using minima values of smoothed periodograms is calculated [4]. Then, the past spectral power values are averaged, using time-varying frequency dependent smoothing parameters

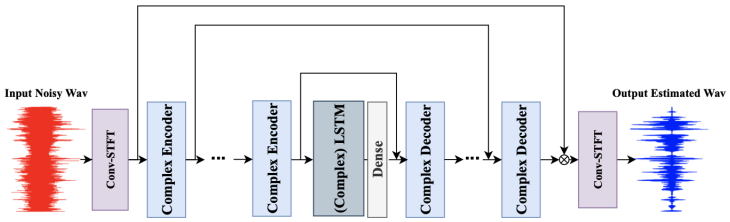

Figure 10: DCCRN Structure. [11]

controlled by speech presence probability, and the result is the power spectrum of the noise, which is the input to the OMLSA estimator. IMCRA is proved to be effective under non-stationary and low SNR situations, and it can remove most of the white noises from the microphone input [4].

2.3.3 EQ

Equalization (EQ) is a process involving increasing or decreasing the presence of certain frequency ranges with respect to the composite signal. This can be achieved using cascaded filters. The incoming audio signal is processed using first a low-pass filter (LPF), then a high-pass filter (HPF), then a Low Shelf filter, then a High Shelf filter, and a Peaking filter. The user may control the center frequency of these filters, though the Q-factors and A-values have been set. The A-value (used in the peaking filters) was selected to be 2 in order to increase gain by a noticeable amount. The Q-factor was selected to be .707 in order to focus the weighting of the signal in a small enough frequency range to achieve noticeable changes. Both of these values were found through experimentation and testing informed by provided equations [6]. The filters were designed by using the bilinear transform on analog High Pass, High Shelf, Peaking, Low Shelf, and Low Pass filters. These individual filters were all Biquad IIR (Infinite impulse response) filters, designed to perform with very little delay.

2.4 Facial Processing

As for the algorithms used for the facial processing, the process flows are as shown in Figure 11. The first step is to take in the frame using OpenCV's libraries and capture function. The next step is to Detect the Face and Eyes. This is done using Dlib, which is a C++ toolkit that must be compiled to be used in Python [13]. Dlib contains machine learning algorithms that can be used for various projects. Dlib has a built-in frontal face detector along with a predictor. The detector collects points along the face and recognizes the shape of the face as a whole. The predictor requires a trained model to be utilized, and the model that the team utilized is called Shape Predictor 5 Landmarks [9]. It is trained on 7,198 faces and is trained to identify the corners of the eyes and the

bottom of the nose as can be seen in Figure 12. It uses these locations to make judgements about where the centers of the eyes are. By using this predictor just for the eyes, we were able to speed up the detection process by not having to detect all features of the face but still get very consistent results. The next step after detection of the face is to process the facial locations. The original locations are saved and the incoming locations of the eyes are subtracted from each other to get the eye width, as well as the facial width. The widths are stored in a NumPy array of size 10 and averaged in order to create a moving average filter to create more consistent measurements and reduce high frequency aspects of the changing facial elements. Before using the widths to adjust the gain, the pan/tilt is adjusted. This utilizes Adafruit's servokit library in order to send commands to the servos regarding how far to shift to keep the center of the face in the center of the display [1]. In order to do so, the angles of the servos is initialized at 90 degrees for pan and 45 degrees for tilt. The center of the face is derived from the corners and widths of the given parameters from Dlib's facial detection. The error away from the center of the frame is then calculated, and if the calculated error is greater than 15 pixels, the pan and tilt values are adjusted to the desired angle, and those angles are sent to the servos. The face is then back in the center of the display. After every 30 frames, the face width average is saved. If the width of the face at a current frame is more than a 25 percent difference of the face check value, then this implies a change in the distance of the face to the camera, and the eye average is used to calculate the distance to the camera based on the camera parameters and the average eye width. This distance is then used to adjust the gain of the incoming microphone signal. Before the display can be shown, a digital zoom is implemented in order to keep a more consistent facial size within the frame. The average eye width is used to adjust the zoom. The program looks at the width of the eyes and bases the zoom on various steps that are implemented based on observation. If the eye distance reaches a certain step, then the frame is cropped and then resized using cubic interpolation in order to maintain some quality within the displayed frame.

Figure 11: Pipeline of video processing utilized in the team's project. The process is initialized at the top when the framework receives a frame from the camera and then proceeds to perform the processing techniques before displaying the frame (either on the Jetson's display directly or another streaming platform) and then receiving the next frame and repeating the process.

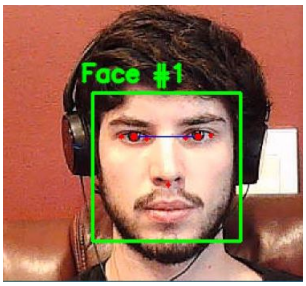

Figure 12: Camera capture showing the points detected using Dlib's facial detection as well as predictor. The facial detection is used to draw the box around the face, while the predictor is used to draw points on the corners of the eyes and the tip of the nose.

3 Outcomes

3.1 Concepts and Tools

3.1.1 Embedded Systems and IOT

The use of embedded systems and the broadly defined "Internet of Things" is heavily involved in this project and probably also led to the most issues. The major components to the project are the development on the Nvidia Jetson, as well as concepts such as UDP streaming and Flask web frameworks, which were introduced in such classes such as "Embedded Systems" and "The Internet of Things." The 5 Layer Network Model serves as the basis of this class and such topics were introduced throughout the semester that became necessary to use in this project when it came to communicating between various devices and web servers from the Jetson. Though the class mainly focused on the use of the Raspberry Pi, the same Linux based concepts and communication tools were transferred over to the Jetson.

3.1.2 Linear Algebra

"Introduction to Linear Algebra" was one of the cornerstone courses that was built upon in the development of this project. The introduction of data manipulating concepts such as matrix calculation, Fourier analysis, clustering, and other concepts were all introduced in a very theoretical manner, but applied throughout this entire project and in every aspect, including both audio and visual format. Audio data in this project was analyzed both in the time and frequency domains. Fourier analysis was used to assist in visualization of the frequency spectrum of audio data before and after Equalization and Noise-cancellation processes. Since the human brain cannot pinpoint frequency range or concretely quantify differences in sound levels, this visualization step was essential to understanding both patterns in typical speech data as well as eval-

uating the success of the filtering techniques being tested. Further, filtering in the frequency domain heavily relied on topics such as convolution. Video data can be looked at simply as a matrix of digits corresponding to different channels of colors. Though the brain perceives this as a continuous spectrum of information, the computer only sees discrete values. Therefore, any form of processing utilizes linear algebra as introduced in the team's undergraduate course. This includes simple concepts such as matrix subtraction to more complex ideas such as convolution, which in its essence is a dot product. These concepts were also built upon and leveraged into much more complex ideas such as neural networks.

3.1.3 Programming

One of the first classes introduced to team members while undergoing their undergraduate program was the "Introduction to Programming" course. This course centered around C++, and though the project was mostly written in Python, the same thinking processes were picked up when introduced to the concept of programming in general. In fact, in a majority of classes, team members learned how essential it is to learn how to think algorithmically through computer programming, and each bit of knowledge was applied practically to this project.

3.1.4 Digital Signal Processing

When considering "Real Time" (minimum delay) systems, it is incredibly important to understand the simplest and most efficient way to achieve a desired processing chain. In the course "Introduction to Digital Signal Processing," information was provided about designing filters based on application. The basics of time and frequency domains were introduced and built upon to introduce very important concepts such as the Fourier Transform, and specifically the Short Time Fourier Transform (STFT) which was heavily used in the audio chain of this project. That was applied to this project in the selection of filter type as well as in understanding bilinear transforms, cascaded systems, and filter forms.

3.2 Implementation, Trade-offs, and Issues

3.2.1 Jetson

The Jetson was made primarily available to the team, and upon initial inspection was believed to be quite suitable to the task. Some of the problems faced though were due to the physical separation of the members working on the project. Each member of the team acquired a Jetson, but there were slight differences in some, as one was of an older generation, and also had a faulty camera port that had to be returned. From there, all testing and programming had to be done remotely and sent to another member with a working Jetson in order to test programs written using a PC and separate camera. Another problem faced with the Jetson was learning the intricacies of the operating system from Nvidia. Though much like the Raspberry Pi, there were certain ideas

and concepts that had to be leveraged and also learned in order to adapt to the Jetson. Many libraries that were used in test environments were unable to be used on the Jetson, such as OpenCV 4.x.x. The Jetson was also unable to host the Audio Client and Server code. Basic audio and visual processing could be achieved (eye detection, distance measurements, and classical filtering), but due to limitations, not all modules developed separately could be joined to operate as one on the Jetson as planned.

3.2.2 Microphone

The microphone the team chose to go with was the Samson Go Mic. It was chosen due to its multiple directivity patterns (cardiod and omnidirectional), digital output, and low pre-processing coming directly from the microphone. Other microphones the team looked at had their own signal processing chip built in and did not give the raw signal that we desired to process digitally. The digital output is ideal for this project situation since the team desired to work with smaller amounts of data per audio frame as well. The team members wanted to do most of the audio improvements with code, rather than buying a nice sounding microphone. Multiple directivity patterns allowed the team to either pick up audio in all directions (omnidirectional pattern) if we designed 360 degrees of rotation camera tracking or select the cardioid pattern to only pick up audio in roughly 180 degrees, reducing the sound coming from the servos on the camera or noise sources off camera.

3.3 Pan/Tilt Camera

The Pan/Tilt Camera from Arducam was also made available to the team through the school and was suitable for the project prototype. Though better cameras and servos do exist, for the purposes of prototyping the project, the already available Arducam model was more than capable of performing. The development of the programs for the pan/tilt system could be leveraged into any other servo/camera system with a simple change of some parameters.

3.3.1 VAD

A classical DSP approach to detect voice activity is to use the short-time zero crossing rate (ZCR) and short-time energy of the audio input. Short-time energy sets a minimum energy threshold to identify sound activity, and ZCR measures how frequently signals cross zero to identify whether the sound is likely to be speech. It is found that voices have a lower ZCR and high energy compared to non-voices. A ZCR threshold and energy threshold are designed to give an estimation of voice activity. While this classical approach requires low computation resources and has good results against noises with high ZCR or low energy, its accuracy is still too low for the project's application. It fails to identify quilt voices when the energy threshold is low and will identify loud noise as voices when the noise threshold is high. It is time consuming to adjust

optimal ZCR and energy threshold. The team wanted to have a VAD module that can be directly used by the user without any additional adjustment using any input devices under any kind of noise environment. It needs to have a high accuracy, be robust against invariant noises, have short delay, and require low computation power. In the end, the team members decided to have a deep learning based VAD module that takes Mel-Frequency Cepstrum Coefficients (MFCC) as input feature.

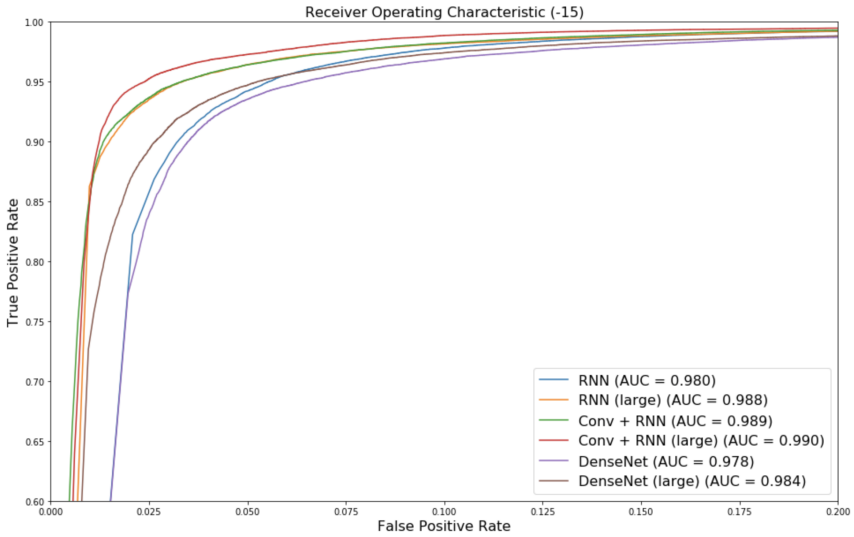

Figure 13: ROC curves of different network [16]

MFCCs are short term spectral based features that reflect human hearing perception of input audio [14]. MFCCs convert the array of audio input into an array of coefficients, and they are usually used for speech recognition tasks. Compared with using raw audio data without feature extraction, MFCCs reduce the size of the network. The team tested 12 and 24 parameters MFCCs and found that 12 parameter MFCCs have lower validation loss and faster speed.

In "Voice Activity Detection in Noisy Environments," the author compared several network candidates, including LSTM, Gated Conv + GRU, and DenseNet [16]. The Area under the ROC Curve score (AUC) is shown in Figure 13. AUC measures the area under Receiver Operating Characteristic (ROC), which shows True Positive Rate against False Positive Rate. The higher the AUC value, the better the accuracy. It is shown that the LSTM the team used (same structure as the one used by the blue curve in the plot) achieves an AUC score of 0.980 under -15 db noise. Though it is smaller than Gated Conv + GRU and DenseNet, the LSTM network has the least number of parameters (with 13200 total parameters), and the accuracy is good enough for the project's application.

3.3.2 Denoiser

To ensure the meeting quality, the team wanted to denoise the raw input as much as possible without introducing too much distortion and also wanted to keep processing speed fast. While deep learning denoisers have a better denoise effect, the U-Net structure and LSTM layer introduce a large number of parameters and have a high requirement on computation power. Traditional denoisers based on least mean square error (LMS) or log-spectral amplitude estimator (LSA) have a better processing speed and much simpler calculation. Because the denoiser level is not that high, traditional DSP denoisers also introduce less distortion to the speech. Deep learning denoisers can either work in the time domain or Time-Frequency (TF) domain. Time domain denoisers use an end-to-end approach that takes raw audio as input and outputs denoised audio. On the other hand, Time-Frequency domain takes a spectrogram as input and outputs spectrograms. In the application, the team used both time domain approach and frequency domain approach. Time domain approach can reduce the step of spectrogram-audio conversion, preventing inverse Short Time Frequency Transform (ISTFT) that has high CPU consumption. Also, compared to the TF domain method, time domain methods will not lose phase information. When the TF domain method is used, only the magnitude of the spectrogram is used after STFT, and the output is calculated using the original phase, which contains the noise. The loss of phase information causes a reduction in SNR. However, it is possible to train TF-domain models on both the real and imaginary part of the input spectrogram, as shown in "Deep Complex Convolution Recurrent Network for Phase-Aware Speech Enhancement (DCCRN)" [11]. The benefit of TF-domain methods is that current time domain models usually have more feature channels, hence a large number of parameters. For example, Demucs with a hidden size of 48 have 33M parameters (12M parameters for bidirectional LSTM), while DCCRN only has 3.7M parameters. However, their processing speed on i-5 CPU does not differ a lot, and the RTF of DCCRN only exceeds Demucs by 32 percent.

3.3.3 EQ

The EQ consisted of cascaded biquad IIR filters which were designed by performing the bilinear transform on various analog filters [6]. Initially, the EQ chain was designed to include successive Low Pass, Low Shelf, Peaking, High Shelf, and High Pass filters. However, the shelving filters in this model would not allow for the fine changes the chain was intended to correct by weighting small ranges of frequencies differently than the rest of the frequencies. Therefore, this model was traded for one which only included the Low Pass, Peaking, and High Pass modules. In implementation, the number of Peaking filters could be increased from 1 to as many as the application requires, though 1-5 should be enough for most applications.

In selecting Q and A values for these filters, both equations and experimentation were used. The Q-factor, or Quality factor, is related to the "width" of

the filter. It describes the transition width of the filter, with a lower Q corresponding to more sudden transitions between passband and stopband and a higher Q corresponding to a more gradual transition. A Q-factor of .707 is associated with a steeper transition, and is often used in audio applications. A value is associated with the gain of the filter, with a larger value associated with greater gain. Though an A-value of 1.19 would change gain by a maximum of 3 dB (twice or half the volume, depending on sign), an A-value of 2 was chosen in order to quite noticeably hear a difference in the processed signal.

3.3.4 Facial Processing

The Dlib toolkit was utilized for its accuracy, which was above other models. The initial test was done using OpenCV's Haar Cascade facial detection model, which while was usable, was not very accurate and consistent over time [3]. Also when the user was wearing glasses, the eye detection as part of OpenCV failed considerably. Therefore, the use of the Dlib toolkit was used to improve accuracy and actually marginally improve performance time. The improved accuracy was necessary for the team's desired outcome. Since the team needed to get accurate position data and not just recognize that there was a face in the display, the better accuracy was a trade off with speed that was acceptable. Other non-learning algorithms are available for facial feature detection, such as SIFT or SURF, but the accuracy is not close at all compared to the learning-based approach. Dlib uses a Histogram of Gradient's approach to learn from the model, but it also has a convolutional neural network approach to facial detection [8]. While the deep network approach would yield slightly more accurate results, the overall time to detect the faces was not optimal for this project situation. The HoG approach provides a balance between speed and accuracy that the team decided was necessary for project implementation and further metrics are provided as evidence in the Experimentation and Data Analysis section.

Another part of the video pipeline was the distance detection. Some of the early ideas that were floated out and researched include using a dual camera model to get disparity and obtain the distance that way [15]. Another way was using a deep learning algorithm to learn distance and return. The team ended up using a small heuristic in order to find the distance. Recognizing that the environment in which the project's camera would be positioned would be severely limited, and that in order to get distance, all the team would want to find is the face of the subject, then team members could use some generalities and basic algebra to find the distance. By knowing the focal point of the camera and some known distance in the frame in the real world, the team could find that distance in pixels and then use the focal length to calculate the distance from the camera. The known distance the team decided on was the eye distance, which had a pretty reasonable average across most people. By assuming we know the eye width of the subject, then the team can calculate the eye width in pixels and then find the distance to camera using the focal length.

3.4 Experimentation and Data Analysis

3.4.1 Voice Activity Detector

In the test, with 300ms input frame length, the VAD module reaches an average real time factor of 0.14 on i-5 CPU and 0.21 on Jetson GPU. An example of the VAD result is shown in Figure 14. The top part shows the audio with true label, and the bottom part shows the audio with predicted label.

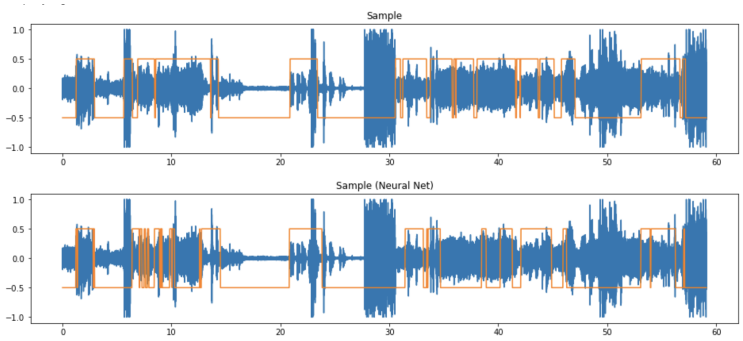

Figure 14: VAD example (top: true label, bottom: prediction [16])

3.4.2 Denoiser

Under an input length of 200ms, which is 3200 data points under 16Khz sampling rate, the Demucs denoiser works with a Real Time Factor (RTF, time taken for process/input length) of 0.65 on a 4-core i-5 MacBook Pro CPU or 2080-Ti GPU, but it cannot run in real time on a Jetson GPU with a RTF of 2.78. The DCCRN denoiser reaches a real time factor of 0.49 on MacBook Pro CPU. The STFT is calculated inside the DCCRN model, so the time and frequency domain transformation time is included in the 0.49 RTF. Both deep learning denoisers work on PC but fail to reach near real time on Jetson. On the other hand, the OMLSA denoiser achieves a RTF of 0.2 on Jetson CPU and 0.03 on computer CPU. The RTF of deep learning denoisers decreases when the frame length of the input increases, while the RTF of OMLSA denoiser does not change much.

On DNS challenge dataset, Demucs with a hidden layer size of 48 reaches a Perceptual Evaluation of Speech Quality (PESQ) score of 2.93, while the DCCRN model reaches a PESQ of 3.181 [7] [11]. As a result, DCCRN is preferred.

The following graph (Figure 15) shows the denoising effect of the OMLSA denoiser on the white noise of earphone input.

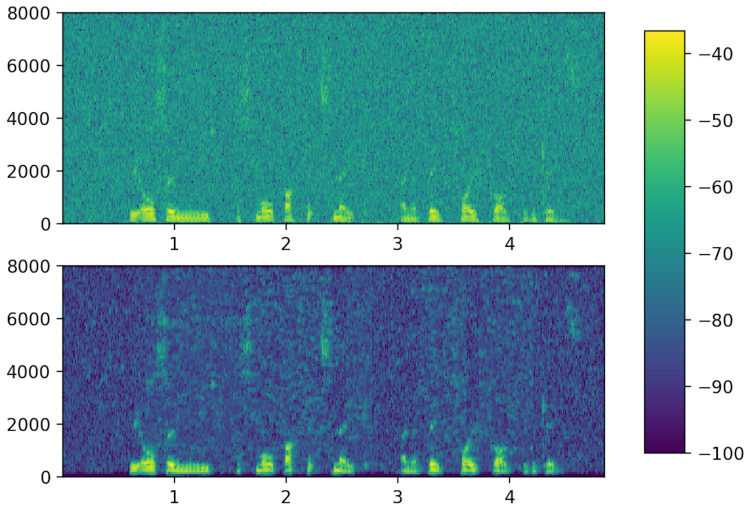

Figure 15: OMLSA denoiser sample (top: noisy signal, bottom: denoised signal)

3.5 Facial Detection

When comparing the execution times of the various algorithms, Dlib was actually faster than the Haar Cascade and more accurate for a single face subject. The average execution time per frame for the Haar Cascade using OpenCV was around .12 seconds, while the average execution time per frame for the Dlib HoG detector and predictor was .08 seconds. HoG and the Haar Cascade had about the same performance when the mouth of the subject is occluded, but HoG was more resilient to eye occlusion than Haar Cascade. If the Haar Cascade had almost any part of the eye occluded, then it failed to recognize a face. Another aspect in which the HoG outperformed the Haar Cascade was in the realm of resiliency to movement. Following the same head movement pattern for each video, the HoG maintained 100 percent accuracy of facial recognition, but with the Haar Cascade, any movement resulted in missed faces in frames much below 100 percent. The final aspect in which HoG outperformed was consistency of facial detection boundaries. The standard deviation of frame boundaries for the HoG using Dlib was much lower than that of the Haar Cascade. The deviation from the mean of the corner horizontal position of the facial bounding box for the Haar Cascade was 10.366 pixels while for HoG using Dlib it was 1.684 pixels. This was measured by taking the first 1000 frame values for that position while feeding the same facial image as a different frame into the processing pipeline. As for the eye detection, HoG was incomparable to the eye detection for the Haar Caascade. Most frames for the Haar Cascade only detected one eye while the Dlib values were again almost at 100 percent accuracy.

3.6 EQ

Evaluation of the EQ code was done through listening and visualizing audio files before and after processing. Since audio spectrum information is difficult to perceive in a quantitative way, visualization of the spectrum is essential to truly measuring the effectiveness of the filtering process. Though testing was done on a number of signals throughout the refinement of Q and A values, only one EQ chain will be evaluated here for sake of brevity. The EQ chain used in this testing involved a Low Pass filter with center frequency 7000 and Q-factor .707, a High Pass filter with center frequency 900 and Q-factor .707, and a Peaking filter with center frequency 3000, Q-factor .707, and A-value of 2. These parameters are by no means recommended for common use, but were selected to cut out high and low frequencies not associated with speech data (above 7kHz and below 900 Hz) and increase speech intelligibility by increasing gain of frequencies associated with consonants (2kHz - 4kHz).

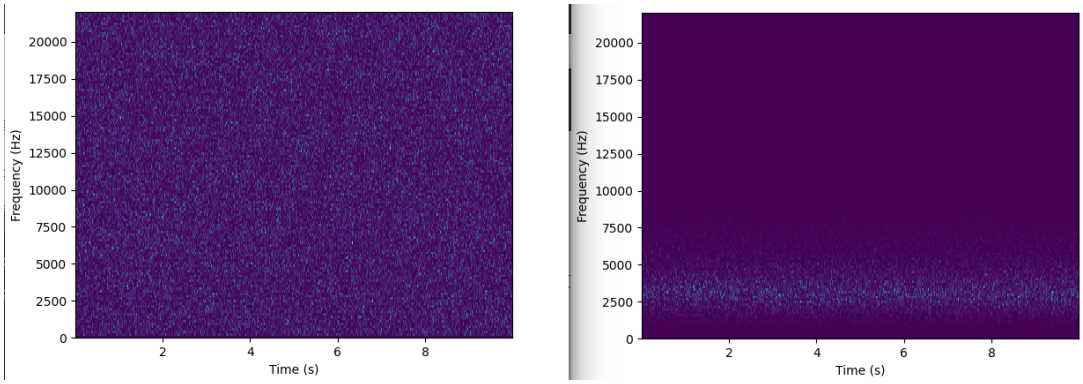

Figure 16: White Noise before (Right) and after (Left) EQ Processing

When a white noise signal was processed with this EQ chain, we can see that frequency content above 7 kHz and below 900 Hz was greatly reduced as shown in Figure 16. The highest weighted frequency content also is centered around 3 kHz as intended.

When testing the processing chain with a sine sweep from 20 Hz to 44 kHz, the effect of the peaking EQ is much more noticeable as shown in Figure 17. Not only do frequencies below 900 Hz and above 7 kHz appear to be much less prevalent, but the frequencies beginning around 3 kHz appear to be much stronger. This proves that the peaking filter is in fact weighting frequencies around its band much higher than frequencies outside the band.

By processing pink noise with the EQ chain, we can evaluate the effectiveness of this filter chain in redistributing frequency content to a desired range as shown in Figure 18. Initially, we can see that frequency content of the pink noise signal is much higher in density in low frequency ranges and less dense in higher frequency ranges. After being passed through the EQ, which includes a

normalization process, we can see what appears to be "more" frequency content information than we had before in the desired band. This is because the EQ chain not only removes unwanted frequencies and weights others, but also includes a normalization process to ensure consistent output volume.

The most important test of the EQ is on speech signal recorded with the designed hardware setup. A few seconds of vocal information was recorded using the Samson Go Mic, then processed using the specified EQ chain. We can see in Figure 19 that frequency content around 3 kHz was correctly increased in gain and frequencies below 900 Hz and above 7 kHz were correctly decreased in gain. When listening to the speech and processed speech, the signal after processing does have less low frequency content, higher volume when it comes to frequencies where consonants occur, and less high frequency content in the range of "breathy" sounds (above 6 kHz). From this evaluation, we can conclude that the EQ chain is working in a noticeable way to process the audio.

Another important criteria for filter performance is processing time. Processing on a speech signal was timed for each individual module of the EQ. Using a second-order-sections filtering approach with the SciPy library, the Low Pass filter (at 900 Hz) could process 1 second of audio in roughly 2.5 milliseconds, the Peaking filter (at 3 kHz) could process 1 second of audio in toughly 2.2 milliseconds, and the High Pass filter (at 7 kHz) could process 1 second of audio in roughly 16 milliseconds. The large difference between the Low Pass and High Pass filter execution times results from the frequency ranges they had to process. Since the speech signals processed only began having content around 100-200 Hz, this filter only had to reduce information between 100 and 900 Hz, an 800 Hz range. However the speech signals contained information up to 10 kHz, so the High Pass filter worked over a range of 7 kHz to 10 kHz, a total of 3 kHz. This could explain the large difference in processing time.

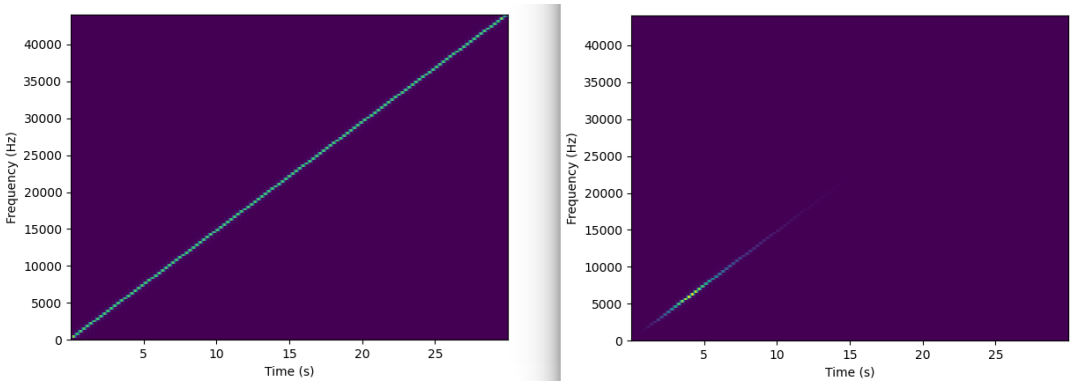

Figure 17: Sine Sweep 20 Hz - 44 kHz before (Right) and after (Left) EQ Processing

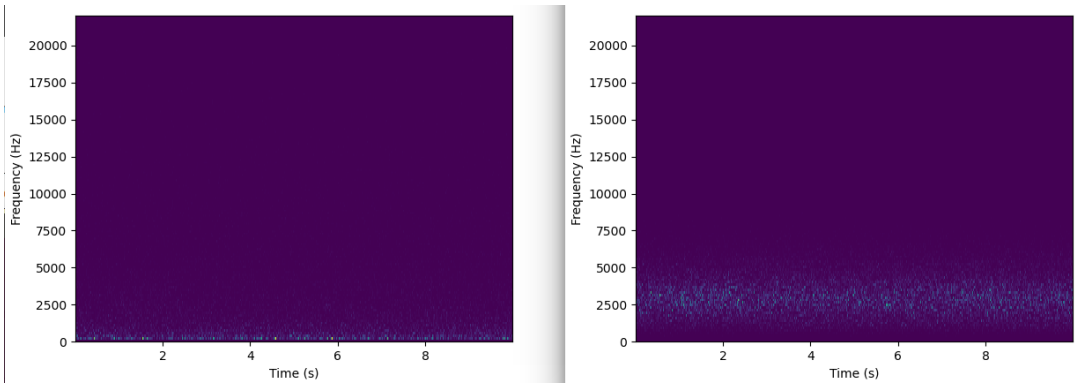

Figure 18: Pink Noise before (Right) and after (Left) EQ Processing

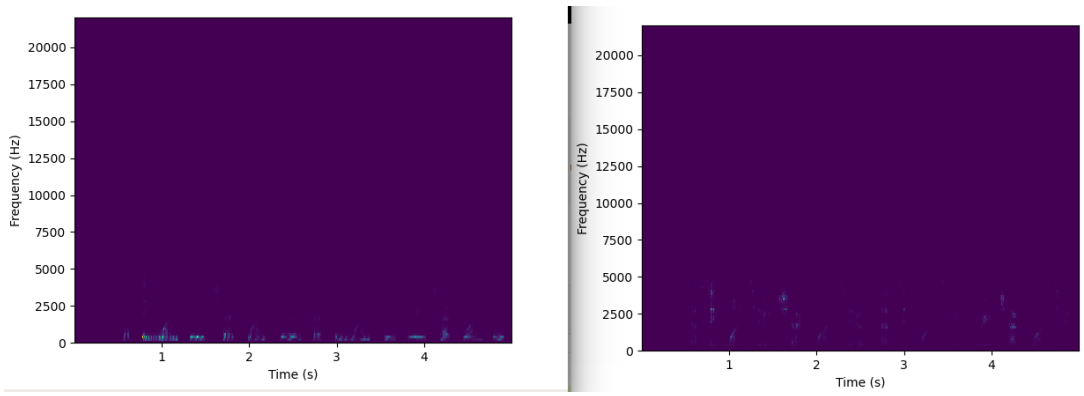

Figure 19: Speech Recorded on Samson Go Mic before (Right) and after (Left) EQ Processing

3.6.1 Moving Forward

Some of the faults in our design that would need to be ironed out first and foremost was one of the aspects that stalled the team for a while. We cannot establish socket communication between the Jetson and the PC. The Nvidia tool kits can only set up RTP server from video or audio files, and Jetson Python socket cannot reach a PC. If the goal of the project moving forward would be to implement this design so that it would be compatible with pre-existing video chat platforms such as Zoom or Microsoft Teams, then both the video and audio feeds would have to be transmitted to a device with such capabilities. This could either be through streaming or through a wired connection. Another aspect that would have to be improved upon would be the synchronization of both audio and video. Given that both were done on device, the synchronization was not as much a problem, but if the outputs were to be streamed or transmitted, then

synchronization would be a major factor in research and implementation.

The current control interface does not support meeting rooms, and users have to use feedback loops and other meeting software to hold a meeting. In the future, we can hold the application on a cloud GPU server or a cluster of GPU servers, and allow the user to meet with other people online through our application.

There are also improvements possible on the denoiser models. Many tricks and novel model architectures can be used to improve accuracy and speed. For example, we can add more skip connections between U-Net layers of different resolution or combine time domain features with frequency domain features. Models based on attention or dilated convolution can also be used to extract global features.

Since not all modules were able to be combined on the Jetson due to issues with library and platform compatibility, finding a work around to this in order to connect them would be a great improvement. Visual information about speaker distance or head tilt was unable to be communicated to the audio modules, so the audio modules were unable to react to this.

3.7 Societal Impacts

3.7.1 Public Health with COVID-19

Though this project is not directly involved in the progression of public health, it is tangentially related, especially with the developments of society in the past year. The world as a whole saw an increase in the number of people communicating digitally via personal computers. Everyone was forced to perform many business functions electronically and thus a necessity was having some sort of audio-visual system. This system, if implemented, would allow for an increase in the quality of communication. Lack of quality can distract from the presenter's goal in communication, and thus, having a better quality system allows for better communication in the current world.

3.7.2 Business

Another aspect where both the idea and the impact of a device like this project is in the world of business. Many professional organizations typically spend significant financial resources on the transportation, housing, feeding, etc. of various employees in order to hold meetings all around the globe. This can include national and international travel as well. Due to the pandemic and the de facto forcing of people to stay home and communicate electronically, many organizations have realized that many of their meetings can be held electronically, not only to stay safe during these strange times, but also in the future as a measure of cutting costs. The benefit of having a system that provides higher quality electronic communication is one aspect that allowed these companies to make such a decision. Employees can come to the office less and travel less due to the advancement in communication technology quality.

3.8 Team Details

3.8.1 Matt's Responsibilities

Matt's responsibilities were to first select the microphone and also select an optimal classical DSP audio filter type. He was also responsible for interfacing the various components (such as the microphone and camera) to the Jetson and troubleshooting issues with the Jetson, including installing required libraries and debugging programs on the Jetson. Matt created the customizable EQ chain code and experimented with EQ parameter values to determine reasonable ranges for Q, A, and important frequencies. He fine tuned the EQ code to reduce processing time to as little as 2.2 milliseconds of processing for 1 second of audio. As Matt had some audio processing experience coming in to the group, he also was responsible for researching speech signal characteristics.

3.8.2 Michael's Responsibilities

Michael's responsibilities were to research camera and pan/tilt systems. Prior to learning of the faulty camera port, his responsibilities also included troubleshooting and testing Jetson functionality. Other responsibilities were centered around the image processing pipeline. He was responsible for the research and development of facial detection, tracking, and processing as well as adjusting for distance and digital zoom. This includes evaluating various facial detection and tracking algorithms and deciding on which one properly suits the team's needs. He also was responsible for researching distance detection algorithms and thinking about a novel heuristic to use in the team's limited scope.

3.8.3 Yuzhou's Responsibilities

Yuzhou's responsibilites included the design and programming of the main structure of the application (the server, client, and Flask control panel). He also integrated audio modules into an audio DSP chain as well as design, train, and implement an audio-based voice activity detection module. Yuzhou also focused heavily on the deep learning models including the research and testing of various audio processing network models. Yuzhou also trained the pre-trained network model on extra datasets, and integrated the denoiser into the project. He also implemented the OMLSA+IMCRA denoise algorithm on Python in real time.

3.9 Independent Learning

3.9.1 Hard Skills

Matt: The hard skills that will be taken away from this project for Matt mostly revolve around how to read and write audio using Python and how to filter that audio data using Python libraries as well. A first for this project was real time audio processing using Python and thus how to process audio by designing efficient digital signal processing chains. Another key feature of

Python learned with respect to audio is how to use SciPy to filter and analyize audio signals as well as how to multithread within Python. Some general skills learned from this project are the various characteristics of speech signals and speech intelligibility. Much of audio processing is a combination of human preferences along with the math and science of the signal, and thus learning this characteristic is very important and will be applied moving forward within audio engineering projects.

Michael: The concrete ideas that will be applied in the future for Michael include the leveraging of existing GitHub repositories into the current code. This means that recognizing that many people have done projects before and though the entirety of the project is not already done, many subproblems have been faced before. So, dividing a problem into subproblems and looking for how to solve those subproblems can lead to a better understanding of how to complete the project. One thing that must be kept in mind though is that just because there is an existing project does not mean the implementation of that project was optimal; therefore, other solutions must be researched and tested against to find the optimal answer to the problem one faces. The use of the Jetson was brand new and learning how to read documentation was vitally important. This idea was also applied to the servos as well as the camera. The better understanding of how to use Python's powerful NumPy library was also vital to the project and will be further used as Python will most likely be used again at some time in the future.

Yuzhou: Much of the knowledge and hard skills taken away from this project for Yuzhou are centered on deep learning models and denoising algorithms. Specifically, the focus was on the design, modification, training, and testing of deep learning models on PyTorch using both CPU's and GPU's through CUDA. He also learned general knowledge about common layers and structures of networks like CNN's (1D, 2D, and dilated), LSTM's, Transformers, U-Net, DenseNet, skip connections, and fully connected layers. He also learned how to apply statistics knowledge to calculate prior and posterior noise probability and implement traditional DSP denoiser using recursion (LMS and OMLSA). Much like Matt and Michael, Yuzhou also furthered his skills in Python, specifically in how to perform faster implementations of NumPy array calculations as well as learning how to apply websocket programming in Flask, and combine Flask, html, and javascript for a dynamic web page.

3.9.2 Soft Skills

Compared to the more technical aspects learned through the process of developing the device, the team also learned more general aspects of development as a whole. One of the major issues the team members ran into with this project was learning how to adapt to failures. Many times we would try a certain protocol on the Jetson and something catastrophic would happen. We had to learn how to look back at the problem, research more, and adapt to the issue. One such problem we had was severe trouble with the UDP of communication from the Jetson to the PC. Something we took away from this issue was to re-evaluate

what our goal of this project was. The main focus and goal was to focus on the signal processing aspect of the project. We were spending a good amount of time trying to figure out the communication, which should have been spent focusing more on the signal processing aspects of the project. We decided to forgo the UDP and do all processing and prototyping on the Jetson. This re-evaluation is a concept that will happen more in future projects, and staying focused and not getting lost in weeds is incredibly valuable to remember. Another part of the project that we learned from is more about the teamwork of the project itself. There have been projects throughout our undergraduate program that involved working as a team, but those projects were usually done in person. The disconnected nature of the assignment due to conditions out of our control was valuable to learn from. We had to leverage multiple communication systems such as Slack, Zoom, and GitHub in order to work together to develop the project. Hopefully we will not find ourselves in a global pandemic again anytime soon, but the take-away is that there will be certain aspects of life that arise and projects have to adapt around them in a team setting.

References

[1] *Adafruit PCA9685 16-Channel Servo Driver*. URL: `https://learn.adafruit.com/16-channel-pwm-servo-driver/using-the-adafruit-library`.

[2] *Arducam pan Tilt CAMERA bundle for Nvidia Jetson Xavier NX/NANO, 2 TWO-DOF platforms with 8mp IMX219 cameras, with PTZ Control Board*. URL: `https://www.arducam.com/product/arducam-pan-tilt-platform-for-raspberry-pi-camera-ptz-control-board-8mp-b0191pt/`.

[3] Girija Shankar Behera. *Face Detection with Haar Cascade*. Dec. 2020. URL: `https://towardsdatascience.com/face-detection-with-haar-cascade-727f68dafd08`.

[4] Israel Cohen. *Noise Spectrum Estimation in Adverse Environments: Improved Minima Controlled Recursive Averaging*. 2003.

[5] Israel Cohen and Baruch Berdugo. *Speech enhancement for non-stationary noise environments*. 2001.

[6] *Cookbook formulae for audio equalizer biquad filter coefficients*. URL: `https://webaudio.github.io/Audio-EQ-Cookbook/audio-eq-cookbook.html`.

[7] Alexandre Defossez, Gabriel Synnaeve, and Yossi Adi. *Real Time Speech Enhancement in the Waveform Domain*. 2020. arXiv: `2006.12847 [eess.AS]`.

[8] *Face Detection – OpenCV, Dlib and Deep Learning (C++ / Python)*$_2$018. Oct. 2018. URL: `https://learnopencv.com/face-detection-opencv-dlib-and-deep-learning-c-python/`.

[9] *Facial landmarks with dlib, OpenCV, and Python*$_2$017. Apr. 2017. URL: https://www.pyimagesearch.com/2017/04/03/facial-landmarks-dlib-opencv-python/.

[10] *Go Mic*. URL: http://www.samsontech.com/samson/products/microphones/usb-microphones/gomic/.

[11] Yanxin Hu, Yun Liu, and Shubo Lv. *DCCRN: Deep Complex Convolution Recurrent Network for Phase-Aware Speech Enhancement*. 2020. arXiv: 2008.00264v4.

[12] *IMX219 - 8MP Camera*. Aug. 2020. URL: https://www.arducam.com/docs/camera-for-jetson-nano/native-jetson-cameras-imx219-imx477/imx219-8mp-camera/.

[13] Davis E. King. "Dlib-ml: A Machine Learning Toolkit". In: *Journal of Machine Learning Research* 10 (2009), pp. 1755–1758.

[14] Beth Logan. *Mel-Frequency Cepstral Coefficients for Music Modeling*.

[15] Nabil Madali. *Object Distance Measurement By Stereo Vision*. July 2020. URL: https://towardsdatascience.com/object-distance-measurement-by-stereo-vision-37897a7ecb62.

[16] Simon Holst Albrechtsen Nicklas Hansen. *Voice Activity Detection in Noisy Environment*. 2018.

[17] *py-webrtcvad*. URL: https://github.com/wiseman/py-webrtcvad.