THE HONG KONG POLYTECHNIC UNIVERSITY
香港理工大學

Department of Electronic and Information Engineering

Final Year Project Final Report

(2022/23)

Smart Security Robot

Student Name: WANG Zilai

Student ID: 19080007D

Program: EIE (42470)

Supervisors: Prof. Mak Man-Wai and Dr. Lun Pak-Kong

Submission Date: 2023.3.16

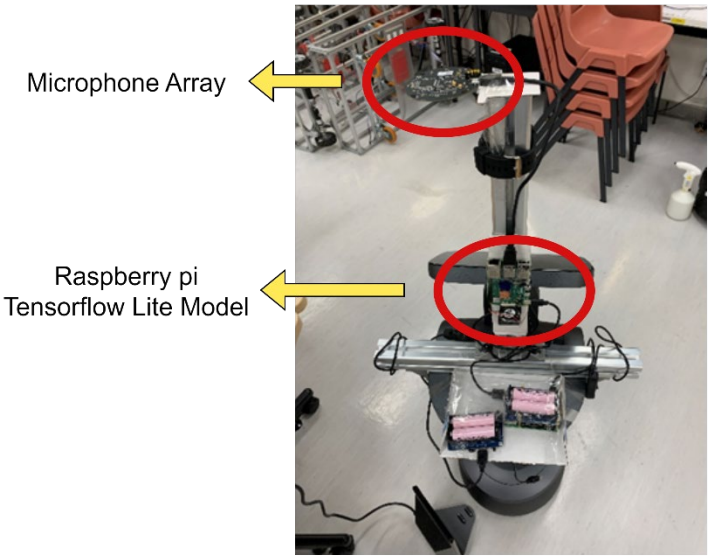


Table of Contents

Abstract

In the initial semester of the IFYP project, I developed a sound event detection system. However, due to deployment issues on the Raspberry Pi board and significant delays, I replaced the Pytorch-based sound event detection system with a locally deployable speech recognition system in the second semester. This new system was created using TensorFlow. Additionally, I built a sound source localization system to detect the direction of arrival. After analyzing two types of microphone arrays, I selected the Respeaker microphone array for its superior performance and flexibility. The link of the individual demo video for speech recognition system and sound source localization system is attached below: FYP_Pre_Individual.mov

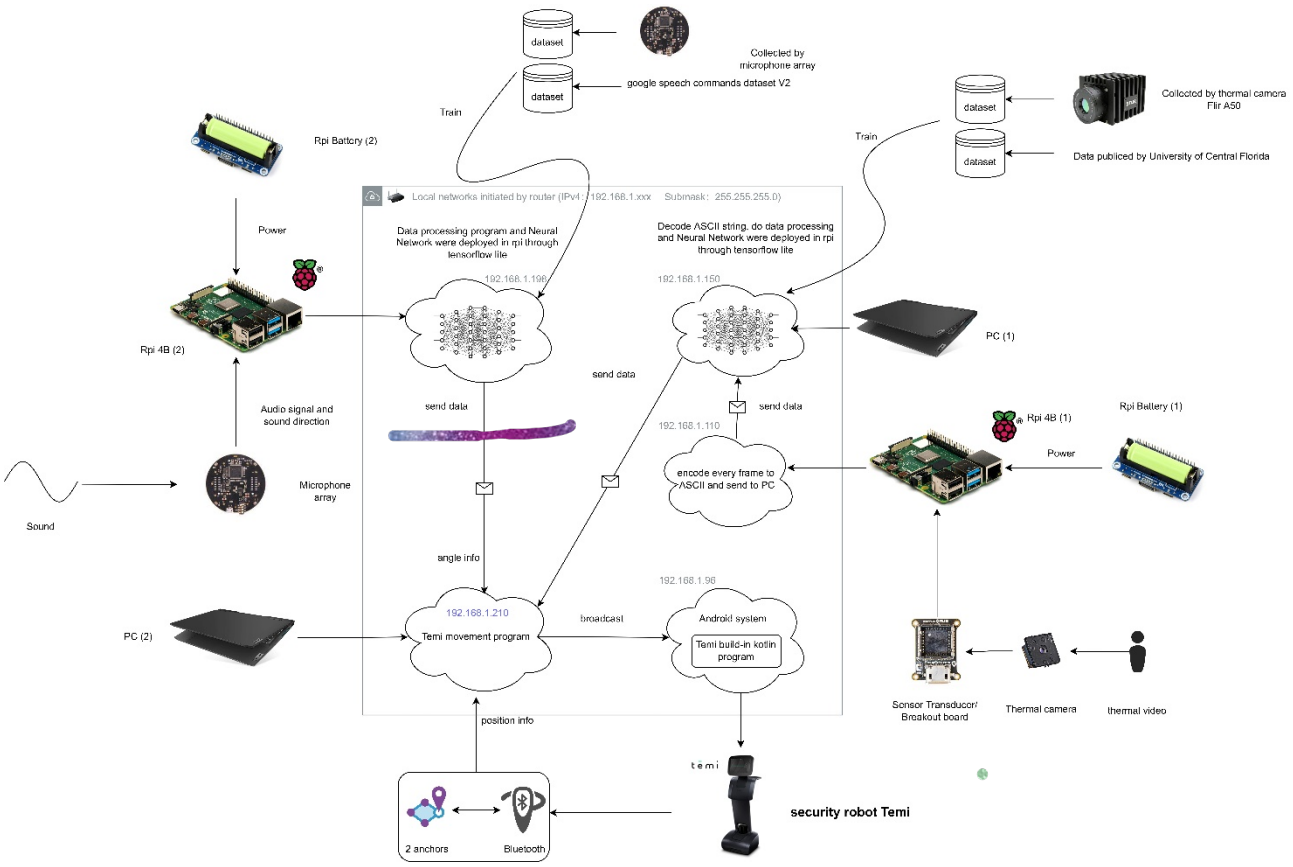


Figure 1: The process of the smart security robot project

1. Interdepartmental Project Introduction

There is a growing trend of utilizing artificial intelligence in surveillance systems [1]. With guidance from the EIE department, the intelligent security robot named Temi [2] aims to enhance laboratory security by detecting anomalies with sound and thermal video, specifically detecting people seeking help and people screaming.

My work focuses on three main areas: sound event detection, speech recognition, and sound source localization. In the first semester, I developed a sound event detection system using Pytorch to classify

sounds like air conditioners and the dog barking. However, the large size of the Pytorch model (i.e., more than 20 MB) made it unsuitable for deploying on the Raspberry Pi board, and there was a non-negligible delay in streaming audio data from the Raspberry Pi to a local computer where the Pytorch model was deployed. Therefore, in the second semester, I developed a speech recognition system from scratch using TensorFlow Lite so that the model could be deployed on the Raspberry Pi board for inference. After developing the deep learning algorithms, the speech recognition system was able to detect anomalies with sound, specifically detecting and classifying screaming and seeking help events. In addition to sound event detection and speech recognition, I also designed a sound source localization system that sends the direction of arrival of the sound to Temi once an abnormal event is detected. Communication between the Raspberry Pi board (client) and Temi (server) is based on the TCP-based socket method, which allows messages to be transferred between devices on the same local network.

2. Sound Event Detection

2.1 Background and Related Work

The intelligent robot Temi requires an automated system for detecting and classifying potentially dangerous sound events, such as screaming. To achieve this, I have designed two sound event detection systems: 1D convolution and 2D convolution (MFCC). The first system is based on the Audio Classifier Tutorial by Winston Herring [3] and involves processing audio data by converting it to 1 channel, padding with zeros if necessary, and down sampling to 8kHz before inputting it to the M5 network [4]. The second system is inspired by the convolutional recurrent neural network proposed by Cakir et al. [5] and utilizes a combination of CNN and RNN on MFCC and Mel Spectrogram data using 2D convolution. Both systems aim to detect and identify anomalous sound events.

2.2 Problem Definitions and Objectives

The goal of the sound event detection system is to identify potentially dangerous sound events, such as screaming and sirens, and classify them for use on the Temi Robot. Additionally, the system is designed to calculate a confidence score when classifying new audio data.

2.3 Design and Methodology

The UrbanSound 8k dataset [6] was chosen for this project, as it contains ten relevant classes of sound events. The dataset details are listed in the figures below, and it is well-suited to our goal of detecting and classifying anomalous sound events.

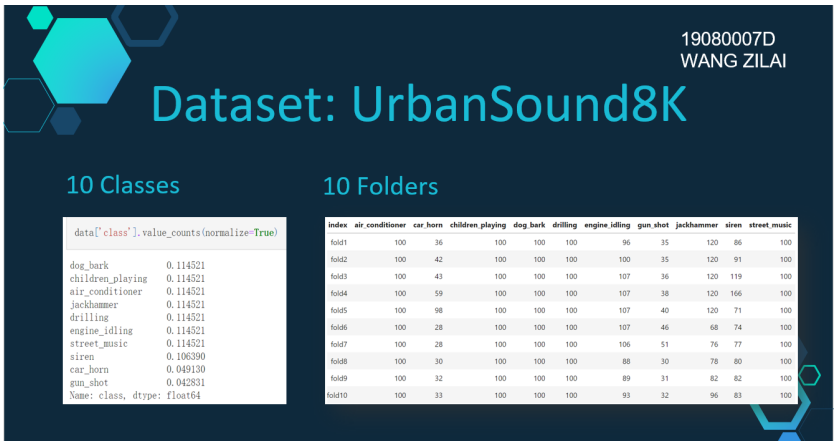


Figure 1: Content of the UrbanSound 8k dataset

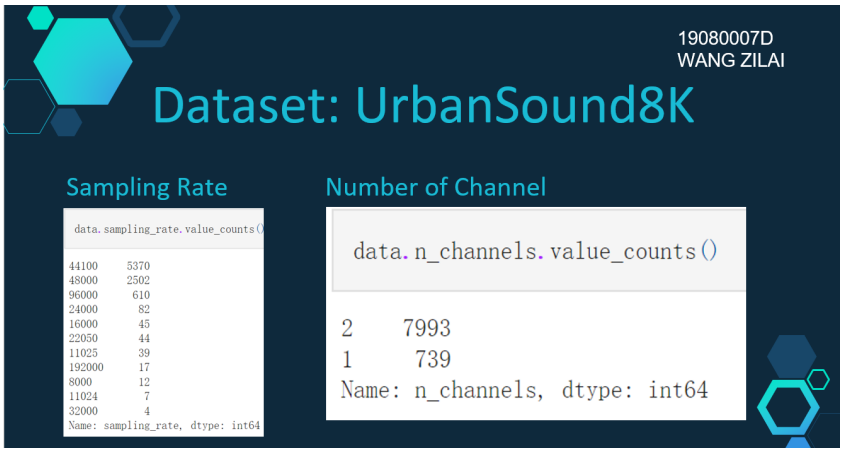


Figure 2: Properties of the UrbanSound 8k dataset

2.3.1 Convolution of time series

Prior to convolution, the data is preprocessed by converting it to a single channel, padding with zeros if needed, and down sampling to 8kHz.

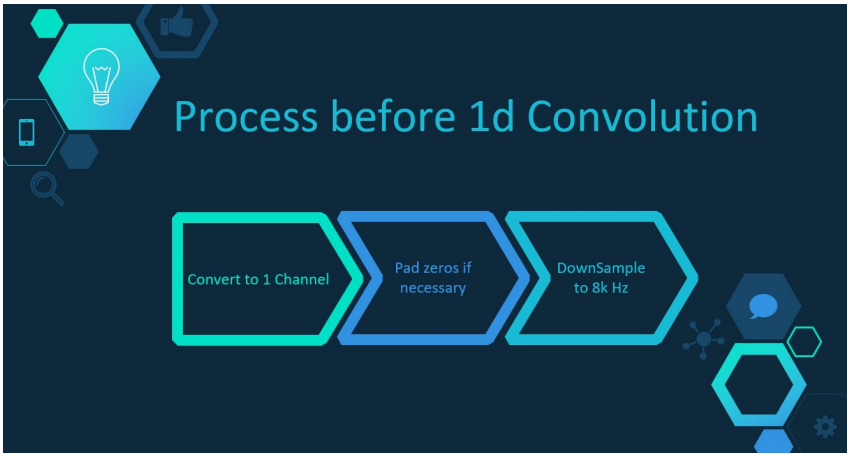


Figure 3: Processes of audio data by 1d convolution method

The torchaudio.transforms. The DownmixMono library used in the tutorial is now deprecated. Therefore, I utilized the torch.mean function as an alternative to achieve the same effect.

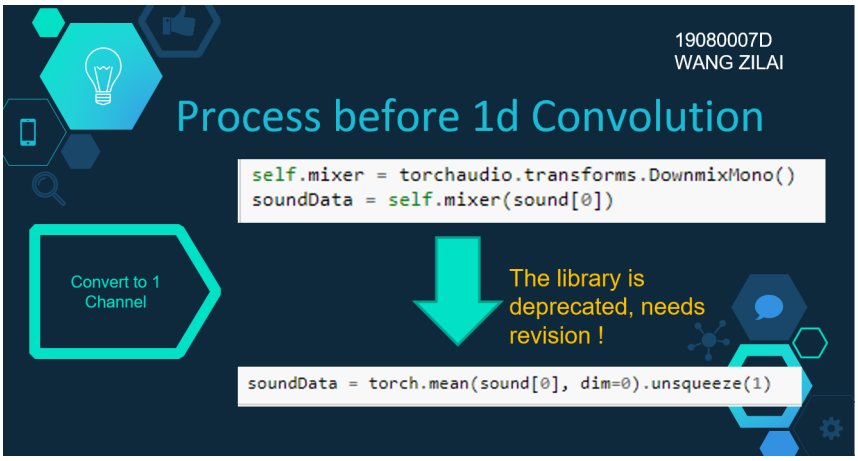


Figure 4: Revision of codes of 1d convolution method

Following the preprocessing steps, the data is then fed into the M5 network [4], which is composed of multiple 1D convolutional layers, batch normalization layers, pooling layers, and a single fully connected layer.

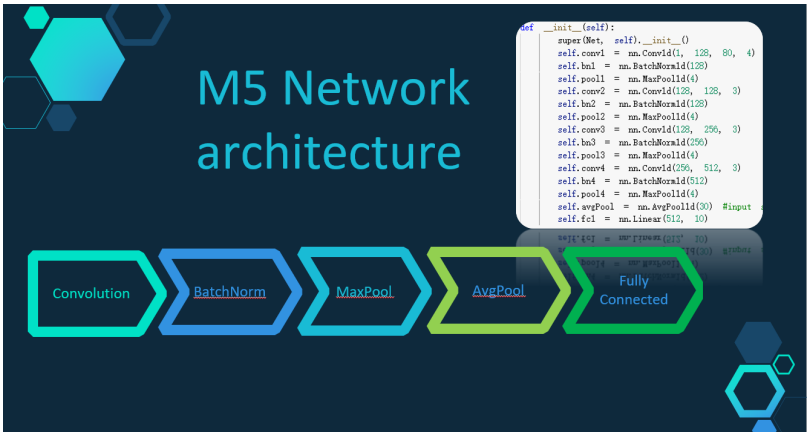


Figure 5: structure of the M5 network

2.3.2 Convolution of MFCC and Mel Spectrogram

Prior to performing convolution, I preprocess the audio data by applying short-time Fourier transform (STFT) and Mel scaling to obtain the Mel Spectrogram. The Mel Spectrogram is then subjected to a log transformation and discrete cosine transform (DCT) to produce the Mel-Frequency Cepstral Coefficients (MFCC).

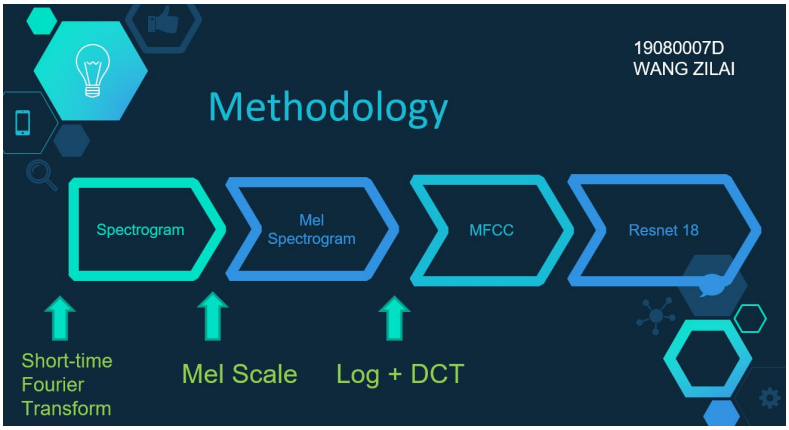


Figure 6: procedures of 2d convolution of MFCC and Mel Spectrogram

Following the preprocessing steps, I feed both the MFCC and Mel Spectrogram data into the Resnet 18 network. The Resnet 18 network is composed of four types of convolutional layers and plays a crucial role in mitigating the gradient vanishing problem.

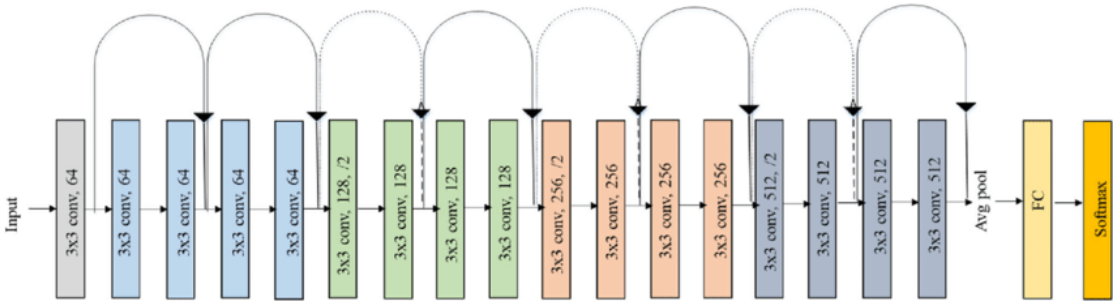


Figure 7: Structure of Resnet 18 network [7]

2.4 Experimental Results

2.4.1 Results of Convolution of time series

To train the model, I utilized the cloud server "智星云," with a GeForce RTX 2080 Ti GPU. The UrbanSound 8k dataset was divided into ten folders, each containing mixed classes of audio data. The first nine folders were used for training, while the tenth folder was reserved for testing. After downsampling, all audio data was approximately three seconds in length. The results of our experiments are summarized in the figure below.

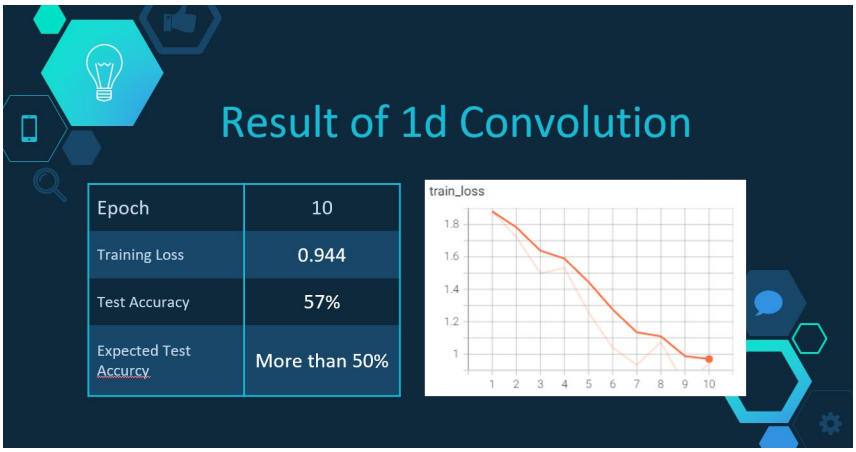


Figure 8: Result of 1d convolution of time series

2.4.2 Result of Convolution of MFCC and Mel Spectrogram

My model was trained on the cloud server "智星云" using a GeForce RTX 3090 GPU. To facilitate the training process, we created two text files that list the names and labels of the training and testing audio data, respectively. Additionally, I created a third text file that contains all possible classes of audio data. My code automatically reads these three files and processes the corresponding audio data,

providing greater flexibility to add or modify the number of classes and audio data for training and testing. The results of the convolution of MFCC and Mel Spectrogram are presented in the figures below.

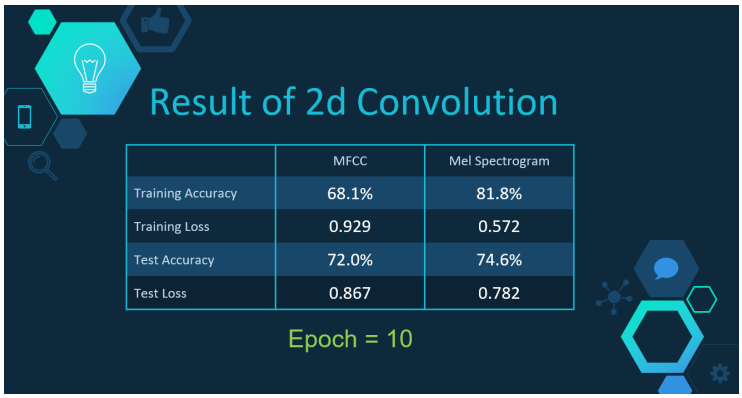


Figure 9: Result of convolution of MFCC and Mel Spectrogram

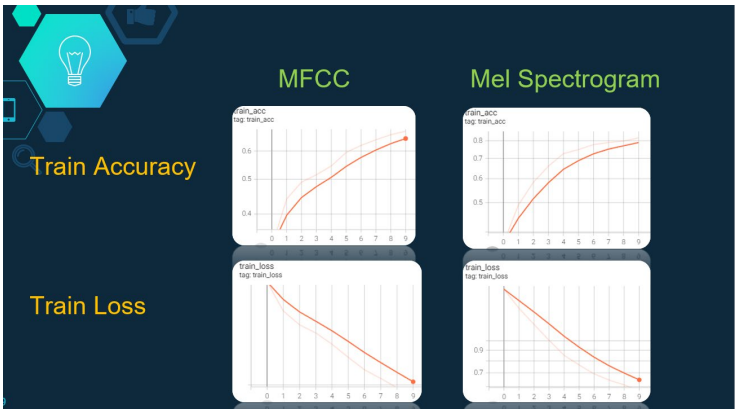


Figure 10: Diagrams of training accuracy and loss

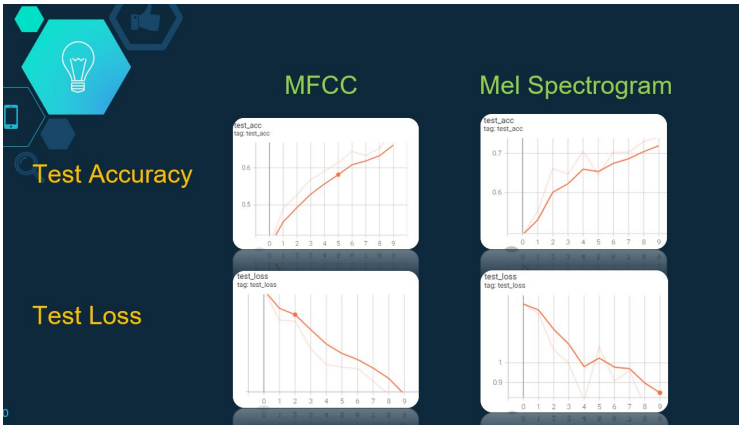


Figure 11: Diagrams of test accuracy and loss

2.4.3 Discussion of the Results

The accuracy of the 1D convolution of time series was unsatisfactory, with only 57% accuracy after ten epochs. There were several reasons for this. Firstly, averaging or mixing two channels to generate a single audio channel is not appropriate since there are usually phase differences between the two channels. This compromises the effectiveness of the generated features. Secondly, downsampling all audio data to 8kHz is not appropriate since the UrbanSound dataset contains audio data with frequency components higher than 4kHz, and our hearing threshold is up to 20kHz. Therefore,

downsampling to 8kHz based on Nyquist Sampling Rate is not suitable.

On the other hand, the results of the convolution of MFCC and Mel Spectrogram were acceptable. The accuracy of MFCC was around 70%, while the accuracy of the Mel Spectrogram was more than 80% after ten epochs. However, the models had not yet converged. Given a larger number of epochs, the performance tended to continue improving.

Interestingly, there was a significant difference between the accuracy of MFCC (68.1%) and Mel Spectrogram (81.8%). This is because the DCT transformation used in MFCC produces one large DC coefficient, while most AC coefficients are close to zero. After quantization, most AC coefficients become zero, which results in a loss of information. As a result, the Mel Spectrogram feature contains more useful information than MFCC's.

3. Speech Recognition

3.1 Background and Related Work

Speech recognition is the process of automatically recognizing what a person says based on the information in the speech signal, which has a wide range of applications in security systems [8]. Since speech is a continuously varying signal, features can be extracted from it to obtain information. Mel Frequency Cepstral Coefficients (MFCC) is a widely used feature extraction technique in speech recognition. MFCCs are derived from the power spectrum of the audio signal that represents the distribution of energy across different frequency bands. The MFCC algorithm mimics the human auditory system by using a filter bank to extract spectral information. Followed by a logarithmic compression and a Discrete Cosine Transform (DCT) to obtain a set of cepstral coefficients that capture the shape of the spectral envelope [9]. MFCCs have several advantages for speech recognition, including their ability to capture the important spectral information in speech signals, their robustness to variations in speaker characteristics and environmental noise, and their computational efficiency [10]. The process of MFCC is shown in the below figure.

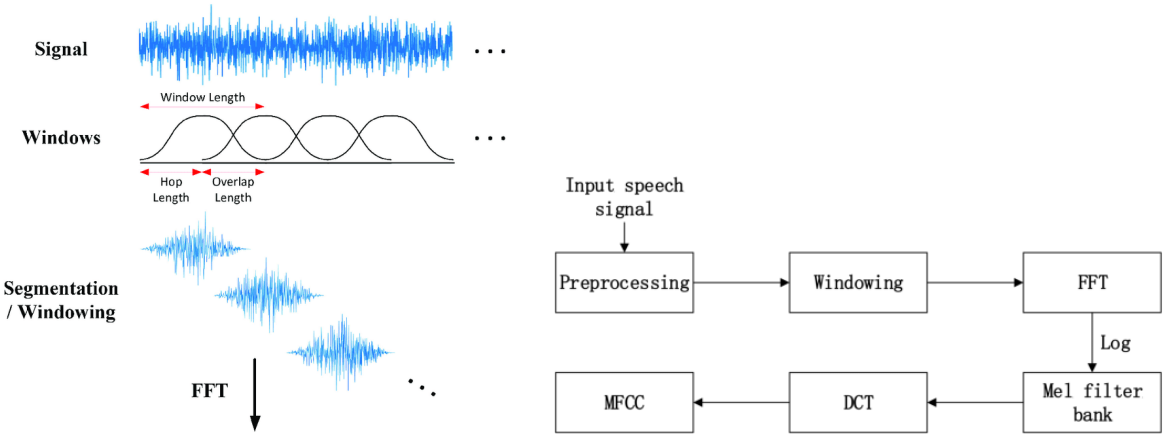


Figure 12: Process of MFCC [11]

There are two main types of speech recognition: speaker-dependent and speaker-independent. Speaker-dependent systems require the users to train the system to recognize their voice, while

speaker-independent systems can recognize any user's voice without prior training. Speaker-independent speech recognition systems are typically more complex than speaker-dependent systems as they must be able to deal with variations in accent, pronunciation, and speaking style, as well as background noise and other environmental factors [12]. There are several approaches to building speaker-independent speech recognition systems. One approach is to use a large corpus of speech data to train the system on a wide range of speech patterns and variations. Another approach is to use machine learning algorithms to continually adapt and refine the systems' recognition capabilities based on user feedback. One of the main challenges with speaker-independent systems is reducing the error rate, as they must be able to accurately recognize a wide range of voices and speech patterns. To improve accuracy, these systems often use a combination of acoustic and language models, as well as advanced signal processing techniques to filter out the noise and other unwanted signals [13].

3.2 Problem Definitions and Objectives

The speech recognition system designed for detecting anomaly events will be deployed on a Raspberry Pi board for inference. The system aims to accurately classify dangerous sound events, such as screaming and sounds of people seeking help.

3.3 Design and Methodology

The speech recognition system is built in four steps, involving data collection, feature extraction using MFCC, model training and testing, and deployment of the model on the Raspberry Pi. The speech data is collected from various sources, including the Google speech command dataset, which contains background noise, and recordings made by my friends and me using a computer microphone. All speech data is preprocessed to have a sampling rate of 16k, an audio length of 1 second, and a single channel after volume normalization. The parameters for calculating MFCC are listed in the above Table.

Sampling Rate	**Window Length**	Window Step	Number of Coefficients	**Number fft**
16k	**0.032s**	0.01s	16	**512**

Table 1:Parameters of MFCC

When considering the parameters in the Table above for audio analysis, it is important to carefully analyze the sampling rate (sr), window length, and number of FFTs (nfft). The resolution of the audio is determined by the sampling rate divided by the nfft. For example, with a sampling rate of 16kHz and an nfft of 512, the resolution of each spectral line would be 31.25Hz. Larger FFT sizes provide higher spectral resolution but also take longer to compute. The number of frames can be calculated as (audio length - window length) / window step + 1= (1 - 0.032) / 0.01 + 1 = 97.8 (my program shows it is 98).

The window length is determined by the inverse of the resolution, which in this case would be 1/31.25 = 0.032s. There is a trade-off between the length of the window size. If the window is too short, proper frequency information will not be captured in the frame. Conversely, if the window is too long, events will not be well resolved in the time since a frame will frequently "straddle" two or more

phonemes. This trade-off is illustrated in diagrams (a) and (b) below.

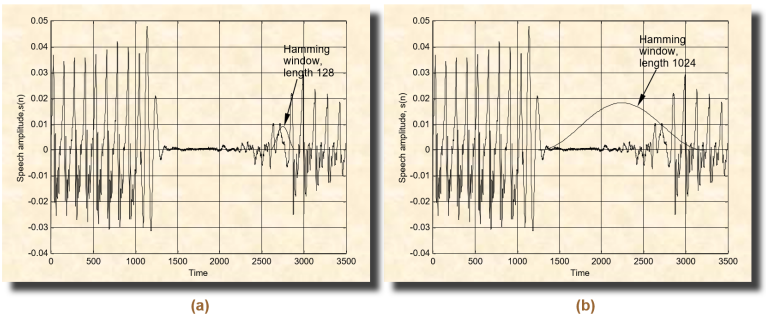


Figure 13: Trade-off of window length [14]

The images generated from MFCC have a size of 16x98 and are fed into a convolutional neural network. Batch normalization layers are added to improve stability, while SoftMax layers are included to enable the classification of more than two classes. The model structure is presented below:

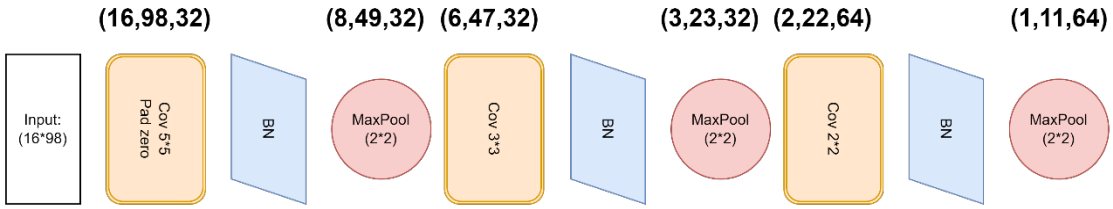


Figure 14 : Cov layers, BN layers, and Pooling layers

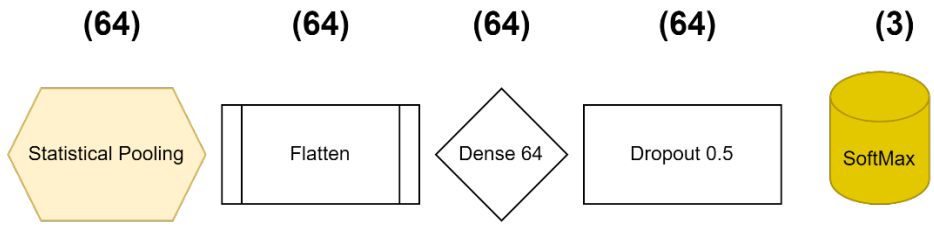


Figure 15 : Classification layers

```
# size of input is 16*98
model = models.Sequential()
model.add(ZeroPadding2D(padding=(2, 2), input_shape=sample_shape))
model.add(layers.Conv2D(32,
                        (5, 5),
                        activation='relu',
                        input_shape=sample_shape))
model.add(BatchNormalization())
model.add(layers.MaxPooling2D(pool_size=(2, 2)))

model.add(layers.Conv2D(32, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(layers.MaxPooling2D(pool_size=(2, 2)))

model.add(layers.Conv2D(64, (2, 2), activation='relu'))
model.add(BatchNormalization())
model.add(layers.MaxPooling2D(pool_size=(2, 2)))

# Classifier
model.add(GlobalAveragePooling2D()) # statistical pooling
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(num_classes, activation='softmax'))
```

Layer (type)	Output Shape	Param #
zero_padding2d_16 (ZeroPadding2D)	(None, 20, 102, 1)	0
conv2d_57 (Conv2D)	(None, 16, 98, 32)	832
batch_normalization_55 (BatchNormalization)	(None, 16, 98, 32)	128
max_pooling2d_55 (MaxPooling2D)	(None, 8, 49, 32)	0
conv2d_58 (Conv2D)	(None, 6, 47, 32)	9248
batch_normalization_56 (BatchNormalization)	(None, 6, 47, 32)	128
max_pooling2d_56 (MaxPooling2D)	(None, 3, 23, 32)	0
conv2d_59 (Conv2D)	(None, 2, 22, 64)	8256
batch_normalization_57 (BatchNormalization)	(None, 2, 22, 64)	256
max_pooling2d_57 (MaxPooling2D)	(None, 1, 11, 64)	0
global_average_pooling2d_3 (GlobalAveragePooling2D)	(None, 64)	0
flatten_12 (Flatten)	(None, 64)	0
dense_24 (Dense)	(None, 64)	4160
dropout_12 (Dropout)	(None, 64)	0
dense_25 (Dense)	(None, 3)	195

Figure 16: Code of the Keras model

The choice of kernel size in speech recognition using convolutional neural networks should depend on the size of the spectral features in the input data that are relevant to the task. For input data containing small spectral features, a smaller kernel size (e.g., 3x3 or 5x5) is more appropriate. Batch normalization layers can improve the stability and performance of deep neural networks by normalizing the inputs of each layer. MaxPool layers are used to down sample the spatial dimensions of the feature maps while retaining the most important information. In addition, statistical pooling layers can be used to summarize the acoustic feature vectors (such as MFCC) over time frames by computing the mean, standard deviation, or maximum value of the feature vectors over a fixed window or a variable-length sequence of frames.

The chosen model structure has several advantages. Firstly, the model size of 73KB is suitable for performing inference on the Raspberry Pi board, which has limited computational capabilities. Additionally, the model architecture is suitable for the small input size of 16x98 generated from MFCC. Statistical pooling is used to combine the outputs of multiple acoustic models or feature vectors (i.e., MFCCs) into a single representation that is more robust to noise and speaker variability. Despite its simple structure, the model performs reasonably well at classification, as detailed in the next section.

The model is designed to classify three classes of sound events. The first class, labeled as 0, consists of screaming sounds ("Ah!" and "Wow!") obtained from both the Google speech dataset and my own recordings on my local computer at W311. The second class, labeled as 1, consists of sounds of people asking for help ("Help!") obtained from my own recordings on my local computer at W311. The third class, labeled as 2, includes background noises and other non-key words. The background noise includes white noise, pink noise, noises of making dishes, and noises of exercising bikes from the Google speech command dataset, as well as noises recorded in W311. Other non-key words, such as "Yes," "Stop," "Hi," and "Go," are collected from the Google speech command dataset.

The process of inference on the Raspberry Pi board involves feeding the audio input to the model, which classifies the sound event into one of the three classes. The detailed results of the model's performance are presented in the next section.

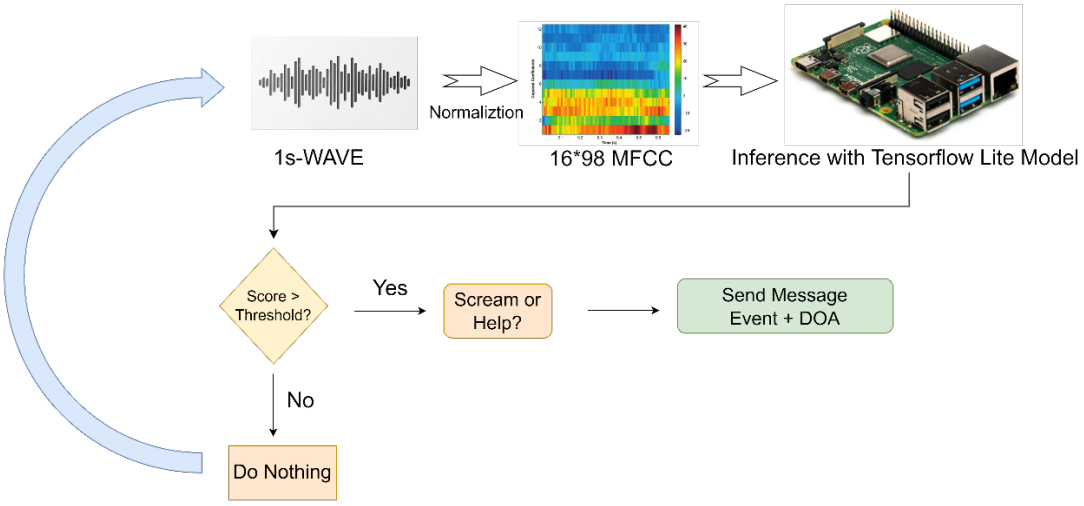


Figure 17: Process of inference on Raspberry pi board

The program starts by loading the TensorFlow Lite model. It continuously records 1-second audio, whose volume has been normalized, and computes MFCC to extract the features. The features are then input to the trained model, and each class is assigned a score based on SoftMax. If the score for the screaming or asking for help class is greater than 0.9, which is the threshold, the program sends a message indicating the direction of the event to Temi for further action.

3.4 Experimental Results and Discussions

3.4.1 Results of the speech recognition system

The model is trained on the cloud server "智星云," and GeForce RTX 2080 Ti is used as GPU for training. The hyperparameters setting is summarized below:

Batch Size	Learning Rate	epochs	Shuffle	Dropout Rate
16	0.001	30	True	0.5

Table 2: Hyperparameters of model

The results after 30 epochs are summarized below:

Training Accuracy	Test Accuracy	Training Loss	Test Loss
0.9873	0.9219	0.2879	0.036

Table 3: Accuracy and loss of training and test

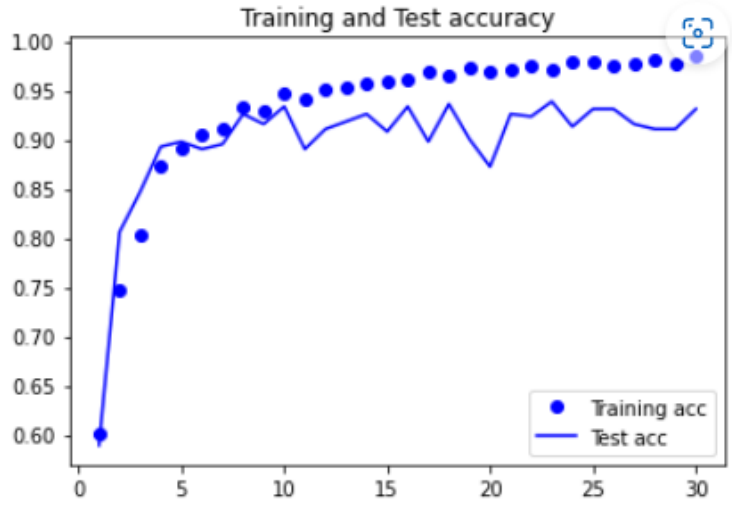


Figure 18: Accuracy of training and test

Figure 19: Loss of training and test

	Number	False Positive	False Negative	Accuracy
Scream	52	6	4	0.9412
Help	48	0	2	
Others	100	6	6	

Table 4: False Positive and False negative of Validation

3.4.2 Discussion of the Results

The data is shuffled and split into three parts: training, testing, and validation. Both the training and testing accuracies are greater than 0.9, which is satisfactory. However, the difference between the training and testing accuracies suggests that the model may be suffering from overfitting. This could be due to the mixed quality of the data, which includes recordings from both the Google Speech command dataset and recordings made by me at W311. The Google dataset may have been recorded using higher-quality equipment and methods, which could have affected the overall performance of the model.

The diagrams show that the model is able to converge, and its overall performance improves until reaching convergence. Additionally, the validation data shows relatively low rates of false positives and false negatives, indicating that the model has satisfactory classification abilities for speech.

During the demo, I observed that my speech recognition system was able to successfully detect screams from people whose sound was not included in the training data. However, it had limited performance in detecting requests for help from others whose sound had not been trained for the model. Specifically, the softmax score was above 0.9 when I requested help in the demo, while the score was only around 0.01 to 0.2 when others requested help. This suggests that the speech recognition system is speaker-dependent when it comes to detecting people seeking help. This is likely due to the limited source of the dataset used to train the model, which only included a small number of people requesting help, as opposed to the thousands of people in datasets such as the Google Speech Command dataset. To address this issue, it is recommended to either design a more comprehensive dataset of people seeking help, or revise the algorithm used (such as not using MFCC and deep learning).

4. Sound Source Localization

4.1 Introduction

Sound source localization is the process of identifying the location of a sound source in a given environment. It is an important area of research in fields such as robotics and audio signal processing. The ability to locate sound sources is crucial for many applications, such as speaker tracking in video conferencing, acoustic surveillance in security systems, and noise reduction in hearing aids [15].
Sound source localization can be used in robotics to improve the performance and functionality of robots. By accurately locating the source of a sound, robots can better perceive their environment and respond appropriately to auditory cues.

DOA, or Direction of Arrival, is a method for determining the direction from which a sound wave is arriving at a microphone array. It is used in sound source localization and can be used to estimate the location of a sound source in space. To determine the DOA, a microphone array with at least two microphones is required. The microphones should be placed at known locations in relation to each other. When a sound wave arrives at the array, it will reach each microphone at a slightly different time. By analyzing the time difference of arrival (TDOA) between the microphones, the DOA can be estimated [16].

4.2 Problem definition and Objectives

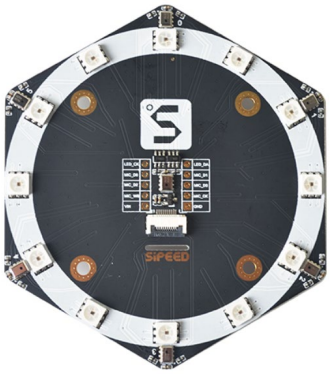
The speech recognition system mentioned above can detect dangerous events such as screaming or people calling for help. However, it is equally important for the intelligent security robot to locate the direction of these events. To achieve sound source localization in this project, the TDOA method has been chosen. Once a dangerous event is detected, the Raspberry Pi board will send a message with

the direction of the event to Temi for further action.

4.3 Methodology and Results

There are two kinds of microphone arrays designed for sound source localization. The first microphone array is the Sipeed Microphone [17], which consists of six microphones along the board and a center microphone.

There are twelve LEDs and each corresponding to pi/6 degree. The code is attached below.



```
imga = mic.get_map()    # Get images of sound source distribution
b = mic.get_dir(imga)   # Calculation, acquisition of sound source direction
for i in range(len(b)): # len b is 12, every b correspnds to 30 degree
    if b[i]>=1: # Remove noise and non-zero values
        AngleX+= b[i] * math.sin(i * math.pi/6)
        AngleY+= b[i] * math.cos(i * math.pi/6)
AngleX=round(AngleX,6) #Calculating coordinate conversion values
AngleY=round(AngleY,6)
if AngleY<0:AngleAddPi=180
if AngleX<0 and AngleY > 0:AngleAddPi=360
if AngleX!=0 or AngleY!=0: #Parameter Correction
    if AngleY==0:
        Angle=90 if AngleX>0 else 270 #Filling the X-axis angle
    else:
        Angle=AngleAddPi+round(math.degrees(math.atan(AngleX/AngleY)),4) #Calculation angle
    AngleR=round(math.sqrt(AngleY*AngleY+AngleX*AngleX),4) #Calculation intensity of the signal
    mic_list.append(AngleX)
    mic_list.append(AngleY)
    mic_list.append(AngleR)
    mic_list.append(Angle)
a = mic.set_led(b,(0,0,255))
return mic_list #return a list. [X, Y, intensity, angle]
```

Figure 20: Sipeed Microphone array and part of codes to implement it

The parameter b in the figure consists of twelve elements, each representing the intensity of sound in a particular direction. The intensity is measured on a scale of 0-15 and is divided into intervals of one pi/6. Let AngleX[i] be equal to r[i]sin(iπ/6), with the Y-axis set to 0 degrees. The angle from the LED to the Y-axis can be represented as iπ/6. Let the sound intensity be denoted as r, then the sound in the direction of the i-th LED can be decomposed as
AngleX[i]= r[i]sin(iπ/6) and AngleY[i]= r[i]cos(iπ/6) [18].

To calculate the direction of the sound, we use the formula θ[i] = arctan(AngleX[i]/AngleY[i]), and the intensity can be calculated using the formula R[i]^2 = AngleX[i]^2 + AngleY[i]^2. We then sum the values of AngleX[i] and AngleY[i] to obtain the values of Angle X and Angle Y, respectively. The direction of the sound is given by θ=arctan(AngleX/AngleY), and the intensity is the sum of the squares of the component intensities. The link to the demo video of Sipeed is attached below:
https://drive.google.com/file/d/1SfuqavJ5ZswfQEJPrAP_SusAz4drW2NN/view?usp=sharing

Despite its benefits, the Sipeed microphone array has several disadvantages. Firstly, it only supports the K210 microcontroller board, which can make integrating our speech recognition and sound source localization system with Temi more challenging. The results of each system are transferred independently, which adds complexity to the integration process for my teammates. Additionally, the DOA calculation algorithm is embedded in the hardware and cannot be easily tuned or modified if necessary, which limits its flexibility. Furthermore, the microphone array is susceptible to disruption from noise, which can cause errors in sound source localization, especially in noisy environments like W311.

Taking these factors into account, I have selected the Respeaker microphone array [19], which consists of four microphones, to achieve sound source localization. This array is compatible with the

Raspberry Pi board and will allow for easier integration with our project. Additionally, it provides more flexibility in parameter tuning and has better noise suppression capabilities, making it more reliable for sound source localization tasks.

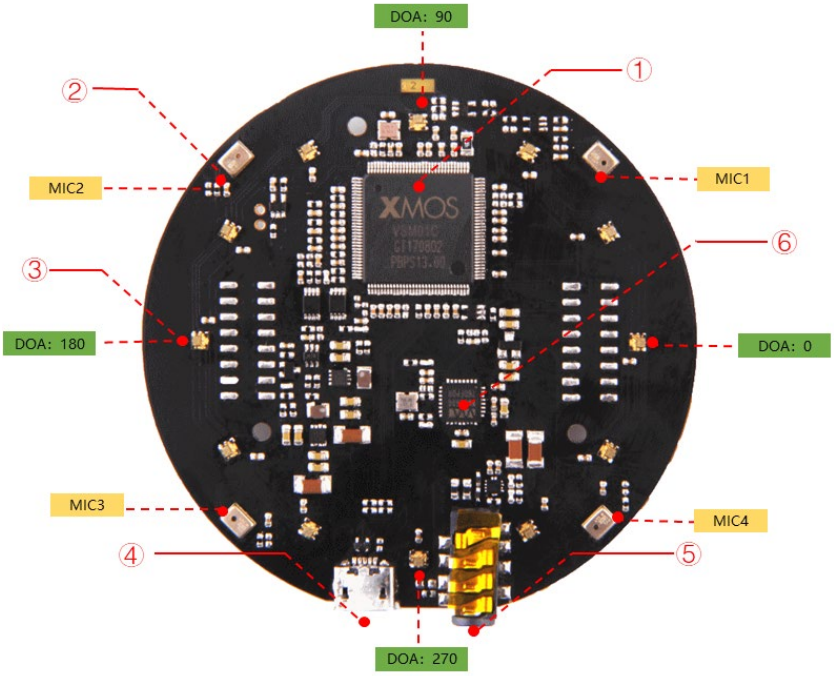


Figure 21: structure of the Respeaker Microphone array

The Respeaker microphone array has several advantages over the Sipeed array. Firstly, it is compatible with the Raspberry Pi, allowing for simultaneous operation of the speech recognition and sound source localization systems. This allows the integrated system to send the angle to Temi only when a dangerous sound event is detected. Secondly, the open-source algorithm of Respeaker makes it easy to modify the function and tune the parameters to suit our needs.

The structure of the Respeaker array consists of four microphones that are evenly distributed around the board. Microphone 1 and Microphone 3, as well as Microphone 2 and Microphone 4, are placed opposite each other. The distance between opposite microphones is approximately 64.63 mm, and the sampling rate is 16000 [18]. After filtering out high-frequency components, an unbiased cross-correlation function is computed for opposite microphones. The estimated TDOA is obtained by finding the time difference with the largest cross-correlation value. Suppose TDOA_x_axis is the TDOA calculated from microphone 1 and microphone 3, and TDOA_y_axis is the TDOA calculated from microphone 2 and microphone 4. We can then convert TDOA to angle and approximate the direction of arrival (DOA).

The procedure for calculating the direction of arrival (DOA) in a 4-microphone array can be expressed using the following mathematical formulas:

- 1. Microphone array setup: Let the four microphones in the array be denoted as M1, M2, M3, and M4, with known positions relative to each other.
- 2. Signal acquisition: Let the signal received by each microphone be denoted as s1(t), s2(t), s3(t), and s4(t), respectively.
- 3. Preprocessing: The recorded signals are preprocessed to remove any noise, reverberation, or echo. This may involve filtering the signals, applying noise reduction techniques, and aligning

the signals in time.

- 4. Time delay estimation: Let the time delay between the signals received by M1 and M2 be denoted as τ1, between M1 and M3 be denoted as τ2, and between M1 and M4 be denoted as τ3. The time delay estimates can be calculated using cross-correlation or phase difference methods:

  $\tau 1 = \arg\max\tau\ (s1(t) * s2(t-\tau))$

  $\tau 2 = \arg\max\tau\ (s1(t) * s3(t-\tau))$

  $\tau 3 = \arg\max\tau\ (s1(t) * s4(t-\tau))$

  where * denotes the cross-correlation operation, and arg maxτ denotes the time delay that maximizes the cross-correlation.
- 5. Angle of arrival estimation: The angle of arrival of the sound wave can be estimated using the time delay estimates and the known geometry of the array. Let d be the distance between adjacent microphones in the array, and let θ be the DOA. Then, the following equations hold:

  $\tau 1 = d \sin(\theta)$

  $\tau 2 = d \sin(\theta + 90°)$

  $\tau 3 = d \sin(\theta + 180°)$

  Solving these equations for θ yields:

  $\theta = \sin^{-1} [(\tau 1 + \tau 3) / (2d)]$

- 6. Output: The estimated DOA is outputted as an angle relative to the normal axis of the microphone array.

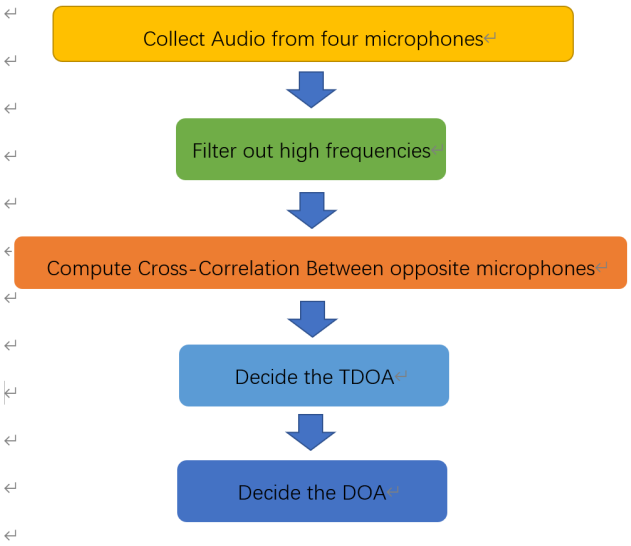


Figure 22: Procedure of deciding DOA

This method allows us to accurately estimate the direction of arrival of a sound signal using the Respeaker microphone array.

The link to the demo video of Respeaker is attached below:
https://drive.google.com/file/d/1bknuXxGhSnK-nzt7fqUguBL9LKT_ACQS/view?usp=sharing

5. Conclusion

As part of this inter-departmental project, I have designed a sound event detection system that is capable of detecting anomalous events such as screaming and people seeking help. Additionally, I have designed a sound source localization system that can determine the direction of arrival of sound events. When combined, these two systems provide the smart security robot Temi with the ability to locate the source of anomalous sounds, thereby improving the robot's performance and functionality.

Reference List:

[1]R. Ke, Y. Zhuang, Z. Pu, and Y. Wang, "A Smart, Efficient, and Reliable Parking Surveillance System With Edge Artificial Intelligence on IoT Devices," IEEE transactions on intelligent transportation systems, vol. 22, no. 8, pp. 4962–4974, 2021, doi: 10.1109/TITS.2020.2984197.

[2] "Temi - the personal robot (Hong Kong)," *temi robot HK*. [Online]. Available: https://www.robotemi.com.hk/. [Accessed: 16-Dec-2022].

[3]"Winston6 - Overview," *GitHub*. [Online]. Available: https://github.com/winston6. [Accessed: 16-Dec-2022].

[4]W. Dai, C. Dai, S. Qu, J. Li, and S. Das, "Very Deep Convolutional Neural Networks for Raw Waveforms," 2016, doi: 10.48550/arxiv.1610.00087.

[5] E. Cakir, G. Parascandolo, T. Heittola, H. Huttunen, and T. Virtanen, "Convolutional recurrent neural networks for Polyphonic Sound Event Detection," IEEE/ACM Transactions on Audio, Speech, and Language Processing, vol. 25, no. 6, pp. 1291–1303, 2017

[6] "UrbanSound8K," Urban Sound Datasets. [Online]. Available: https://urbansounddataset.weebly.com/urbansound8k.html. [Accessed: 16-Dec-2022].

[7]F. Ramzan et al., "A Deep Learning Approach for Automated Diagnosis and Multi-Class Classification of Alzheimer's Disease Stages Using Resting-State fMRI and Residual Neural Networks," Journal of medical systems, vol. 44, no. 2, pp. 37–37, 2020, doi: 10.1007/s10916-019-1475-2.

[8]Bhadragiri Jagan Mohan and Ramesh Babu N., "Speech recognition using MFCC and DTW," *2014 International Conference on Advances in Electrical Engineering (ICAEE)*, 2014.

[9]“Crypto,” *Practical Cryptography*. [Online]. Available: http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/. [Accessed: 14-Mar-2023].

[10] I. Kamarulafizam, S.-H. Salleh, J. M. Najeb, A. K. Ariff, and A. Chowdhury, “Heart sound analysis using MFCC and Time Frequency Distribution,” *World Congress on Medical Physics and Biomedical Engineering 2006*, pp. 946–949.

[11] H. Jeon, Y. Jung, S. Lee, and Y. Jung, “Area-efficient short-time Fourier transform processor for time–frequency analysis of non-stationary signals,” *Applied Sciences*, vol. 10, no. 20, p. 7208, 2020.

[12] X. D. Huang and K. F. Lee, “On speaker-independent, speaker-dependent, and speaker-adaptive speech recognition,” *[Proceedings] ICASSP 91: 1991 International Conference on Acoustics, Speech, and Signal Processing*, 1991.

[13] S. Shahnawazuddin, N. Adiga, H. K. Kathania, and B. T. Sai, “Creating speaker independent ASR system through prosody modification based data augmentation,” *Pattern Recognition Letters*, vol. 131, pp. 213–218, 2020.

[14] J. R. Deller, J. G. Proakis, and H. J. H. L., *Discrete-time processing of speech signals*. New York: Institute of Electrical and Electronics Engineers, 2000.

[15] R. R. Fay and A. N. Popper, “Introduction to sound source localization,” *Sound Source Localization*, pp. 1–5.

[16] A. M. Zoubir, M. Viberg, R. Chellappa, and S. Theodoridis, *Array and statistical signal processing*. Amsterdam: Academic Press, 2014.

[17] “Micarray 麦克风阵列,” *MicArray 麦克风阵列 - Sipeed Wiki*. [Online]. Available: https://wiki.sipeed.com/hardware/en/modules/micarray.html. [Accessed: 14-Mar-2023].

[18] USTHzhanglu, “Maixpy/code/script/MIC at master · USTHZHANGLU/Maixpy,” *GitHub*. [Online]. Available: https://github.com/USTHzhanglu/Maixpy/tree/master/code/script/mic. [Accessed: 14-Mar-2023].

[19] “Respeaker MIC array v2.0: Seeed studio wiki,” *Seeed Studio Wiki RSS*. [Online]. Available: https://wiki.seeedstudio.com/ReSpeaker_Mic_Array_v2.0/. [Accessed: 14-Mar-2023].