Audio Privacy Protection

by

Ben Tierney

This Report is submitted in partial fulfilment of the requirements of the Honours Degree in Computer and Communications Engineering (DT021A) of Technological University Dublin

May 24th, 2021

Supervisor: Paul Leamy

Abstract

In this paper an algorithm is developed in MATLAB and is used for research into audio privacy protection. The aim of the algorithm is to obfuscate any speech that an input audio signal may have to render the speech unintelligible while preserving other salient audio features. The frequency of speech lies in a certain domain and using this knowledge it is possible to focus on this range of frequencies.

The idea is to reduce all samples in the range of speech to zero while retaining the samples outside of this range. A frame-by-frame analysis is used by the algorithm to test the input audio signal and the main quantities used to tell if a given frame contains speech are the root mean squared energy (RMS) values and zero-crossing rate (ZCR) values of the signal. The program will also map the time domain, RMS values and ZCR values of the input and audio signals.

This report will examine the effectiveness at using the RMS and ZCR of an audio signal to set thresholds from which to identify speech samples, and if other samples are affected by this process. It is possible that other auditory sounds that are not those of speech will lie within the range of speech samples.

Table of Contents

Table of Figures

1 Introduction

In the modern world, data has become an increasingly valuable commodity since the development and widespread use of the Internet. With most social media mobile applications recording audio constantly for voice activated assistant and then used for targeted marketing purposes, there is a need for added security for users. It is estimated that in 2019 over 110 million people in the US used a voice assistant at least once a month, accounting for roughly a third of the population [1].

EU regulations pertaining to data protection have impacted how companies like Google, Facebook, and Apple deal with personal data, and has changed the way these companies operate. There is significant concern about how data is both gathered and stored, and the potential costs involved with breaching EU GDPR regulations [2]. Recording audio signals is a relatively low-cost exercise that can be used to monitor the health of patients [3] and the status of industrial equipment [4] among other applications. However, there is the potential to inadvertently record private and sensitive conversations while trying to detect other audio features. This unintended consequence may nevertheless be in breach of EU regulations.

The main aim of this project is to identify the parts of an audio recording that contain speech and subsequent removal of those parts to produce an audio clip that still retains the background noises present in the initial recording.

While recorded audio will almost always contain various background noises, the samples to be focused on for this project will be those which contain speech. Different noises will possess different qualities and it is hoped that these qualities will make it easier to identify and distinguish between noises. Below in Figure 1 is a graph representing the discrete time domain of an audio recording. All notable sounds in this recording are that of speech.

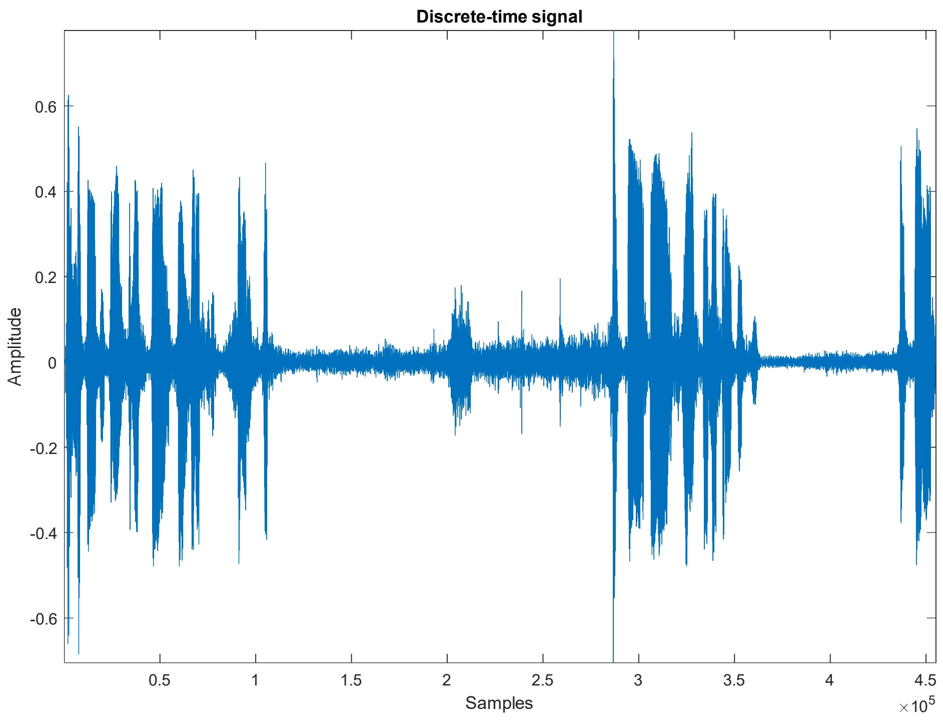


Figure 1: Discrete Time Domain of audio recording

The report is structured as follows. A background on speech frequencies is discussed to provide a base level of understanding for the reader with frequencies that speech typically lie in being detailed. An algorithm will be written using MATLAB and so other previous algorithms that have been created for use in speech detection will be explored.

An important point for consideration is the type of data that will be used to assess the performance of the algorithm. Various datasets are examined, and the dataset used for training and testing purposes is a subset developed using Google's Audioset. The Audioset contains labels for 10 second audio segments extracted from YouTube videos and has a wide range of audio clips to examine. Both male and female speech compose the subset and several clips contain background noises to further test the algorithm's accuracy. It is important to consider both genders for ethical considerations and to obtain balanced data. In a real-world scenario, it is equally possible that an input signal will contain either male, female or even both types of speech so this must be accounted for.

The root mean squared energy (RMS) and zero-crossing rate (ZCR) of input audio signals will be briefly discussed as the main features for analysis used for identification of speech samples in the input audio signals. The RMS values are calculated by performing a square root on the mean of the frame samples squared. The ZCR is a measure of how often the signal passes through the zero of the x-axes in the time domain. A suitable threshold must be found for both quantities for this purpose.

From Google's Audioset, speech labels need to be added manually to test the accuracy of the algorithm as the labels that the Audioset contains already are too general in marking 10 second clips where speech lies. Audacity is the program to be used to label speech samples of each test signal as a benchmark for accuracy. Results from testing will be a comparison between the benchmark labels and samples the algorithm labels as speech/non-speech. An additional test using artificial Gaussian white noise will be performed with a varying degree of signal-to-noise ratio.

2 Literature Review

2.1 Introduction

Although speech is an analogue signal, once it is converted to a digital signal via a microphone, it can be processed in the same way all digital signals can be. There is a need for study on the background nature of audio signals, and in particular speech, as different components will possess different qualities. The benefit of conducting research on audio processing is that there is a readily available source of knowledge from the Internet.

With speech being the main quantity of focus, it will be discussed over what frequencies these samples may lie and a background on the analysis of speech signals. The theory behind the sampling rate will be discussed and it will be shown why there is a need to have a minimum sampling rate with different types of frequencies.

Another thing that was considered before undertaking work on the project was the type of data to be used. As mentioned previously, one of the benefits of conducting work in speech processing is the amount of data readily available. Different datasets will be considered before choosing an appropriate one to compose the testing dataset.

Previous works in speech processing will also be examined. A focus will be placed on algorithms written for the purpose of identifying speech samples and differentiating from other samples within an audio file.

2.2 Background on Speech

A formant is a resonance in the human vocal tract's frequency response, whereas the frequency response of the vocal tract's is formed by numerous formants. These formants lie in the frequency range of between 300 Hz to 3500 Hz and thus this range will be the area of focus for extraction of speech samples [5].

There is a need for a minimum sampling rate for audio to avoid aliasing of the data of interest. According to Shannon's theorem on sampling rates, "If a function contains no frequencies higher than W cps, it is completely determined by giving its ordinates at a series of points spaced ½ W seconds apart" [6]. This means that the sampling rate must be at least twice that of the max frequency. From the information gathered in [5], it can be found that an audio signal containing speech must a minimum sampling rate of 7 kHz to avoid aliasing. The sampling rate of the audio clips used for processing in MATLAB is 44.1 kHz. Below in Figure 2 is an example of aliasing taken from Dataforth [7]. Note how the coloured lines are substantially different from the lines they are superimposed over due to a sampling rate lower than the minimum.

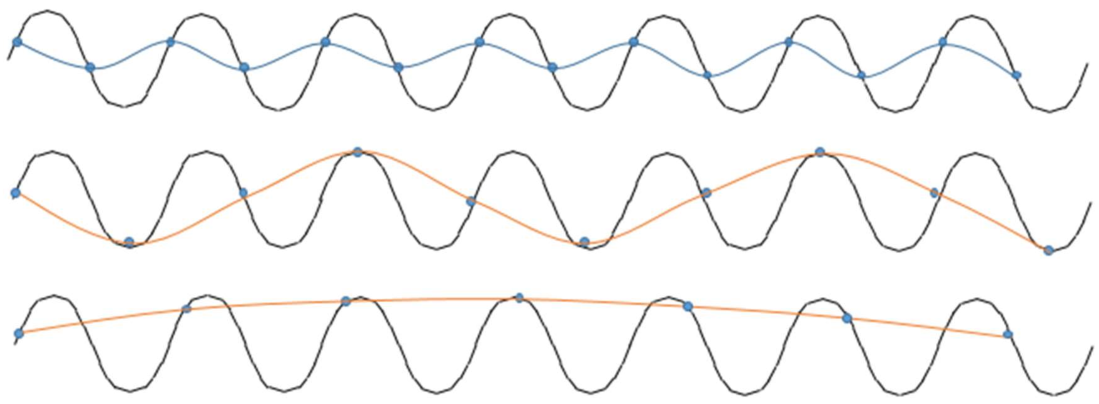


Figure 2 Example of aliasing

2.3 Datasets

One of the main considerations before undertaking the project is type of data that would be used for training and testing the algorithm. It is fundamental to have a dataset with varied data and without the same signals repeated so a greater degree of accuracy can be obtained. If the same input audio signals are used, then the same set of results will be produced by testing and any comments on accuracy will not be correct.

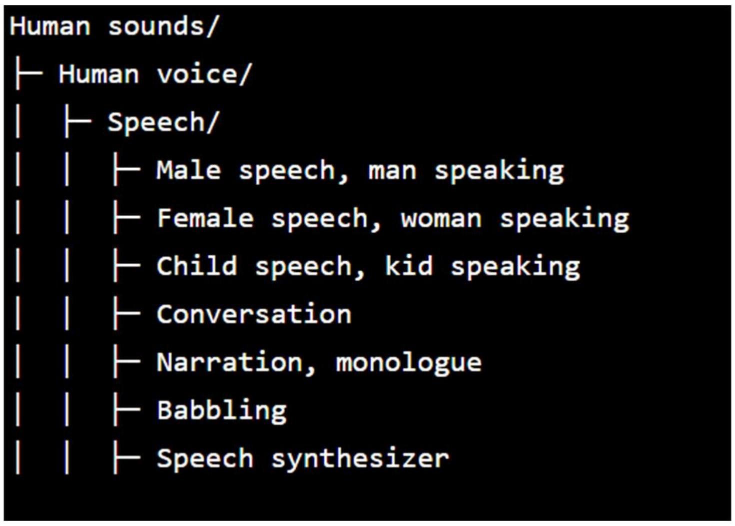
While many datasets exist for implementation in audio processing projects, an ideal dataset for the purposes of research would be a publicly available dataset. A dataset containing audio signals which can be downloaded anywhere in the world with internet access and can be used for implementation on an audio processing project/system is highly advantageous. If the work conducted in this project or other projects can be reproduced by other researchers, it makes future works and further research easier. For these reasons, implementation of a publicly available dataset will be examined further.

2.3.1 Speech Commands Dataset

A useful dataset to be considered for use in the project is the Speech Commands Dataset released by Google's AI blog [8]. The dataset was released into the public domain for audio recognition work. The dataset contains 65,000 one-second-long samples of 30 short words spoken by thousands of different people. There is a great degree of variability in this dataset, and it is readily available for use. The dataset was released under a Creative Commons BY 4.0 license which means it is free to be shared i.e., copied, redistributed, and adapted and built upon (even commercially) [9]. It will be considered for use in the project but there are limits to what is achievable with the dataset as the samples are noticeably short at durations of one second. Longer samples may be needed to examine more in-depth functions of the developed code/algorithm.

2.3.2 Google Audioset

Another dataset which may be more useful for the purposes of this project is another dataset from Google called Google's "Audioset" [10]. In contrast to the previous dataset, this one contains longer samples at a standard ten second duration per sample. The dataset is created from clips from human-labelled YouTube videos. The section to be the focus of this project is the speech section containing 1,010,480 overall videos [11]. With the previous one second samples, there may not be enough non-speech samples to give an accurate representation of the accuracy of the speech detection algorithm, but with longer ten second samples there is more flexibility for testing. Like the previous Google dataset, this dataset is covered under a Creative Commons BY 4.0 license and is widely available [9]. The section of interest in the ontology is distributed as shown in the ASCII tree in Figure 3.



```
Human sounds/
├─ Human voice/
│  ├─ Speech/
│  │  ├─ Male speech, man speaking
│  │  ├─ Female speech, woman speaking
│  │  ├─ Child speech, kid speaking
│  │  ├─ Conversation
│  │  ├─ Narration, monologue
│  │  ├─ Babbling
│  │  ├─ Speech synthesizer
```

Figure 3 Audioset ASCII tree

It is intended that the first Google dataset with one-second clips will be used initially to start writing the code for the algorithm but the second Google dataset with ten-second clips will be used for later implementation and final testing. Respective training and testing subsets will be created for these purposes.

One paper which implemented Google's Audioset for the purposes of privacy protection was one which used the dataset specifically for testing the impact of environmental noise [12]. Voice assistants such as Amazon's Alexa, mentioned in the paper's title, are the focus for the purposes of privacy protection. As mentioned previously, ethical concerns exist with modern software applications constantly recording audio and this paper seeks to address this with a companion device in the form of a jamming device called MicShield. While testing for experimental evaluations, the researchers use Audioset for speech noise and music noise. According to the paper, MicShield performs better with the addition of the two types of environmental noises when the signal-to-noise ratio is greater than 3dB and in this case the jamming effectiveness is not affected in the noisy environment.

2.4 Previous Works

Rather than writing an algorithm completely from scratch, it is preferable to study previous works of interest. Many previous works exist already in speech processing and each use different methods to distinguish speech from non-speech samples.

One example of an algorithm that process audio for speech detection is presented in [13], which uses several methods for automatically detecting speech . The article titled “A simple but efficient voice activity detection algorithm through Hilbert transform and dynamic threshold for speech pathologies” makes use of a few methods for speech detection. The primary method of speech detection is to create a dynamic threshold using the zero-crossing rate (ZCR) and root mean squared (RMS) values of the speech signal. The dynamic threshold is calculated by first normalising the data vectors obtained from the RMS and ZCR so they can be compared with the input signal. The maximum and minimum level of these two features is then used to calculate a scaling factor and thus the dynamic threshold can be calculated from these quantities. The dataset used to test the algorithm was created from political speeches and speech content was marked manually by an expert operator to test the accuracy of the algorithm. This dataset has not been made publicly available. Background noise was also added to the speech samples when testing the accuracy of the algorithm using artificial Gaussian white noise with a signal-to-noise ratio of 5 dB, 15dB and 20dB.

Another paper that contains some similarities to the previous work mentioned is a paper presented in [14]. The paper also deals with identifying sections of an audio sample that contain speech and notably uses MATLAB to perform this function. The main methods of determining speech in this paper are the zero-crossing rate of the signal and the short-time energy of the signal. The data used to test the algorithm was limited though in that only one speech sample with the word “four” spoken was used. One of the main methods of interest of this paper was the frame-by-frame analysis of the speech sample and identifying each frame by whether it contained speech or not.

A further paper which covers the classification of speech is one titled “A Speech/Music Discriminator using RMS and Zero-crossings” [15]. This paper specifically covers identifying the distinction between speech and music audio samples. Again, the calculated RMS values and zero-crossing rate of the signal are used as the main characteristics for classifying speech samples. It is important to note in this paper that the RMS variance for speech is higher than that of music and that there will be a significant number of null ZCR values for speech. The classification algorithm used by this paper makes use of these characteristics. Testing the algorithm on a dataset created from the Internet and audio CDs, the paper claimed to have an accuracy of 86% using the RMS values alone and then a final 95% accuracy with the other features.

All the previous works mentioned and outlined above will be considered when implementing the algorithm in this project. The RMS and zero-crossing rate will be the main characteristics discussed below for use in the project.

2.5 Root Mean Squared Energy

Using the root mean squared (RMS) energy of an audio signal is covered by other papers in the previous works section and appears to be a staple in the process of identifying speech samples. The RMS of the energy of the signal was also part of the methodology.

$$RMS = \sqrt{\frac{1}{n}\sum_{i} x_i^2}$$

Equation 1 RMS

2.6 Zero-Crossing Rate

Another quantity of interest in past works was the zero-crossing rate of the signal. The zero-crossing rate is the measure of the number of times the signal crosses the x-axis of the time domain from positive to negative or from negative to positive. The equation for the zero-crossing rate of a signal is shown below:

$$Z_j = \sum_{i=(j-1)\cdot N+1}^{j\cdot N} |sgn[x(i)] - sgn[x(i-1)]|$$

Equation 2 ZCR

2.7 Signal to Noise Ratio

A signal-to-noise ratio (SNR) can be used as a means of measuring an input signal to background noise present in the signal. In this context, the SNR will be used to test if the performance of the algorithm holds up after artificially increased noise levels.

$$SNR = \frac{P_{Signal}}{P_{Noise}}$$

Equation 3 SNR

Where P is average power. The SNR will be changed on a logarithmic scale for testing purposes and the units used in MATLAB are Watts (W). A built-in MATLAB function named *awgn* will be used to add white Gaussian noise to the input signal.

$$SNR_{dB} = 10\log_{10}\left(\frac{P_{Signal}}{P_{Noise}}\right)$$

Equation 4 SNR in dB

Below in Figure 4 is an example of adding Gaussian white noise to an input signal in the form of a basic sine wave. The MATLAB function measured the base power of the sine wave before adding noise with a SNR of 10 dB. It is important to note a lower SNR for adding Gaussian white noise produces a greater range of frequencies in the noise.

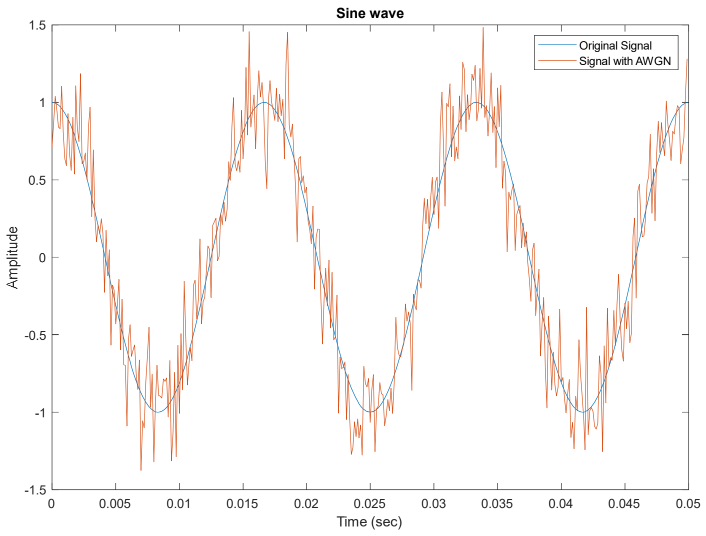


Figure 4 Sine wave with added Gaussian white noise

3 Implementation

This section covers the implementation of an algorithm for the use in audio privacy protection and relevant methods gathering data for analysis and on measuring the accuracy of the algorithm developed in MATLAB code. Most of the analysis will be performed on Google's Audioset and the bulk raw data from this dataset must be edited before testing the algorithm.

The implementation of the code will be carried out in MATLAB and the relevant toolboxes and associated functions will be used alongside code written. MATLAB is a useful programming platform for the purposes of audio processing due to the aid of these extra build-in functions and the user interface is also helpful for the quick indexing of relevant functions/arrays. A flow chart of the implementation in MATLAB code can be seen below in Figure 5.

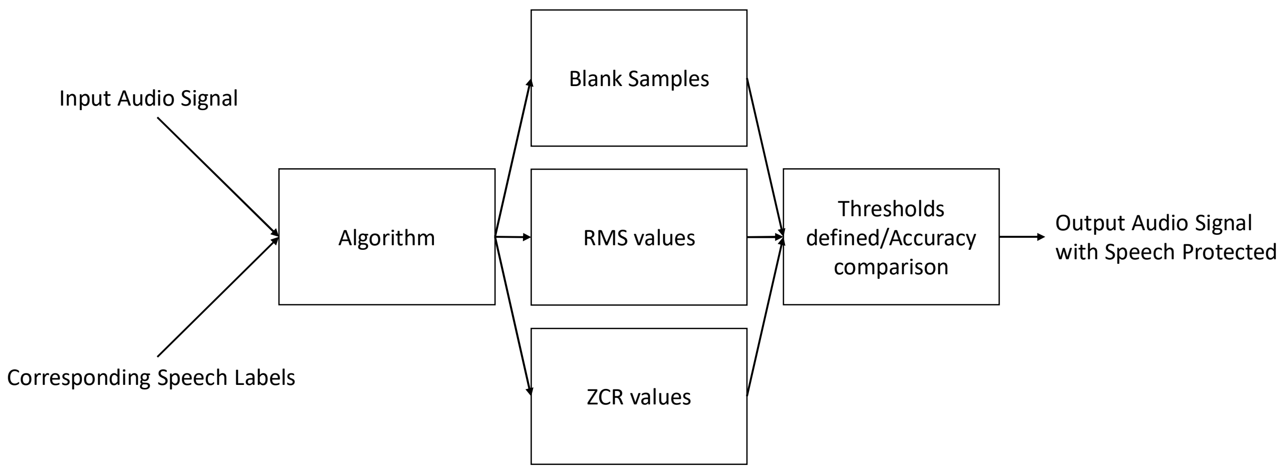

Input Audio Signal
Corresponding Speech Labels
Algorithm
Blank Samples
RMS values
ZCR values
Thresholds defined/Accuracy comparison
Output Audio Signal with Speech Protected

Figure 5 Flow Chart of overall algorithm functionality

3.1 Framework for measuring performance

The main framework for measuring the performance of the algorithm implemented in MATLAB are the user defined speech labels created using the program Audacity. From this program it is easy for the user to visualise the signal in the discrete time domain while the audio is also playing and thus correctly label speech samples. This must be done for ever audio signal to be tested to have a benchmark for accuracy. The labels are exported to a text file and this file is to be read into MATLAB situationally with the audio signal. Each audio signal must have a relevant labels text file attached to it or it will not be possible to calculate the accuracy and performance metrics of the algorithm.

From the labels text file, a binary vector is created the same length of the input audio signal, that contains markings for the content of speech in the time domain. A binary 1 value indicating speech and a binary 0 value indicating non speech samples. This vector will be compared against another vector that has been marked by the algorithm via an AND operation. This operation can be subdivided based on whether the values do or do not match and on the binary value to give true positives/negatives (TP/TN) and false positives/negatives (FP/FN). Once these quantities are obtained, the confusion matrix of the system and performance metrics can be calculated.

An easy way to interpret the results given by the algorithm is to show them clearly on a confusion matrix. A confusion matrix is a way of representing the information by matching predicted classes with actual classes. Between the benchmark true class and the predicted class created by the algorithm, the results and accuracy of the predictions can be indexed visually by the simple confusion matrix chart. In the case for this paper, the algorithm makes predictions on what samples belong to that of speech, with a 1 being a sample labelled as speech. Below is a sample confusion matrix in Figure 6, the layout of which will be used by the built-in function in MATLAB for later results.

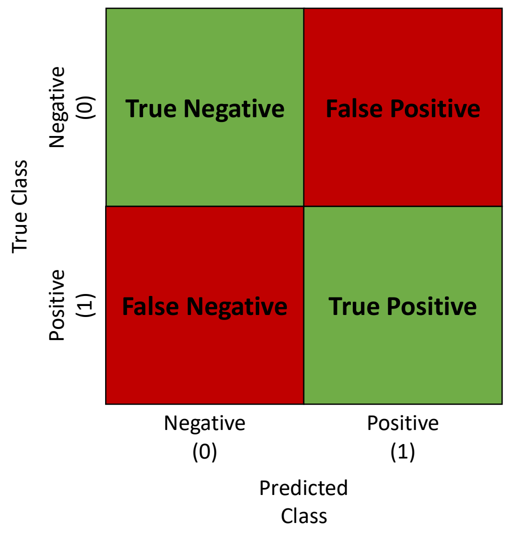


Figure 6 Confusion Matrix

The performance metrics used for better understanding of the functioning of the algorithm and its accuracy are derived from the values mentioned in the previous paragraphs. Once the true positives/negatives etc. are obtained, then the precision, sensitivity, specificity, and negative prediction value (NPV) may be calculated. The equations for calculating the performance metrics are detailed below:

$$Sensitivity = \frac{TP}{TP + FN}$$

$$Specificity = \frac{TN}{TN + FP}$$

$$Precision = \frac{TP}{TP + FP}$$

$$NPV = \frac{TN}{TN + FN}$$

3.2 Procedure for gathering and downloading data

Google Audioset was the dataset chosen to use for training and testing the algorithm mainly due to the variety of the audio clips it is composed of. Two subsets of data were created with one being used for training the algorithm and another being used for performing tests and obtaining results.

From the YouTube links for each audio clip, a full-length video can be downloaded in a MP3 format using a simple YouTube downloader application [16]. The ontology of the Audioset is described in section 2 previously, but the balanced training and testing subsets will be composed of clips from the subset of speech labelled primarily “male speech, man speaking”, “female speech, woman speaking” and “narration, monologue”. Any other labels attached to a clip will be noted so it will be possible to comment on the effect on the accuracy of the additional label, for example, music, conversation. Once downloaded, the full audio clips must be trimmed according to the start and end times specified in the CSV file. Again, a basic application called Music Editor on Windows can perform this task. A training subset of ten audio clips of 10 seconds long was used for initially training the algorithm before a subsequent unseen testing subset of ten audio clips of 10 seconds long was used for gathering results. The sampling rate used for these MP3 files is 44.1 kHz and thus a 100 ms frame used for analysis will contain 4410 samples.

3.3 Problem with Audioset annotations

Accompanying the Audioset ontology is a CSV file containing the details for each audio clip. There is a minor problem with the CSV file for the purposes of this project in that the labels it contains are too general. As mentioned previously, the start and end times for each audio clip taken from their respective YouTube videos only indicate a general section where the speech content lies and does not label specific samples. This problem was overcome by creating specific labels for speech in Audacity and exporting the labels to a text file for each audio clip. Below in Figure 7 is the labelling process in Audacity.

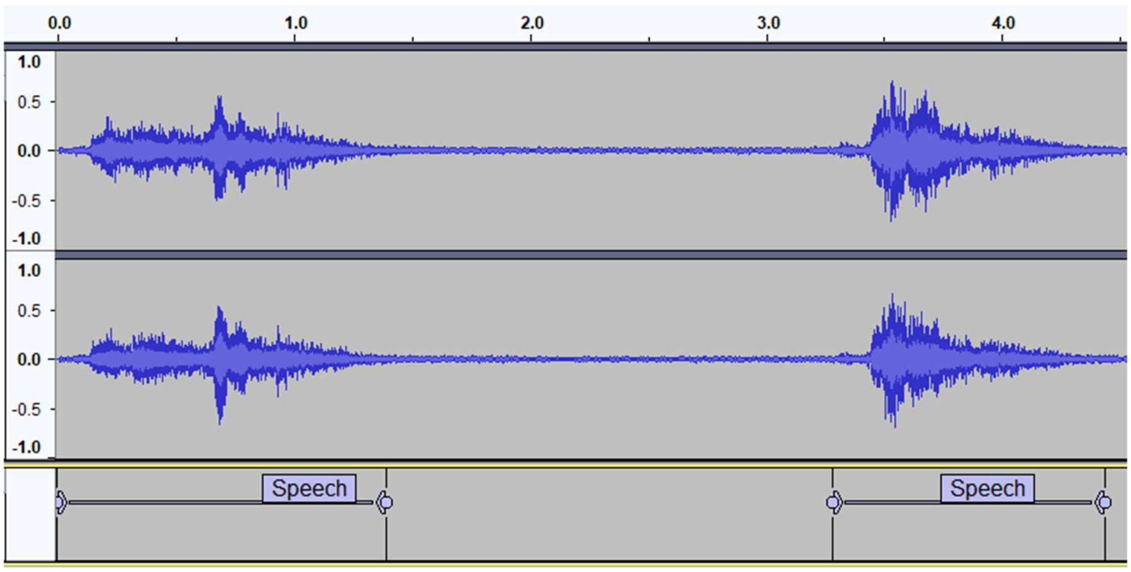
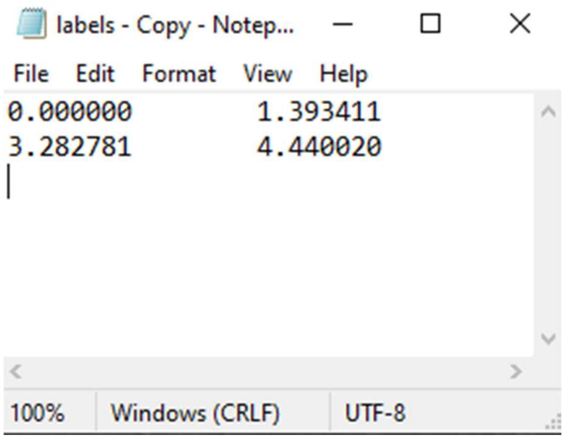


Figure 7 Audacity labelling

The labels are exported to a text file as shown in Figure 8 below in a numeric format in seconds for benchmarking the accuracy of the algorithm's performance. The labels are used to initialise a vector for the location of speech content which will be detailed further in the proceeding section.



```
0.000000	1.393411
3.282781	4.440020
```

Figure 8 Exported labels text file

3.4 Implementation of MATLAB code

The section-by-section implementation of MATLAB code is detailed in the corresponding subsections below. Figure 9 represents the output of the code, with graphs of the discrete time domain, RMS values and ZCR values displayed below.

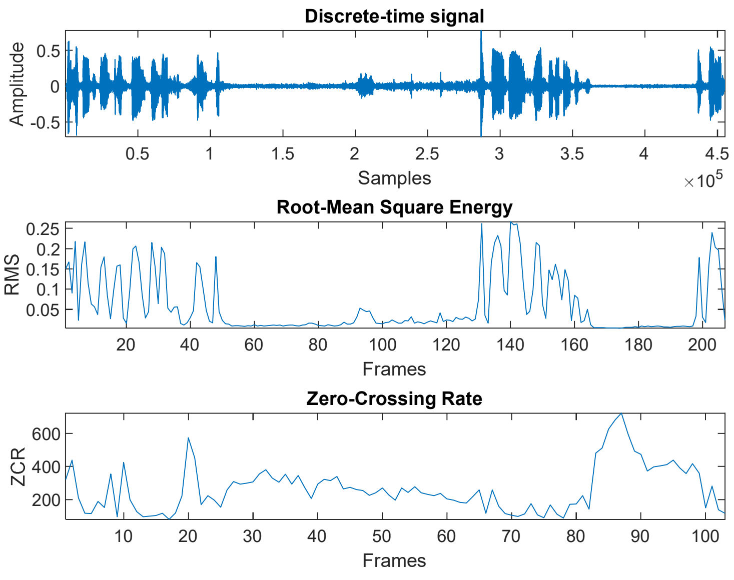


Figure 9 Graphs of MATLAB code functions

3.4.1 Section 1. Read in audio signal and labels

One of the first parts of code to implement is to read in the audio signal to be processed. The audio files for training/testing the algorithm are in an MP3 format and each one has an associated labels file in TXT format indicating the speech samples of interest.

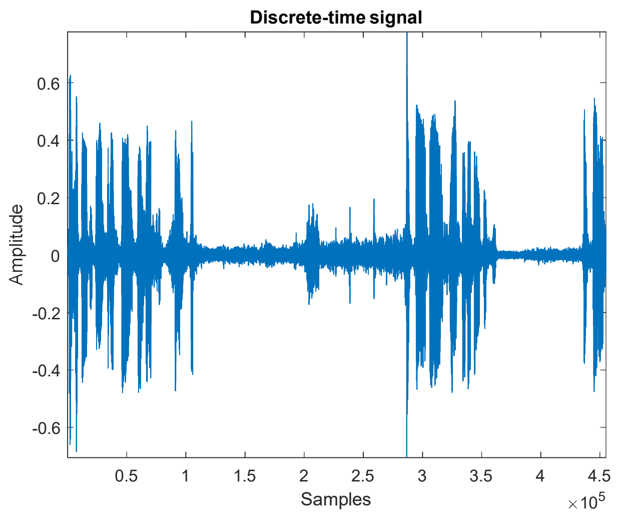


Figure 10 Sample input Audio signal

A vector called accuracyVector is created that contains a corresponding value for each value in the audio file read in initially for the purposes of benchmarking the accuracy of the speech detection algorithm. This vector is used for the purpose of identifying which sample corresponds to a sample possessed by a speech segment with the vector being a binary quantity with 1 representing speech and 0 representing any value other than speech. From the values in the labels text file, the vector is initialised. It can be visualised in Figure 11 below with the accuracyVector values shown in red.

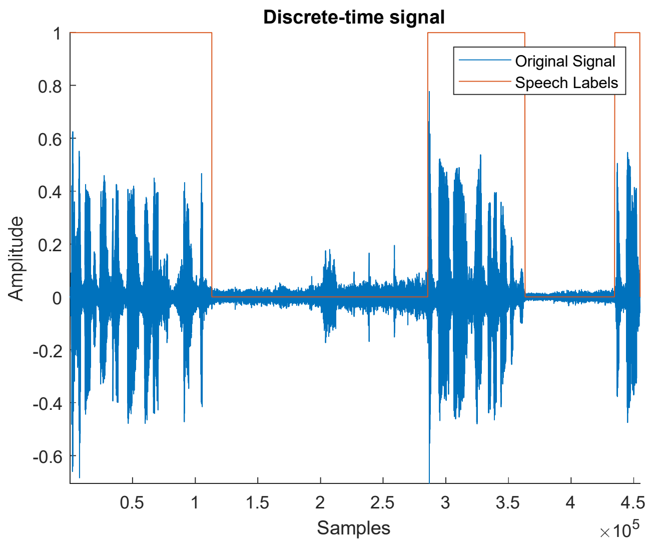


Figure 11 Audio signal with corresponding speech labels

3.4.2 Section 2. Identification of blank samples

The next step in the processing of the audio file is the identification of blank data in the file. These samples will have an amplitude of close to zero in the time domain, so it is easy to set a lower threshold for labelling of these samples. Like how the accuracyVector is used as vector to identify speech samples, the accuracy of the algorithm from using the minimum amplitude threshold is measured by the speechVector which is created to be the same length as the input audio signal. Using a frame-by-frame analysis with a frame size of the sampling rate * 0.05, the max value in the frame is compared against the minimum threshold set, in this case at 0.1. If the max value in the frame is above the threshold, then all values in the frame as set as binary 1 in the speechVector.

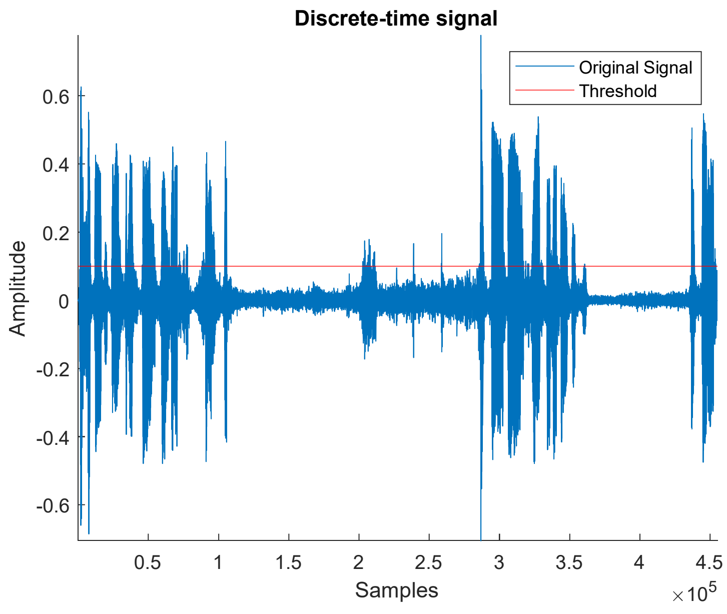


Figure 12 Audio signal with added threshold

Shown in Figure 12 is the sample signal in the discrete time domain with a minimum threshold set at 0.1

3.4.3 Section 3. Extract Root-Mean Square Energy

The RMS values of the input audio signal are extracted by the rmsEnergyValues function using a frame-by-frame analysis. The function takes the input audio signal and uses a frame the size of the sampling rate * 0.05. The values calculated by the function are returned to the array finalRmsValues. The RMS values of the sample input audio signal can be visualised in Figure 13 below.

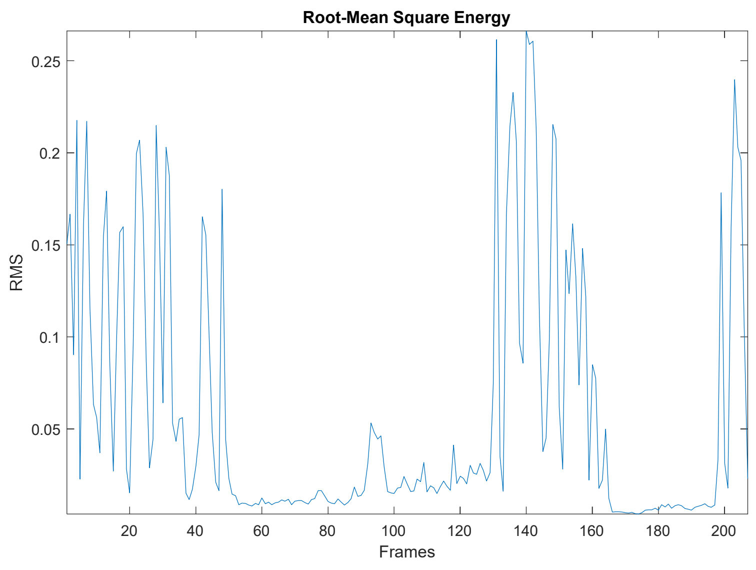


Figure 13 Root-Mean Square values

In the same way as the previous vectors, the accuracy of the algorithm from using the RMS threshold is measured by the rmsVector which is created to be the same length as the input audio signal. Depending on the threshold set for the RMS values, any frames with values above the threshold will be entered as a binary 1 value in the rmsVector. For the first test, the RMS threshold was set to 0.1.

A problem that was observed when implementing a threshold for the RMS values was that a lower value in a so called "block" of speech samples would affect the rmsVector by giving a false negative reading. To counter this problem, a condition was added that for a frame to be recorded in the vector as non-speech, there must be 5 previous frames in a row with the same values, so that there must be a break of 0.5 seconds of RMS values lower than the threshold indicating a break in speech. This is to prevent outliers in the middle of a block of speech samples from zeroing a frame and helps to produce a more accurate reading.

3.4.4 Section 4. Extract Zero-Crossing Rate

The ZCR values are extracted by the zcrValues function using a frame-by-frame analysis like that of the rmsEnergyValues function of the previous section. The zero-crossing rate measures how many times the signal crosses the x-axis in each window. The ZCR values of the sample input audio signal can be visualised in Figure 14 below.

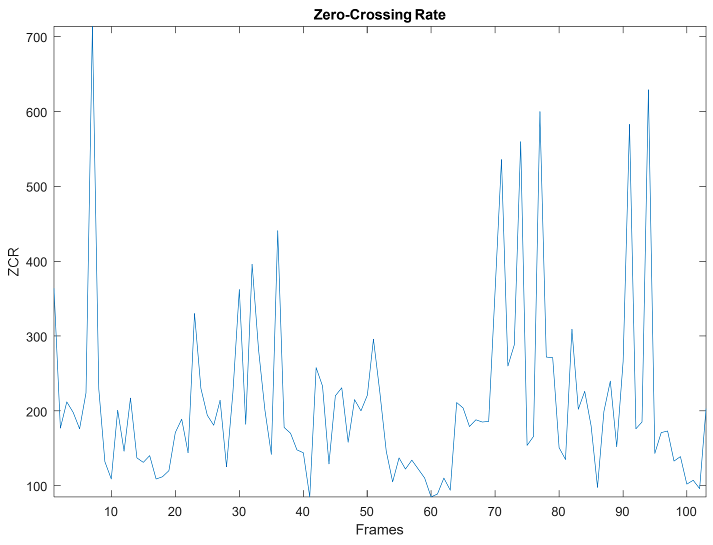


Figure 14 Zero Crossing Rate

In the same way as the previous vectors, the accuracy of the algorithm from using the ZCR threshold is measured by the zcrVector which is created to be the same length as the input audio signal. It is expected that lower values for the ZCR will indicate speech and higher values will indicate non-speech. A general threshold was set for 100 with speech samples expected to be below this threshold.

3.4.5 Section 5. Accuracy test & Confusion matrix

This section in the MATLAB code is to primarily give measurements for the accuracy and performance metrics of the algorithm. As mentioned previously, the user defined speech labels are the benchmark of accuracy for the algorithm. The labels are read in from the text file that corresponds to the audio signal and used to initialise the accuracyVector via the index values. The speech samples identified by the algorithm using the previous methods are stored in the testVector using an AND operation. The testVector is used to initialise a new signal from the input signal with its speech content removed and privacy protected.

The accuracyVector and testVector are compared and the true positives/negatives and false positives/negatives are recorded manually with a for loop. From these values the accuracy, precision, sensitivity etc. can be calculated and an overall accuracy for the algorithm can be obtained. A confusion matrix with the same values as the manually calculated values can be obtained by using the built in MATLAB functions confusionmat/confusionchart and a sample matrix is displayed in Figure 15 below. Note the true negatives are in the top left box and true positives in the bottom right.

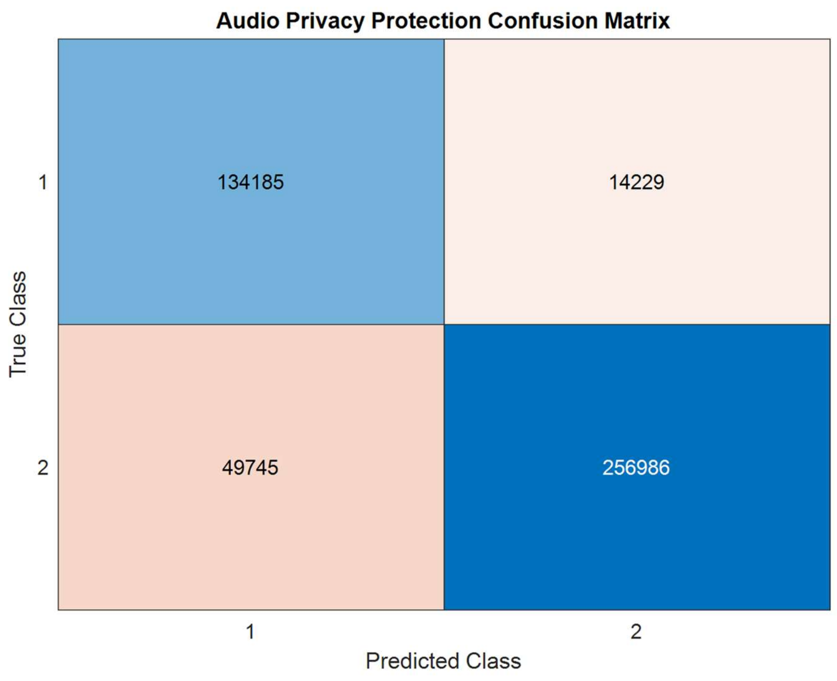


Figure 15 Sample Confusion Matrix

4 Results

This section covers the results obtained after testing the performance of the algorithm on the audio signals of the testing subsets. While all signals are the same in containing speech content, a few audio signals will possess slightly different labels according to the Audioset CSV file. Shown in Table 1 below is the testing set with the labels from Audioset for each signal. This subset will give a measure of the performance of the algorithm developed in MATLAB when the performance metrics are analysed.

Signal	Set	Conversation	Male speech	Female speech	Narration, monologue	Music
test1	Male speech	yes	yes			
test2	Male speech		yes			
test3	Male speech		yes			
test4	Male speech		yes			
test5	Male speech		yes			yes
test6	Male speech		yes			yes
test7	Male speech		yes			
test8	Female speech			yes		
test9	Female speech			yes	yes	
test10	Female speech			yes	yes	

Table 1 Test set composition

4.1 Results using blank samples/RMS

For this section, only the identification of blank samples and RMS values were used to identify speech. The vectors containing the samples identified as probable speech by the blank samples method and those identified by the RMS values were put through an AND operator to add the vectors together. The results when compared with the benchmark vector generated from the Audacity labels are shown in Table 2 below.

Signal	Accuracy %	Precision %	Sensitivity %	Specificity %	NPV %
test1	89.7	87.26	91.11	88.48	92
test2	91.42	99.05	86.03	98.86	83.67
test3	87.4	92.84	88.1	85.96	77.75
test4	97.57	97.94	99.38	81.72	93.8
test5	87.78	85.66	95.6	76.09	92.06
test6	67.24	50.96	64.69	68.52	79.33
test7	91.45	78.05	76.11	95.02	94.47
test8	97.25	97.17	99.77	79.15	97.97
test9	73.44	100	56.68	100	59.29
test10	91.92	91.9	99.71	29.08	92.59
Average:	**87.517**	**88.083**	**85.718**	**80.288**	**86.293**

Table 2 Performance metrics results from blank sampling/RMS

There is a reasonably high degree of accuracy in identifying speech in the test signals. Test6 and test9 signals were slight outliers in that all other audio signals had a general accuracy of above 85%. A possible explanation for the test6 signal is the presence of music visible from the Audioset labels in Table 1. The test9 signal was a quieter audio signal with a lower amplitude present throughout and it is thought this affected the performance metrics.

The average accuracy for the test was 87.5% indicating a good overall performance. The algorithm had a high precision at 88% meaning that when the algorithm did label a sample as speech, it was more than likely a correct labelling. The other performance metrics also held high standard but with specificity being the lowest, it indicated that the algorithm slightly lacked in labelling a negative sample as truly negative.

4.2 *Results with added ZCR characteristic*

The samples identified as probable speech by the algorithm using ZCR values were put through an AND operator with the same vectors from the previous methods. The results when compared with the benchmark vector generated from the Audacity labels are shown in Table 3 below.

Signal	Accuracy %	Precision %	Sensitivity %	Specificity %	NPV %
test1	89.7	87.26	91.11	88.48	92
test2	91.42	99.05	86.03	98.86	83.67
test3	87.2	89.68	91.54	78,22	81.73
test4	97.57	97.94	99.38	81.72	93.8
test5	86.82	84.43	95.61	73.67	91.82
test6	41.53	34.89	85.57	19.27	72.54
test7	91.45	78.05	76.11	95.01	94.47
test8	97.25	97.17	99.77	79.15	97.97
test9	73.86	96.71	59.38	96.8	60
test10	91.92	92.9	99.71	29.08	92.59
Average:	**84.87**	**85.81**	**88.42**	**73.56**	**86.06**

Table 3 Performance metrics from added ZCR

In general, the results were similar to the results from section 4.1 with no noticeable improvements in performance of the algorithm. It was difficult to find a general ZCR threshold that fit each test signal appropriately as each signal had a varying range of ZCR values.

Due to the slight drop in overall accuracy, this characteristic was discounted during further testing with artificially added Gaussian white noise.

4.3 Additional Gaussian white noise

Following the initial results gathered from testing the algorithm, Gaussian white noise was added to the input test signal with a built in MATLAB function. Each test signal was then run through the algorithm using the blank sampling method and RMS for analysis. The ZCR was discounted due to a slight drop in performance shown in section 4.2. The SNR was varied between 20 dB, 10 dB and 5 dB with lower values expected to reduce accuracy. The results from the original testing in 4.1 with natural noise is also shown in Table 4 below.

SNR:	**Natural noise**	**20 dB**	**10 dB**	**5 dB**
Signal	**Accuracy %**	**Accuracy %**	**Accuracy %**	**Accuracy %**
test1	89.70	82.92	78.43	46.20
test2	91.42	90.22	89.16	57.81
test3	87.40	73.65	67.59	67.59
test4	97.57	91.98	89.56	89.56
test5	87.78	82.29	73.18	60.10
test6	67.24	69.53	61.55	43.42
test7	91.45	90.48	92.42	86.61
test8	97.25	97.30	87.56	87.56
test9	73.44	69.58	71.02	75.68
test10	91.92	89.06	89.06	89.06
Average:	**87.52**	**83.70**	**79.95**	**70.36**

Table 4 Gaussian noise results

The results turned out as expected with a lower SNR degrading the accuracy of the algorithm. This was most noticeable at a SNR of 5 dB in which the average accuracy was about 70%. The algorithm's performance withstood most of the added Gaussian white noise with the overall accuracy dipping just slightly below 80% for a SNR of 10 dB.

5 Conclusions & Discussion

In this report a method for protecting the privacy of audio recordings is outlined and implemented. The majority of this is done by identifying the blank areas in the recording and examining the root mean squared energy but also while considering the zero-crossing rate.

A range of results were obtained using Google's Audioset to create a subset containing different speech signals from different subjects. These signals, once properly labelled through user identified speech samples, provided a good measure of performance for the algorithm.

Using the blank sampling method and calculating the RMS values of a signal to set thresholds produced a decent degree of accuracy at an average of 87.5% accuracy. Unfortunately, the implementation of the zero-crossing rate threshold yielded no remarkable improvements in performance. While investigating the addition of Gaussian white noise to test signals, the results turned out as predicted with a lower SNR producing a lower overall average accuracy for the algorithm's ability to detect speech samples. A decent degree of accuracy was still upheld with a deteriorated base signal.

While these results are promising, further work is required for improving the accuracy of speech identification, especially when additional background noises are added to test signals. There is also a limitation on what the algorithm can identify as speech and non-speech in the instance of both types of samples being loud and possessing a high amplitude. The MATLAB code for the project has been made available on GitHub as an open-source repository and the Audioset dataset used is widely available.

Overall, the initial aim of the project has been achieved with a relatively high degree of accuracy according to the results obtained. The output signal has the speech content removed and privacy is protected.

6 References

[1] V. Petrock, “Voice Assistant Use Reaches Critical Mass,” Insider Intelligence, 15 August 2019. [Online]. Available: https://www.emarketer.com/content/voice-assistant-use-reaches-critical-mass. [Accessed 10 May 2021].

[2] P. Regulation, Regulation (EU) 2016/679 of the European Parliament and of the Council, Regulation (eu) 679 (2016), 2016.

[3] D. Istrate, J. Boudy, H. Medjahed and J. L. Baldinger, “Medical Remote Monitoring using sound environment analysis and wearable sensors,” *BioMED,* pp. 517-532, 2009.

[4] I. Ubhayaratne, M. Pereira, Y. Xiang and B. Rolfe, “Audio signal analysis for tool wear monitoring in sheet metal stamping,” *Mechanical Systems and Signal Processing,* vol. 85, no. 0888-3270, pp. 809-826, 2017.

[5] T. Bäckström, “Introduction to speech processing - Waveform,” 10 June 2019. [Online]. Available: https://wiki.aalto.fi/display/ITSP/Waveform.

[6] C. E. Shannon, “Communication in the Presence of Noise,” *Proceedings of the IRE,* vol. 37, no. 1, pp. 10-21, 1949.

[7] Dataforth, “Aliasing, Anti-aliasing,” Dataforth, [Online]. Available: https://www.dataforth.com/anti-aliasing.aspx. [Accessed 21 March 2021].

[8] Google AI Blog, “Launching The Speech Commands Dataset,” Google, [Online]. Available: https://ai.googleblog.com/2017/08/launching-speech-commands-dataset.html. [Accessed 05 Feb 2021].

[9] Creative Commons, “About the Licences,” Creative Commons, [Online]. Available: https://creativecommons.org/licenses/by/4.0/. [Accessed 05 Feb 2021].

[10] Google, “Audioset; A large-scale dataset of manually annotated audio events,” Google, [Online]. Available: https://research.google.com/audioset//index.html. [Accessed 06 April 2021].

[11] J. Gemmeke, E. Daniel, D. Freedman, A. Jansen, W. Lawrence, R. Moore, M. Plakal and M. Ritter, “Audio set: An ontology and human-labeled dataset for audio events,” IEEE ICASSP 2017, New Orleans, 2017.

[12] K. Sun, C. Chen and X. Zhang, “"Alexa, stop spying on me!": speech privacy protection against voice assistants,” in *SenSys '20: The 18th ACM Conference on Embedded Networked Sensor Systems*, Virtual Event Japan, 2020.

[13] D. Ortiz and e. al, “A simple but efficient voice activity detection algorithm through Hilbert transform and dynamic threshold for speech pathologies,” *Journal of Physics: Conference Series,* 2016.

[14] R. Bachu, S. Kopparthi, B. Adapa and B. Barkana, “Separation of Voiced and Unvoiced using Zero crossing rate and Energy of the Speech Signal,” *American Soc. for Eng. Education (ASEE) Zone Conf. Proc,* p. 1–7, 2008.

[15] C. Panagiotakis and G. Tziritas, “A speech/music discriminator based on RMS and zero-crossings,” *IEEE Transactions on Multimedia,* vol. 7, no. 1, pp. 155-156, 2005.

[16] YTMP3, “YouTube to Mp3,” YTMP3, [Online]. Available: https://ytmp3.cc/youtube-to-mp3/. [Accessed 4 March 2021].

[17] H. Traunmüller and A. Eriksson, *The frequency range of the voice fundamental in the speech of male and female adults,* Stockholm: Institutionen för lingvistik, Stockholms universitet, S-106 91, 1995.

Appendix A. audioPrivacyProtection.m

```
% Audio Privacy Protection
% Course/Year: DT021/4

% Introduction:
% The purpose of this code is to read in an audio file and
identify
% the samples that contain speech. Those samples will then
subsequently
% be removed to obfuscate the audio recording so as to render
speech
% unintelligible while preserving other salient audio features.

% Methodology:
% The code is broken up into the following sections
% 1. Read in audio signal -  the signal and relevant labels are
read
% by the program. For SNR testing the Gaussian white noise is
also
% added here
% 2. Identification of blank samples - samples above defined
threshold
% are identified using frame-by-frame analysis and written to a
vector.
% 3. Extract Root-Mean Square Energy - RMS values are calculated
and
% relevant samples noted in vector.
% 4. Extract Zero-Crossing Rate - ZCR values are calculated and
% relevant samples noted in vector.
% 5. Accuracy test & Confusion matrix - performance metrics
calculated.
% Output audio signal also initialised.
% 6. Plot Figures

% Variables/Arrays:
% x = input audio signal
% y = signal with added Gaussian white noise
% A = labels in numeric format
% blankSamplesVector = binary vector for non zero samples
identified
% rmsVector = binary vector for rms samples identified
% zcrVector = binary vector for zcr samples identified
% testVector = binary vector containing sum of previous vectors

% Required Functions:
% rmsEnergyValues.m
% zcrValues.m
% awgn / Gaussian white noise function
```

```
clc; close all; clear all;

%----------------------------------------
% Section 1. Read in audio signal
%----------------------------------------

[x,fs] = audioread('train1.mp3');
N = length(x);

% Addition of Gaussian white noise, 10 is SNR in dB
y = awgn(x,10, 'measured');

% Labels created to benchmark speech samples
fileID = fopen('train1_labels.txt','r');
formatSpec = '%f';
A = fscanf(fileID,formatSpec);

%-----------------------------------------
% Section 2. Identification of blank samples
%-----------------------------------------

% blankSamplesVector used to identify samples above threshold,
% non silent samples
blankSamplesVector = [zeros(N,1)];

% Set parameters for analysis
frame_duration = 0.1; % 0.1 of a second
frame_len = frame_duration*fs;
hopLen = frame_len/2;
num_frames = floor(N/frame_len);

% For loop to iterate through frames of input signal
% and identify silent samples
for k = 1:num_frames

    frame = x((k-1)*frame_len + 1 : frame_len*k);
    max_val = max(frame); % find max value in frame

    %frame above threshold
    if(max_val > 0.1)
        %frame values 1 in vector
        blankSamplesVector((k-1)*frame_len + 1 : frame_len*k)=1;


    %frame below threshold
    elseif(max_val <= 0.1)
        %frame values 0 in vector
        blankSamplesVector((k-1)*frame_len + 1 : frame_len*k)=0;
    end
end


%----------------------------------------
% Section 3. Extract Root-Mean Square Energy
%----------------------------------------

% Vector to store values of RMS function
rmsVector = [zeros(N,1)];

% RMS function
finalRmsValues= rmsEnergyValues(x, frame_len, hopLen);

% Count for checking >5 frames in a row below threshold
count1=0;

% For loop to iterate through frames of input signal and identify
RMS
% samples greater than 0.1
for k = 1:num_frames

    frame = x((k-1)*frame_len + 1 : frame_len*k);

    % frame above threshold
    if(finalRmsValues(k) > 0.1)
        rmsVector((k-1)*frame_len + 1 : frame_len*k)=1; %frame
values 1 in vector
        count1=0;

    % frame below threshold
    % added condition that must be 5 frames in a row to
initialise
    % zero values to vector
    elseif(finalRmsValues(k) <= 0.1)&&(count1<5)
        count1=count1+1;
    elseif(finalRmsValues(k) <= 0.1) &&(count1==5)
        rmsVector((k-6)*frame_len + 1 : frame_len*k)=0; %frame
values 0 in vector
        count1=count1+1;
    elseif(finalRmsValues(k) <= 0.1) && (count1>5)
        rmsVector((k-1)*frame_len + 1 : frame_len*k)=0; %frame
values 0 in vector
        count1=count1+1;
    end
end

%------------------------------------------
% Section 4. Extract Zero-Crossing Rate
```

```
%-------------------------------------------

% Vector to store values of ZCR function
zcrVector = [zeros(N,1)];

% Resize signal for zcr analysis
unused_samples = mod(N, frame_len);
frames = reshape( x(1:(N-unused_samples)), frame_len, []);

% ZCR function
finalZcrValues= zcrValues(x, frames, num_frames, fs);


% For loop to iterate through frames of input signal and identify
silent
% samples
for j = 1:num_frames
    frame = x((j-1)*frame_len + 1 : frame_len*j);

    %frame above threshold
    if(finalZcrValues(j) < 100)
        %frame values 1 in vector
        zcrVector((j-1)*frame_len + 1 : frame_len*j)=1;

    %frame below threshold
    elseif(finalZcrValues(j) >= 100)
        %frame values 0 in vector
        zcrVector((j-1)*frame_len + 1 : frame_len*j)=0;
    end
end

%-------------------------------------------
% Section 5. Accuracy test & Confusion matrix
%-------------------------------------------

% Formatting of labels from numeric values into binary values
% in accuracy vector using index values

% Odd index values
oddIndexVals = A(1:2:end) ;
oddIndexVals =oddIndexVals*fs;
oddIndexVals=floor(oddIndexVals);

% Make first value of labels =1
if oddIndexVals(1)==0
    oddIndexVals(1)=1;
end

% Even index values
```

```
evenIndexVals = A(2:2:end) ;
evenIndexVals = evenIndexVals*fs;
evenIndexVals=floor(evenIndexVals);

% Make labels fit to size of signal
if evenIndexVals(end)>N
    evenIndexVals(end)=N-1;
end

% Vector containing benchmark binary values for speech
% created from labels
accuracyVector = [zeros(N,1)];
k=1;

% For loop for writing to accuracyVector
for j=1:size(oddIndexVals)
     for k=oddIndexVals(j):1:evenIndexVals(j)
        accuracyVector(k)=1;
     end
end

% testVector created using AND operation
testVector = blankSamplesVector | rmsVector;
testVector=double(testVector);

% Set performance metric values =0
truePositive=0;
trueNegative=0;
falsePositive=0;
falseNegative=0;

% For loop for comparing accuracyVector and testVector
for j=1:N
    if testVector(j)== accuracyVector(j) && testVector(j)==1
        truePositive=truePositive+1;
    elseif testVector(j)== accuracyVector(j) && testVector(j)==0
        trueNegative=trueNegative+1;
    elseif testVector(j)~= accuracyVector(j) && testVector(j)==1
        falsePositive=falsePositive+1;
    elseif testVector(j)~= accuracyVector(j) && testVector(j)==0
        falseNegative=falseNegative+1;
    end
end

rmsVector=rmsVector(:,1);
accuracyVector=accuracyVector(:,1);

% Confusion matrix displayed
C=confusionmat(accuracyVector,testVector);
```

```
confusionchart(C)
title 'Audio Privacy Protection Confusion Matrix'
C.XLabel = 'Predicted Class';
C.YLabel = 'True Class';

% Perfomance metrics calculated
accuracy=((truePositive+trueNegative)/N)*100;
inaccuracy=((falsePositive+falseNegative)/N)*100;
precision=(truePositive/(truePositive+falsePositive))*100;
sensitivity=(truePositive/(truePositive+falseNegative))*100;
specificity=(trueNegative/(trueNegative+falsePositive))*100;
NPV=(trueNegative/(trueNegative+falseNegative))*100;

% Initialise new audio signal with privacy protected using
testVector
new_sig = zeros(N,1);
for j=1:N
    if testVector(j)==0
        new_sig(j) = x(j) ;
    elseif testVector(j)==1
         new_sig(j)=0;
    end
end

%------------------------------------------
% Section 6. Plot Figures
%------------------------------------------

subplot(5,1,1)
x = x(:,1);
plot(x)
axis tight
xlabel 'Samples'
ylabel 'Amplitude'
title 'Discrete-time signal'

subplot(5,1,2)
plot(accuracyVector)
axis tight
title 'accuracyVector user identified speech samples'

subplot(5,1,3)
plot(testVector)
axis tight
title 'algorithm identified speech samples'

subplot(5,1,4)
plot(finalRmsValues)
axis tight
```

```
xlabel 'Frames'
ylabel 'RMS'
title 'Root-Mean Square Energy'

subplot(5,1,5)
plot(finalZcrValues)
axis tight
xlabel 'Frames'
ylabel 'ZCR'
title 'Zero-Crossing Rate'
```

Appendix B. rmsEnergyValues.m

```
% Audio Privacy Protection
% Course/Year: DT021/4

% rmsEnergyValues.m
% Function for finding RMS energy values in signal
% using frame-by-frame analysis

function outputRmsValues = rmsEnergyValues(signal, frame_lgt,
hop_length)

    rmsArray = [];

    for i = 1:frame_lgt:length(signal)
        % RMS equation to find RMS value in frame
        rms_current_frame =
sqrt(sum(signal(i:i+frame_lgt).^2)/frame_lgt);
        rmsArray=[rmsArray, rms_current_frame]; % value stored in
array
    end

    outputRmsValues = rmsArray;
end
```

Appendix C. zcrValues.m

```
% Audio Privacy Protection
% Course/Year: DT021/4

% zcrValues.m
% Function for finding zero-crossing values in signal
% using frame-by-frame analysis

function outputZcrValues = zcrValues(input_audio, frames,
n_frames, Fs)

    % take mean of frame away from values in frame
    for i=1:n_frames
        frames(:,i)=frames(:,i)-mean(frames(:,i));
    end

    % find sum of values <0, will give ZCR
    zcr = sum(frames(1:end-1, :).*frames(2:end, :)<0);
    outputZcrValues=zcr;
end
```