Adaptive content-based sound compression

Marcus Tedenvall Markus Gerard

June, 2019

Master's thesis work in Electrical Measurements carried out at
Axis Communications AB.

Supervisors: Mersad Jelacic, Martin Stridh
Examiner: Leif Sörnmo

LUND UNIVERSITY

LTH
FACULTY OF ENGINEERING

Department of Biomedical Engineering

Abstract

Three different classification solutions for distinguishing events from non-events in surveillance audio are described and evaluated. The three compared methods are energy functions, Gaussian mixture modelling algorithms (GMM) and existing voice activity detectors (VAD). Recorded test signals with corresponding manually labelled ground truth files are used to determine the accuracies of the methods. The GMM algorithm performed best with an accuracy of 86.63 % in average over different environments, amount of activity and amount of noise. This can be compared to the accuracy of the energy functions (60.08 %) and the VADs (77.85 %). With this method, the data size is reduced from 57.87 to 25.83 MB per hour on average when using an instant bitrate decrease for non-event segments, compared to constantly recording the audio with a high bitrate. This is a reduction in storage size of 55.37 % for the test files which contain 39.9 % activity. A technology we call gradual bitrate decline is also implemented which reduces the bitrate slowly over time instead of instantly after an event has happened. The technology improves the listening experience with the trade-off of taking up more disk space.

The noise level at which the GMM algorithm has the best results is with a signal-to-noise ratio (SNR) around 20 dB, where over 97 % of the events are classified as events, and with an accuracy above 92 % for the measured test file. For lower SNR, fewer events are found where for example only 22 % of the events are found at an SNR of 0 dB. The algorithm does not work with outdoor noises such as wind and it is instead constructed and optimised for indoor use.

Acknowledgements

We would like to thank Martin Stridh and Leif Sörnmo from the department of Biomedical Engineering at LTH for their work as supervisor and examiner respectively. We would also like to thank our supervisor at Axis, Mersad Jelacic for all his help, dedication and assistance throughout the thesis. We are also grateful to the other people from Axis Product Audio team that have also helped with valuable input and knowledge. Lastly we would like to thank for the opportunity to carry out the master's thesis for Axis Communications.

Contents

Chapter 1

Introduction

In this chapter we define the problem which we attempt to solve and what contributions to science it may bring. The idea and motivation behind the problem is discussed in a brief background.

1.1 Background

With the increased amount of information in technology and with the digitalisation and IoT that is going on worldwide, the amount of generated data is increasing rapidly. An estimation is that the total amount of data stored as of 2018 is around 33 zetabytes. This is an incredibly large number but a prediction is that the amount of data stored worldwide in 2025 will be 175 ZB. This represents an annual growth rate of 61 % [23]. In order to combat the increased storage requirements, smart solutions for data that requires lots of storage is needed.

Continuous recording of audio and video in surveillance systems generates a lot of data which in turn consumes excessive storage and bandwidth. These factors are costs for the end user so there is an interest to keep them at a minimum. There are several other reasons for reducing the size of recorded data such as being able to expand the surveillance system with more cameras, record in higher quality or store the recordings for a longer time period. This need to reduce the amount of data has driven the development in video compression in surveillance systems and one technique is to compress video with a variable bit rate that depends on the content of the video.

Video recordings may contain long static sequences and during these sequences

there is no need to transmit a new image until a change in the scene occurs. The bitrate for video during these sequences could potentially reach zero which surprisingly has led to audio taking up a bigger portion than video in stored data. A similar approach as to utilising static sequences in video can be used for audio. Certain uninteresting sound events, such as sequences containing only silence or noise, can be heavily compressed since the desired quality for these events is usually low. This will drastically reduce the amount of stored data since lack of interesting sound events is typical for surveillance recordings.

1.2 Problem

The purpose of this thesis is to investigate possibilities to reduce the amount of data generated by audio in surveillance recordings and to develop a program for this task. This is to be done without sacrificing important parts of the recordings and also while keeping the audio quality acceptable. Uninteresting audio like silence or noise should be identified and heavily compressed without much consideration in the resulting quality. Interesting audio such as speech should be compressed with a high bitrate to maintain acceptable quality. The customer should also not be needing knowledge in audio, whereby the setup should be as simple as possible.

1.3 Scientific Contributions

This work can be used to further research how audio streams can be handled to reduce the size of transmitted and stored data, especially for surveillance systems. The proposed method to identify interesting audio can also be used in other systems where there is a need to monitor activity in audio but its data size is of less concern. This work will tell how well the method performs on general audio and this can be used as a guideline for research with similar goals.

1.4 Distribution of Responsibility

During the thesis, the work have been divided equally between the both authors.

1.5 Outline

This thesis is outlined as follows: A theoretical background to audio and audio compression is given in Chapter 2, along with a summary of previous research done in the field. Chapter 3 gives a short overview of the system and describes the used tools. The

methodology of the thesis is described in Chapter 4. In Chapter 5, the results are presented. The thesis is concluded in Chapter 6 where the results are further discussed and the thesis is evaluated.

Chapter 2

Theoretical Background

This chapter is intended to give the reader a short theory introduction to the subject, and previous work in the field is explained and summarised. First, a brief introduction to audio is given to make it easier to follow and understand the rest of the report. Data compression and audio codecs are then explained followed by an introduction to voice activity detectors. Finally, an overview of related work of the thesis is provided.

2.1 Theory

This section is intended to give the theoretical background of the thesis. This includes basics in audio, data compression, audio codecs and voice activity detectors.

2.1.1 Basics in audio

Sounds are caused by variations in air pressure. Pressure-changes in other media also produce sounds but most sounds we experience propagate through air. The loudness of a sound is determined by how much the air is pressured and the frequency is determined by how often the pressure increases and decreases [29]. These changes in pressure are usually represented as a sound wave which can be seen in Figure 2.1. Loudness is often referred to as the amplitude of a sound wave. There are many units to express the amplitude of sound and a common one is decibel (dB) relative to the human hearing threshold. How many times the sound wave repeats itself during a certain time unit is known as the frequency, often measured in hertz (Hz) which is the number of repetitions

per second. Two or more signals with different frequencies can be combined into one signal as can be seen in Figure 2.2. It is typical for most sounds we hear to contain many different frequency components.

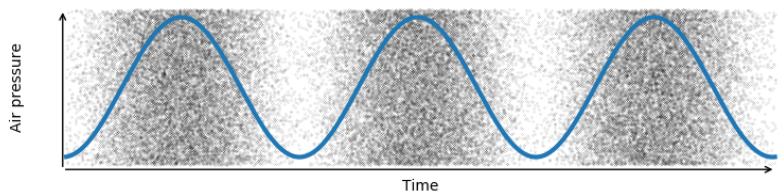


Figure 2.1: Variation in air pressure over time. The small dots represents air molecules and the line shows how much they compress and decompress over time which results in a sound wave.

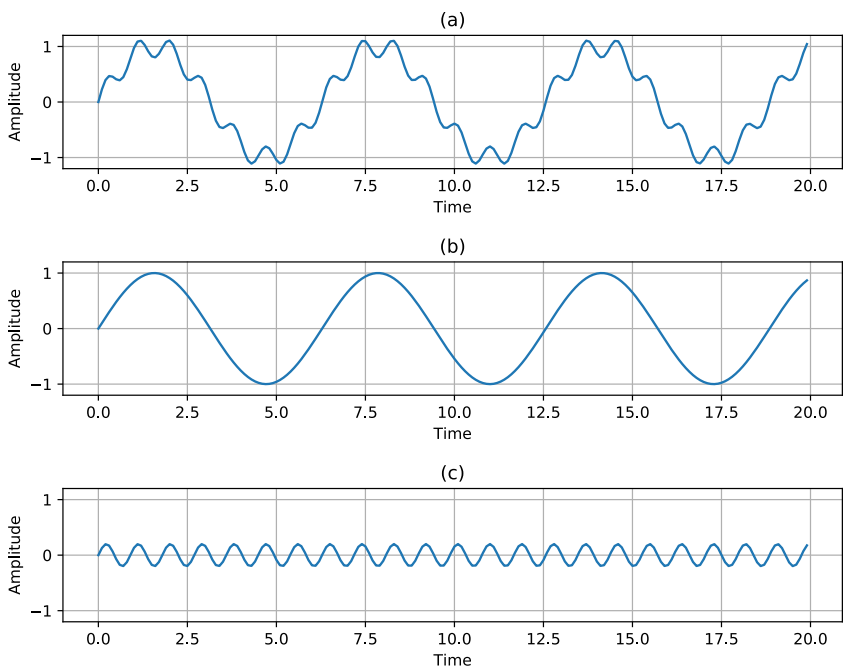


Figure 2.2: The signal (a) is a composition of the signals (b) and (c), which are sine waves with different amplitudes and frequencies.

Analog-to-digital conversion

One way to transmit, save, analyse or manipulate sound is to convert it into digital sound. When sound is recorded, the analog signals are transformed into digital signals by sampling, quantization and coding.

Sampling is done by periodically taking an amplitude value of the signal. How often sampling is done is called the sampling frequency or the sampling rate. To be able to fully reconstruct the signal later on, when it is transformed back to an analog signal, the sampling frequency has to follow the Nyquist sampling theorem. The Nyquist sampling theorem says that the sampling frequency has to be more than twice as high as the highest frequency in the signal [21]. If the sampling frequency is lower than this, aliasing will occur, which is visualised in Figures 2.3 and 2.4. A too low sampling frequency will result in a different signal to the original one during reconstruction, see Figure 2.5. An anti-aliasing filter can be used to prevent aliasing. The filter removes frequency components that are higher than half the sampling frequency before sampling.

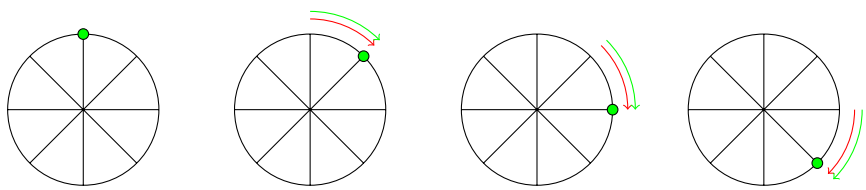


Figure 2.3: Symbolisation of correctly sampled images of a spinning wheel. Since the sampling frequency of the images in this case is eight times higher than the wheel's frequency, the red arrows symbolising the actual movement and the green arrows symbolising the perceived movement are the same.

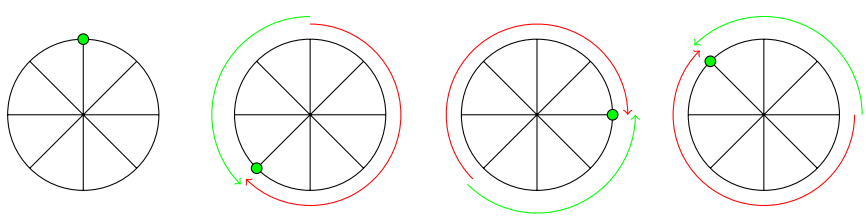


Figure 2.4: Symbolisation of incorrectly sampled images of a spinning wheel. In this case, the sampling frequency is lower than twice of that of the spinning wheel. This causes the red arrows symbolising the actual movement to differ from the perceived movement represented by the green arrows. This effect is called aliasing.

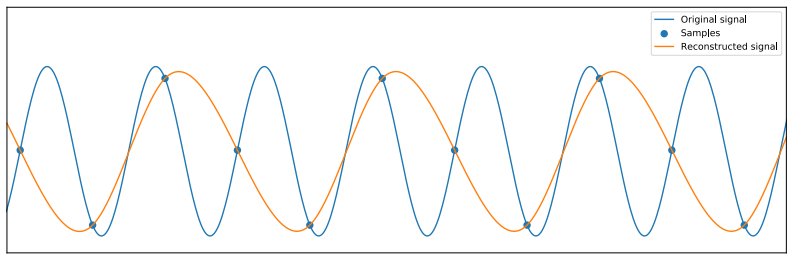


Figure 2.5: An audio signal with frequency 12 kHz sampled with sampling frequency 18 kHz. The sampling frequency does not follow the Nyquist sampling theorem which makes the reconstructed signal into a completely different signal.

Humans can hear frequencies up to roughly 20 kHz so a good strategy when recording audio is to record with a sampling frequency greater than 40 kHz [11]. A common sampling frequency has been 44.1 kHz for quite some time, which is used in the CD format. Nowadays, the standard has shifted to 48 kHz when recording professional audio [33]. The sampling frequency depends on the use case which means that many applications uses a lower sampling frequency than 48 kHz. Since human speech is typically below 4 kHz, a lower sampling frequency can be used when recording speech than when recording music [12]. Common sampling frequencies of communication applications like voice over IP are 8 and 16 kHz [27].

After sampling, the signal is quantized. The samples are either rounded or truncated to the nearest value in a finite set of discrete values, see Figure 2.6. This set is determined by how many bits that are used to represent the signal, also known as the bit depth. If 16 bits are used, which is used in the CD format, the samples can take $2^{16} = 65536$ possible values, so called quantization levels. The difference between the quantized values and their original values introduces an error that is called quantization error or quantization noise. The magnitude of this error depends on the number of bits used, with fewer bits used resulting in a greater error. This means that even though the Nyquist sampling theorem is fullfilled there is still a loss of information in the analog-to-digital conversion. Finally, the quantized values are coded into binary sequences. The digital signal can be converted back to an analog signal by interpolating these quantized and coded values.

Both sampling and quantization provides a trade-off between quality and data size of the recorded audio. Fewer samples per second reduces the amount of data needed to represent the signal but it comes with the cost of potentially losing valuable frequencies. Similarly, fewer quantization levels reduce data size but result in worse quality due to the quantization error. Also, the analog-to-digital conversion device usually becomes more expensive with higher sampling frequency and larger bit depth [21].

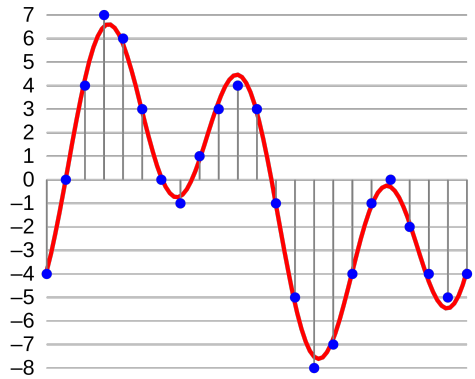


Figure 2.6: An analog-to-digital conversion scheme where audio samples are mapped to a four bit value, from Wikipedia [32].

Bitrate

A common term used in audio coding is bitrate. This refers to the number of bits that are processed or transferred per unit of time. The unit used to quantify bitrate is bits per second (bit/s). For audio, the bitrate for a given audio recording is determined by the sampling frequency as well as the bit depth and the amount of channels according to Equation 2.1.

$$Bitrate = Sampling\ rate * Bit\ depth * Channels \tag{2.1}$$

Audio channels

Audio channels are audio signal communication channels which transfers the audio signals and are often used in amongst others sound reinforcement. By recording sound with multiple microphones it is possible to play the sound using multiple channels and thereby gain stereophonic sound, often called stereo. Simple recording and playback devices usually have a single channel, often called monophonic or mono sound. More advanced devices have stereo sound, like headphones typically using two channels or the 5.1 surround sound systems which uses six channels.

Fourier transform

The Fourier transform is a way to convert a signal in the time domain to the frequency domain. The transform makes it possible to analyse the present frequencies in a signal, for example the individual frequencies of a musical chord in an audio signal. The result of a Fourier transform is a complex value, with the absolute value describing the energy

power in different frequency components. There is also an inverse Fourier transform for converting frequency components back into the time domain.

If the Fourier transform is calculated for a time series with equal time-steps, the Fourier transform is known as a discrete Fourier transform (DFT). The discrete Fourier transform is however computational heavy and often too slow to be practical. Here, the fast Fourier transform (FFT) shines with being much faster to compute and is therefore commonly used in practice. The complexity is $O(N^2)$ for the DFT and $O(NlogN)$ for the FFT. In numbers that would mean that a series with 1000 sample points would for the DFT require $1000^2 = 1000000$ operations while the FFT only would require $1000 * log_2 1000 \approx 10000$ operations which in this case would make the FFT a hundred times faster. [36]

Signal-to-noise ratio

Signal-to-noise ratio (SNR) is a measurement that indicates how the signal strength compares to that of noise. The formula for calculating SNR is shown in Equation 2.2

$$SNR = \frac{P_s}{P_N} \tag{2.2}$$

where P_s denotes the power of the signal, and P_N the power of the noise.

Often however, the SNR is instead written in a logarithmic scale which can be seen in Equation 2.3.

$$SNR(dB) = 10 \log_{10} \left(\frac{P_s}{P_N} \right) \tag{2.3}$$

When measuring SNR in audio, the measurement is often done on the whole audio system. A sine wave with a known amplitude is then sent into the system. The power of the signal could then be compared to that of the noise and a measurement for the system's SNR is received.

2.1.2 Data compression

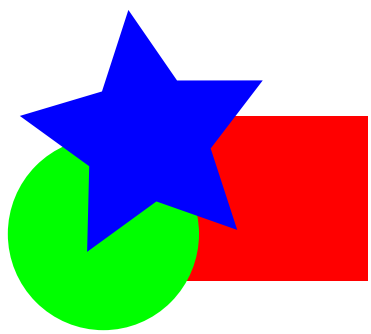
The ability to compress data has been of interest for a long time. An early way of compressing data is the Morse code, created 1837 [7]. It encoded the most common letters in the English alphabet as 'e' and 't' with shorter codes resulting in compressed coding. Later, in the late 1940's when mainframe computers started to take hold, Claude Shannon was the first to publish scientific papers on the subject which essentially created the area of study [35].

Compression is today widely used in order to encode the information with fewer bits than the original representation. This is mainly of interest when sending, receiving and storing data. One way to do this is by using lossless compression. This method is the oldest and is based on mathematical operations to find statistical redundancy in

the data. The other option is lossy compression, where information is lost in favour of a more efficient compression. As an example could 650 MB of disc storage hold one hour of high fidelty uncompressed music, two hours of losslessly compressed music, or seven hours of lossy compressed music in MP3 format at medium bitrate [35].

Lossless compression

Since the algorithms and codecs used in this project uses lossy compression, lossless compression is only explained briefly. As mentioned above, lossless compression utilises statistical redundancy in order to compress numbers. The two primary lossless encoding algorithms are Huffman coding and arithmetic coding. The idea is generally that a sequence of 1 red pixel, 1 red pixel, 1 red pixel etc, could be compressed to say e.g. 200 red pixels instead. For simple images with big one coloured segments such as Figure 2.7a, the compression is rather good. For complex real life nature pictures like Figure 2.7b, the compression is often not very effective due to the complexity of nature. This kind of compression is primarily used when no data can be lost, e.g. in source code or text documents [35].



(a) A simple image which most likely could be compressed well using a lossless compression algorithm.

(b) A more complex image which most likely would not be compressed well using a lossless compression algorithm.

Figure 2.7: Images that would most likely be compressed well and badly respectively using a lossless compression algorithm.

Lossy compression

With the increased use of digital cameras and images in the early 1990's, lossy compression became more common. This way of compressing data is not perfect, meaning that the original copy cannot be resolved from a lossy compressed one. Because of this, when compressing using lossy compression methods, the user would have to decide how much the data should be compressed in relation to the accuracy of the result. The more

the data is compressed, the lower the quality of the resulting image or audio recording will be. This trade-off is usually decided depending on the situation and usage [35].

Most of the time, the data is sampled with more precision than a human can perceive. Both the ear and the eye have physical limitations, for example a human can only hear frequencies between 20 Hz and 20 kHz [11]. It is in this case not vital to store sound information outside of this range. Many times, data in forms of images and audio is however compressed further than the perception limit. This is in order to reduce storage or bandwidth usage e.g. in JPEG images and phone calls. Video can in most cases be lossy compressed with a 100:1 ratio, and audio have a 10:1 ratio without losing much quality [37].

There are many different codecs that compresses audio and video with different goals. The MPEG-2 video coding format is an example of a lossy compression algorithm used in DVDs. MP3 is another example of a codec, but that is used in audio instead [35]. Another codec used when compressing audio is the open source codec Opus. Compared to other audio codecs it has a good quality-to-bitrate ratio and is relatively fast [6]. The Opus codec is used in this project and will therefore be evaluated and explained further in Section 2.1.3 about audio codecs.

2.1.3 Audio codec

An audio codec is an application for encoding and decoding audio signals, which is a way of converting signals between different formats or codes. A common purpose of encoding is to compress digital signals to reduce data size for transmission or storage. The decoder then decompresses the audio signal and tries to recover the original signal.

Opus Interactive Audio Codec

Opus, released in 2012, is a rather young audio codec compared to common used codecs such as AAC and MP3. It is developed by programmers mainly associated with Xiph.Org and Skype. The codec is based on previous compression techniques from two earlier audio codecs, SILK by Skype and CELT by Xiph.Org. SILK is based on linear prediction and CELT is based on the modified discrete cosine transform. These techniques constitutes two of the three modes of Opus, the third one being a hybrid of the two. The different modes suits differently well depending on audio content and this is why Opus has integrated these earlier codecs, along with some modifications, into one codec to be able to switch between them seamlessly. [38]

Opus provides a wide range of use cases. It is primarily designed for speech and music but it also applicates well to general audio. Some of its main features are [28]:

- Support for variable bitrate and constant bitrate.
- Encoding in bitrates from 6 kbit/s to 510 kbit/s, in increments of 0.4 kbit/s.
- Support for input with sample rates 8, 12, 16, 24 and 48 kHz.

- Support for frame sizes 2.5, 5, 10, 20, 40 and 60 ms.
- Support for both mono and stereo audio.
- Allows input and output of audio bandwidths narrowband, mediumband, wideband, super-wideband and fullband.
- Packet loss concealment.

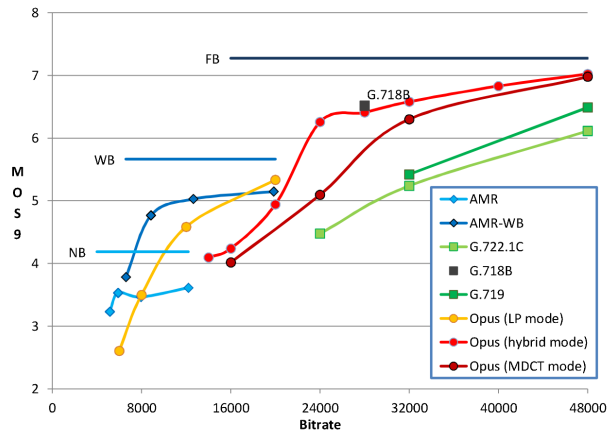
One key feature of Opus is the ability to encode in variable bitrate (VBR). Opus analyses the input and increases or decreases the bitrate depending on what content the audio withholds. When VBR is activated, which it is by default, the different bitrate settings (6 kbit/s to 510 kbit/s) is an approximation of the average bitrate that the encoding will produce, even though the actual bitrate could be higher or lower than this average [38].

In [22], Opus is compared to the following audio codecs: AMR, AMR-WB, G.718B, G.722.1C and G.719. They are compared by quality when coding in bitrates from 6 kbit/s to 48 kbit/s. Voice quality in clean speech and noisy speech is measured with a mean opinion score (MOS) where listeners rate the quality from very bad to excellent in a nine step scale. For noisy speech, the resulting quality is similar between the codecs, see Figure 2.8b. For clean speech Opus provide better quality than G.722.1C and G.719 for all bitrates, see Figure 2.8a. It is also better than AMR for bitrates over 8 kbit/s but worse for lower bitrates. Opus has a marginally worse quality than AMR-WB for bitrates between 6-16 kbit/s but better for bitrates 16-20 kbit/s. G.718B is slightly better than Opus but it is only tested at the bitrate 28 kbit/s. Opus has the ability to code in the entire range 6-48 kbit/s (actually all the way up to 510 kbit/s, but is not tested in the article) whereas the other codecs are optimized to code in a smaller range of bitrates. Opus produces rather decent quality compared to the alternatives, especially when taking its bitrate flexibility into consideration.

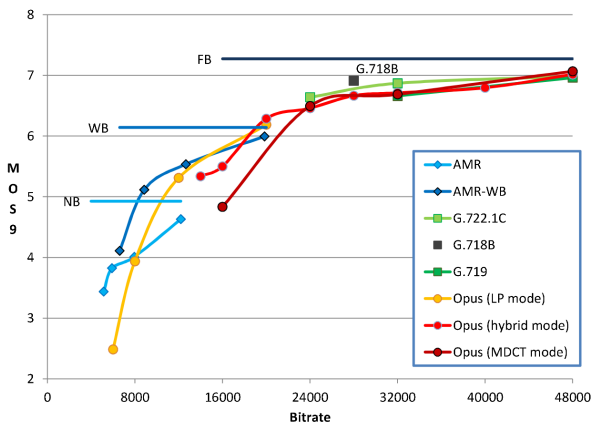
Variable bitrate

Variable bitrate is a technique where the bitrate of a data stream varies over time. The idea is to identify appropriate quality for a coded segment in a data stream and adjust the bitrate thereafter. If an audio segment only contains noise, then the quality of the result is less important and the bitrate can be reduced. If the segment instead contains speech or music, then high quality is important and in order to preserve it, the bitrate can be increased. This means that VBR gives a better quality-to-bitrate ratio than what constant bitrate (CBR) does. A codec that uses CBR will encode a data stream with a fixed bitrate independent of content.

Using VBR over CBR has some advantages and disadvantages. The main advantage is the aforementioned quality-to-bitrate ratio which allows for less transmission and storage. One disadvantage is that the encoding process may take longer time since it is more complex.



(a) Voice quality in clean speech.



(b) Voice quality in noisy speech.

Figure 2.8: Comparison of quality for different bitrates and codecs, adopted from [22].

2.1.4 Voice activity detection

One area where it is interesting to detect audio activity is voiced telecommunication. Most of the time in a conversation there is only one person talking which gives an opportunity to reduce transmission costs by suppressing silence caused by the listeners. This could, roughly speaking, half the transmitted data in a two way conversation, assuming that there always is one person talking and one person listening. This has motivated research and development of voice activity detectors (VAD). Such detectors can be useful in audio surveillance as well, even though interesting audio in surveillance is often much more than just speech. Two VADs that are tested in this project are the G.729 VAD and the WebRTC VAD [16, 30].

G.729

G.729 is an audio codec with a feature for suppressing silence. This is done by detecting voice activity with a VAD, then using discontinuous transmission to only transmit data when there is activity. The gaps caused by silence is filled in by a comfort noise generator in the decoder. Comfort noise is added to make the conversation feel natural and to assure users that the connection is not lost during silent parts. The VAD uses the following four audio features when deciding if there is voice activity or not: [16]

- **Line spectral frequencies** A set of linear prediction coefficients.
- **Full-band energy** The energy over the complete frequency range.
- **Low-band energy** The energy on frequencies 0 to F_l.
- **Zero-crossing rate** The normalized zero-crossing rate for each frame.

WebRTC

WebRTC is a framework for real time communication in the web browser, both for video and audio. It has a VAD function for detecting speech in audio signals. The VAD has a setting with four different levels of sensitivity, which sets how conservative the VAD is when classifying audio as speech. It accepts input with sample rates 8, 16, 32 and 48 kHz and frame sizes of 10, 20 and 30 ms. The classification is done by calculating probabilities for speech and background noise by using Gaussian mixture models and performing hypothesis-tests to decide the most probable class. The audio characteristic it uses for the probability model is logarithmic energy in frequency subbands. [31]

2.2 Related Work

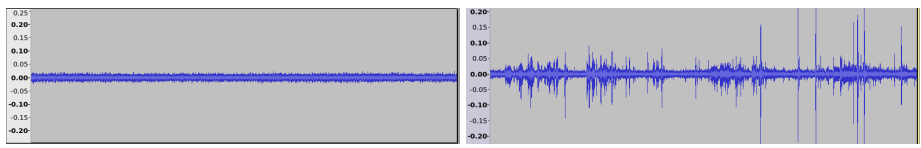
This section exemplifies previous research done on the subject and how similar problems have been solved by others in the past. A few different approaches are described in the sections below.

2.2.1 Energy functions

There are several different ways of using the energy of the sound in order to detect events [2, 3, 13, 15, 24]. The signal is divided into frames, and the energy of each frame is calculated where the peaks of energy correspond to a sound event. One of the common ways of calculating the energy E in a frame j with length N containing samples x is shown in Equation 2.4. [24]

$$E_j = \frac{1}{N} \sum_{i=(j-1)N+1}^{jN} x_i^2 \tag{2.4}$$

Looking at the mean squared signal value gives an indication of when an event has occurred. At silence, there is only noise present, while a sound event in general has a larger amplitude, see Figure 2.9. The frame with a larger amplitude will according to Equation 2.4 also have a larger energy value. A reason to use energy instead of just looking at the amplitude is that a spike of noise will not necessarily increase the energy above the threshold since the energy takes in consideration a mean of the signal amplitude over a period of time. Because of this, using energy instead of amplitude will also most likely enable the recognition of a low intensity long lasting sound event.



(a) A sound recording without sound events. **(b)** A sound recording with speech activity.

Figure 2.9: Silence and speech visualised in their waveforms.

There exist several different techniques for detecting silence in audio which uses energy as a main feature [3]. The techniques use energy features in slightly different ways and the thresholds of what is classified as an event or not is also determined differently. One technique is the basic linear energy-based detector (LED). This method uses an adaptive threshold in order to update the energy. For this technique, the threshold is continuously updated according to the current energy level and a preset learning rate which can adjust the adaptation speed. Another technique is the adaptive linear energy-based detector (ALED) which also uses an adaptive threshold, but the learning rate here is instead decided by looking at changes in energy variance. Weak fricatives detector (WFD) is also a silence detection technique which uses the characteristics of noise with the zero-crossing rate along with energy to detect events. How zero-crossing rate works is described in Section 2.2.2. The linear subband energy detector (LSED) is another energy activity detecion technique in the frequency domain where the signal is divided and analysed in subbands. The spectral flatness detector (SDF) technique guarantees that frames with a low signal-to-noise ratio are saved by doing a comparison of the variance of a speech and a noise frame. Another silence detector is the comprehensive VAD (CVAD) which firstly computes the subband energy as mentioned above. For those frames where energy fail to detect any event, zero-crossing rate and variance evaluation is done to catch any missed events. [3]

It is possible to instead use features of short time energy and low short-time energy ratio amongst others to classify sound events. The short time energy is calculated similarly to Equation 2.4. Low short-time energy ratio is a ratio of the number of frames whose short time energy are less than a predetermined fraction of average short time energy in the given time window. This is an effective feature for detecting speech. In particular because there in general are more silence frames in speech, whereby the ratio

value for speech will be much larger than that of non-speech [13]. The way these features are used is in neural networks, hidden Markov models and support vector machines which are discussed in Section 2.2.6.

Signal energy can also be used in other ways. One technique which utilises it calculates energy for each time frame and the minimum and maximum energy amount is used to create an adaptive energy threshold to determine voice and non-voice. By adjusting the threshold according to the minimum and maximum energy it is possible to create a scaling factor independent and resistant to the variable background environment. [24]

Short time energy can be calculated in other ways. In one method, the signal is divided into frequency subbands as the LSED from above, and energy is calculated and analysed on each subband separately. The reason behind this is because audio activity is not only generated in the time domain, but also with shifts in frequency. In order to overcome changes in background sound in different environments an adaptive normalised "activity value" is also introduced. This value is continuously updated to represent the current background sound. [15]

2.2.2 Zero-crossing rate

Another common feature used in audio analysis is the zero-crossing rate (ZCR). This corresponds to the number of zero-crossings (crossings of the x-axis) done per frame which is demonstrated in Figure 2.10. ZCR says something about the frequency distribution of the signal in relation to energy which do not take frequency into account. The amount of zero-crossings for a pure sine wave would for example be twice of that of the signal frequency. The ZCR does however only give a rough estimate of the frequencies due to the fact that an audio signal usually is a complex combination of different frequencies. The idea is though that a harmonic signal like voice would have lower ZCR because of the low frequency components, while noise for example typically has higher frequencies and thereby a higher ZCR [25]. Human voice usually have five to fifteen zero-crossings per 10 ms of recording [3].

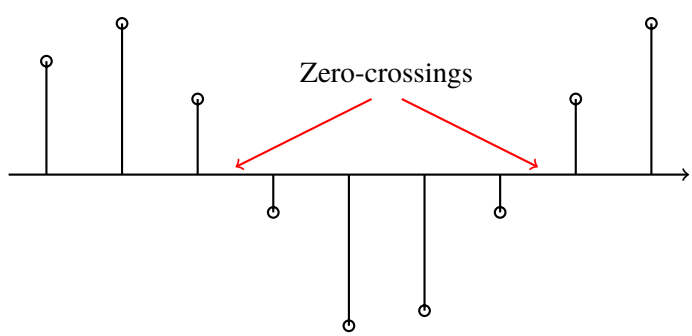


Figure 2.10: Demonstration of zero-crossing rate. The two zero-crossings are marked in the image.

ZCR is often used in conjunction with other more complex features. As stated in Section 2.2.1 about energy functions, energy and ZCR are often combined when detecting audio events. To use energy functions together with ZCR is effective in separating voiced from unvoiced speech. A general formula for calculating ZCR is shown in Equation 2.5

$$Z_n = \sum_{m=-\infty}^{\infty} |sgn[x(m)] - sgn[x(m-1)]|w(n-m) \tag{2.5}$$

where

$$sgn[x(n)] = \begin{cases} 1 & \text{if } x(n) \geq 0 \\ -1 & \text{if } x(n) < 0 \end{cases}$$

and

$$w(n) = \begin{cases} \frac{1}{2N} & \text{for } 0 \leq n \leq N-1 \\ 0 & \text{otherwise.} \end{cases}$$

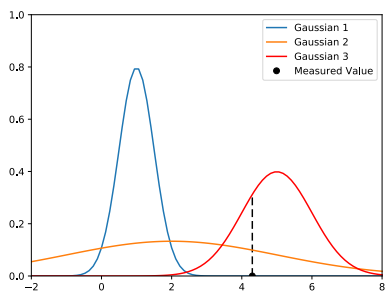
Z_n is the ZCR for frame n, $x(m)$ is the sampled signal at time m and N is the amount of samples in a frame. [9]

2.2.3 Gaussian mixture model

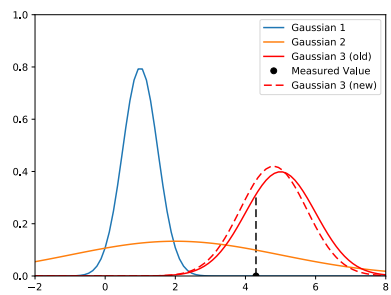
One approach to detect sound activity is to create a model for the background and use that model to classify sound as either background or foreground. This can be done by using a Gaussian mixture model (GMM). GMM is a model that consists of multiple Gaussian distributions with different means and variances. One example of modelling background with a GMM is the Stauffer-Grimson algorithm [26]. The algorithm is used to detect foreground events in video by comparing pixel values to a number of weighted Gaussian distributions. To be able to adaptively model the background, the Gaussians are updated over time with adjusted means, standard deviations and weights. For each incoming frame, the pixel values are checked if they belong to any of the Gaussians with a high enough probability. If a pixel matches any of the Gaussians, the mean and standard deviation of that distribution are adjusted to make a better representation of the measured value, see Figure 2.11. The weight of that Gaussian is increased while the other Gaussians have their weights lowered. The matched Gaussian is examined and if that Gaussian belongs to the distributions that represent the background, the pixel is classified as such and otherwise it is classified as foreground. If the measured value does not match any of the Gaussians, the Gaussian with the lowest probability is replaced with a new that has a mean equal to the measured value, see Figure 2.12. When there is no matched Gaussian the pixel value is classified as foreground since it deviates too much from the background model.

Besides the usage for video applications, the Stauffer-Grimson algorithm can also be used to adaptively model the backgroud in audio [8]. The feature for modelling the background is the power spectral density of the signal, instead of pixel values as in the

video case. The signal is divided into different frequency bands or subbands and each subband has its own GMM, independent from the others. In this way, the audio is not analysed only using energy but also by frequency components. This means that sound events that have similar energy as background noise can be detected since it will show unusual behaviour in the frequency domain. The method contains two problems which can be solved by revisiting the updating formulae and the threshold comparison [18]. The problems and solutions are further explained in Section 4.1.3.



(a) A measured value is within the limits of the third Gaussian, i.e. a match is found.



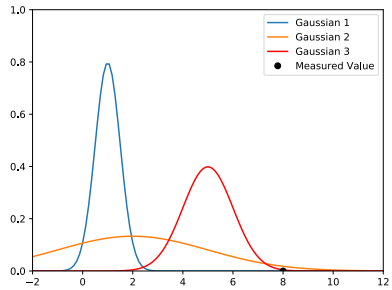
(b) The matched Gaussian gets updated values for mean and standard deviation to fit the measured value better. The third Gaussian gets its weight increased while the others get their weights lowered.

Figure 2.11: Demonstration of GMM with a measured value within the limits of one Gaussian.

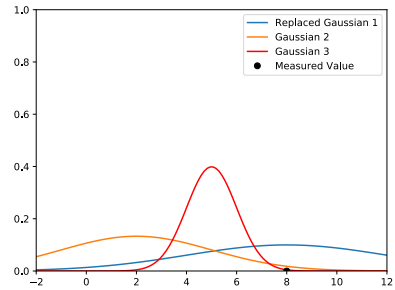
2.2.4 K-means clustering

There are several ways to handle the initialisation of the distributions used in for example the Gaussian mixture model method explained in Section 2.2.3. One way to do this is by using the K-means clustering algorithm, where the sample points are clustered to a distribution. This can be done in slightly different ways, but the most common is described below.

Firstly, initial cluster centers have to be selected. This is a crucial step as clusters might not be as optimised with bad initial cluster center placement as with careful and clever placement. The next step is to classify all sample points to their closest (often Euclidian distance) cluster center. With this new clustering, the cluster centers are shifted to be the mean of the sample points belonging to that class. The sample points are classified again, and the cluster centers updated. If any of the centers position is shifted, the sample points are once again classified, and the loop goes on. [1]



(a) The measured value is not inside the limits of any of the Gaussians. The one with the lowest probability is here the first Gaussian.



(b) The first Gaussian (it has the lowest probability) is replaced in favour of a new Gaussian with mean equals to the measured value and a with high standard deviation and low weight.

Figure 2.12: Demonstration of GMM with a measured value outside the limits of all Gaussians.

Cluster center initialisation

Since the initialisation is so vital to the performance, multiple ways of initialising the cluster centers have been proposed. One way is to randomly place a center at one of the sample points. In the next step, for each of the sample points, the shortest distance to any center is calculated. The sample value with the longest 'shortest distance' is used as a center for the next distribution. This goes on until all centers are assigned. [14]

Another common approach is similar to the one mentioned, but the measured 'shortest distance' to a cluster center instead results in probabilities. The sample points with longer 'shortest distance' to a cluster center are then more likely of being decided as a new cluster center. This is called *K-means++*. [1]

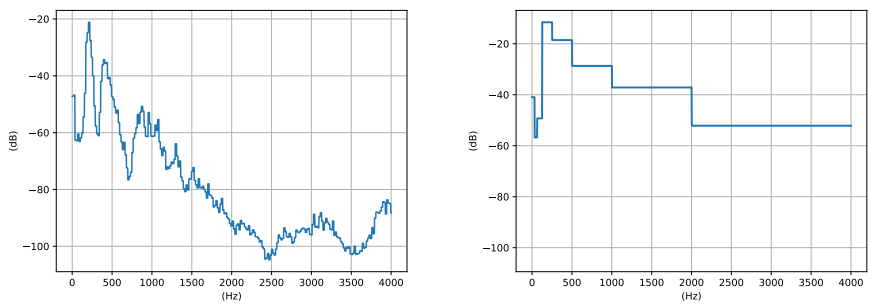
2.2.5 Power spectral density

The power spectral density (PSD) tells how power is distributed among frequency components of the signal. There are different ways of estimating the PSD, many of them use the Fourier transform and one such approach is Bartlett's method [34]. The first step in Bartlett's method is to divide the signal into smaller segments or time frames containing n samples. Then the DFT (see Section 2.1.1) is computed for each segment which results in a sequence of complex values. The power for each frequency f is computed by squaring the magnitude of the corresponding complex value as shown in Equation 2.6, where X_f is the DFT for f.

$$P(f) = |X_f|^2 = \sqrt{Re(X_f)^2 + Im(X_f)^2}^2 = Re(X_f)^2 + Im(X_f)^2 \tag{2.6}$$

The power is divided by the number of samples in the segment. Finally an average over all segments is computed for each frequency, see Figure 2.13a. When dealing with non-finite signals, such as a continuous surveillance recording, the last step is omitted and the PSD for each segment is used instead of the PSD over the entire signal.

In [8], the power for all frequencies in a particular subband are accumulated which gives a PSD over frequency bands instead of individual frequencies. The subbands are logarithmically spaced where each subband is twice as wide as the former one. The distribution of power over the subbands in an example speech signal can be seen in Figure 2.13b.



(a) PSD over frequencies.

(b) PSD over subbands.

Figure 2.13: Power distribution over frequencies and logarithmic spaced subbands in a speech signal with sample rate 8 kHz.

2.2.6 Multimodal background subtraction, hidden Markov model and support vector machines

Except for the GMM, there are a few other multimodal background subtraction methods which are commonly used to model the background and classifying events. Multimodal background subtraction is a more complicated method than simple, single-valued features like signal energy and zero-crossing. In some cases, such simple features might not be as suited for background/foreground discrimination as the multimodal techniques. [9]

GMM is a multimodal classificiation method explained in Section 2.2.3. What is lacking from the GMM is the detection of slowly varying foreground. Since the GMM will continuously update what is considered background, the gradual change will not be detected [9]. In order to solve this issue, a solution is to use supervised or semi-supervised models. One semi-supervised method uses an offline model which is trained on previous knowledge of specific foreground sounds. The offline model is used in conjunction with an online method to classify foreground and background. [5]

Hidden Markov model (HMM) is another multimodal background subtraction option. An HMM is a statistical model which aims to predict a sequence of unknown (hidden) variables from a set of observed variables dependant on the hidden state variables. It differs from the standard Markov models in that the state is not directly visible for the observer in an HMM. One way to use an HMM is to build the model using two parts. The first one is an HMM trained offline on background data. The second part is an online training to account for unusual events which uses the Bayesian adaptation technique [39]. It is also possible to model the background using a combination of HMM, GMM and GMM clustering [19].

Support vector machines (SVM) are also commonly used to classify audio foreground and background. The method uses a binary set of training examples labelled as the class they belong to. The SVM then builds a model which could further classify incoming unlabelled examples based on the training examples. In one technique utilising SVM, a hyperplane is created which separates the feature space into foreground and background audio [17].

To conclude this subsection, there are both pros and cons of using more online versus more offline training. Online training is simpler and easier to calculate while being prone to model drifting. The methods using more offline based methods are effective on specific scenarios but require training data and are unable to adapt to the background. [9]

Chapter 3

Overview

This chapter will start with a description of the setup during this project, what important hardware components that are involved as well as important software. This is followed by an overview of the system and its components.

3.1 Setup

The program that is developed is intended to be implemented on surveillance cameras. For that reason a camera device is needed to carry through with such an implementation.

The program itself is developed on desktop computers and this is also where the initial tests are performed until implementation on the camera is completed. During this stage audio files are recorded with the camera's microphone so that the program can be tested on audio with appropriate characteristics.

In the next stage, when the program is implemented on the camera, the camera is used to perform final tests and to observe how the bitrate adapts to the content of the audio. During these tests, the camera's microphone can be used as audio input and it is also possible to feed the camera with external audio, e.g. pre-recorded and prepared test files.

The company provides two desktop computers which are used throughout the whole project. These are needed to develop and test the program and to connect to the camera. The program is written in C for compatibility reasons, since that is the code language used on the camera.

Two different devices are used to record test material. The microphone of the

surveillance camera is used to record audio in one environment and a Zoom recording device is used to record in another environment. The reason that a Zoom device in particular is used is because it is provided by the company. The recordings are used for testing during development as well as for evaluation.

Plenty of different software and helpful tools are used during the project but the ones that are particularly useful and are worth mentioning are Audacity and VLC media player. Audacity is used to analyse results from the program. By importing the resulting audio file (that is processed by the program) into Audacity it is possible to get a look of its waveform and compare it to a ground truth file to see if the correct sections have been classified as activity. Since multiple files can be imported to Audacity, it is possible to compare the resulting file with previous results where other parameters or methods have been used. Audacity also has a feature for viewing spectrogram of audio files to see intensity in different frequencies. Besides the visual comparisons, Audacity can also be used to listen to selected sections in the audio file, for example to know what kind of audio that has been incorrectly classified. Audacity is also used to create ground truth files and to create and edit test files. With Audacity, it is for example possible to take parts of audio to then edit and cut it into a test file to contain various events that should get detected. Audacity can also be used to generate white noise with different amplitudes so that test files can be created with various signal-to-noise ratios.

VLC can be used to stream audio from the camera to the media player. The media player also has a statistics view that shows the current bitrate of the audio in real time. These two features make it possible to observe changes in bitrate when the program is being run on the camera.

3.2 System Overview

This section describes how the implemented program will fit together with the other components of the camera, since the activity detection and bitrate reduction is intended to be implemented on a camera.

The program will run directly on the main processor of the camera which runs on a Linux system. It will run independently and use the Linux system to capture audio frames and control the bitrate. ALSA (advanced Linux sound architecture) PCM (pulse-code modulation) API (application programming interface) is used to capture the audio frames and D-Bus (desktop bus) is used to communicate with the streaming application to control the bitrate. The overview is explained graphically in Figure 3.1

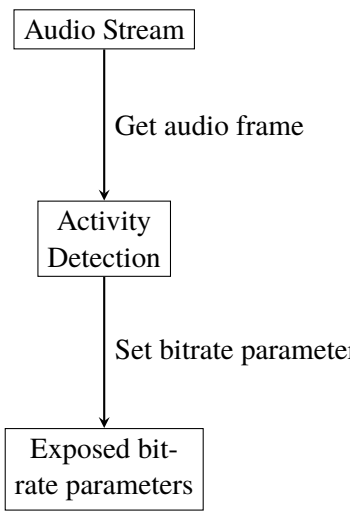


Figure 3.1: The arrows shows the data flow of the system. The activity detection algorithm requests the audio frame from the stream and sets the exposed bitrate parameter according to the classification of that frame.

Chapter 4

Method

In this chapter, the methodology to adaptively compress the audio depending on content is described. The problem itself can be divided into two major sub-problems. The first is to analyse an incoming audio stream and make frame-wise decisions whether there is activity or not. To make such decisions, definitions of activity and non-activity are needed. The second sub-problem is to reduce the data size of the incoming audio stream by compressing with different target bitrates depending on decisions made in the former sub-problem. The idea is to put lower target bitrates for frames that are classified as non-activity. The hypothesis is that this will reduce the resulting data size, especially for long sequences of silence, while keeping high quality for audio that could be interesting. The procedures to solve the sub-problems are followed by a section about strategies regarding evaluation. How the result is measured, interpreted and compared is described here.

4.1 Audio Activity Detection

To be able to detect activity in an audio stream, a definition of activity and non-activity is needed. Surveillance takes place in a wide variety of environments where both foreground events as well as background events have different meanings. For this, a generalisation of what constitutes as background in surveillance scenes is done to be able to make decisions that are accurate for many use cases. For that reason non-activity is simply defined as background noise, where all sounds that deviate from the background noise are considered as activity.

In the theory chapter, multiple ways of detecting activity in audio are explained. Three different approaches were decided to be implemented and tested. One is to calculate energy in audio and comparing it to an adaptive threshold, where energy above the threshold is classified as activity and energy below it is classified as non-activity. Another approach is to use GMM to model the background and then analyse the PSD in the audio to check whether it belongs to background or foreground. The third approach is to use existing VAD software for detecting activity.

In order to implement and test these activity detection algorithms, audio is read and analysed frame by frame. One frame contains a number of samples that depends on the sample rate of the audio input and the length of the frame. If an audio input is sampled with 48 kHz and a frame length of 60 ms is used then one frame contains 2880 samples. Each analysed frame produces a binary output where a one means that activity was detected within the frame and a zero means that no activity was detected. The output from each frame is printed to an output file which is compared to a corresponding ground truth file during testing. This is explained further in Section 4.3.5.

4.1.1 Volume detector

A basic volume detector was created early in the implementation stage, and was thought to be valuable for comparing with upcoming results. The way it functioned was simply by looking at the values in the frame, if the frame contained values higher than a certain threshold the frame would be labelled as containing a sound event. Otherwise, the frame was labelled as non-activity. For this to work, the threshold needed to be set manually for each audio file and recording case. The evaluation can be seen in Equation 4.1

$$\text{if} \begin{cases} \max\limits_{x_i \in X} |x_i| > T, & Event = 1 \\ \text{else}, & Event = 0 \end{cases} \tag{4.1}$$

where x_i are the individual samples in the frame X and T is the threshold.

Because initial tests of the algorithm showed a recall of only 16 %, it was not developed further and is not mentioned in the results, Chapter 5.

4.1.2 Energy functions

As stated in the theory, a very common way of detecting audio events is using energy functions. Since this feature is used so commonly, it was implemented and tested. A decision was made that time and resources only allowed for implementation of two different energy algorithms which are described below.

The energy-based algorithms implemented use the formula found in Section 2.2.1, which can also be seen in Equation 4.2. The energy for a frame is calculated as the mean

of the samples squared values.

$$E_j = \frac{1}{N} \sum_{i=(j-1)N+1}^{jN} x_i^2 \tag{4.2}$$

E_j is the calculated energy for the j:th frame, N is the amount of samples and x_i is the sample point on index i.
There are a few differences of the use of energy from the basic volume detector which only bases its detection of the biggest absolute sample value in the frame. Primarily, by looking at every value of the frame, the method becomes less prone of being impacted by noise. A single spike of noise would make the volume detector labelling the frame as activity, while the energy averages out the signals and could possibly ignore the noise spike. Another point is that the energy also takes in consideration every sample from the frame, which makes it possible to register at least some long lasting, low intensity audio events.

Adaptive threshold

One of the implemented energy functions contains an adaptive threshold. This feature is described in multiple sources including [24]. In the paper, there are several adaptive threshold algorithms described which are also tested and compared. The method that was easiest to implement and had decent results was the linear energy-based detector. This algorithm was therefore implemented, where the energy threshold followed the following Equation 4.3.

$$\text{if} \begin{cases} E_j > kE_r, & Event = 1 \\ \text{else}, & Event = 0 \end{cases} \tag{4.3}$$

E_j is the calculated energy for the j:th frame and E_r is the threshold energy calculated according to Equation 4.4. k is a scaling factor to the threshold, greater than 1, allowing for a safe band for the adaption of E_r.

$$E_{r,new} = pE_{r,old} + (1 - p)E_{silence} \tag{4.4}$$

$E_{r,new}$ is the new threshold energy, p is a threshold constant with $0 < p < 1$, $E_{r,old}$ is the old threshold energy and $E_{silence}$ is the value of the most recent noise frame. p could in this case be considered a learning rate since a smaller value of p would result in a longer time for the energy function to adapt. Even though adaptivity is sought after, surveillance audio is in general similar over time. Therefore, the constant could be set very low resulting in a longer adaptivity time, but a more robust classification. The other algorithms in the paper appeared to have too large p values which were not desired.

In [15], the authors use another method of modelling an adaptive threshold based on the presumption that the recent n frames would model the background sound. This

was implemented, where the energy values from the n recent frames was used to model the background according to Equation 4.5

$$\mu_{E_k} = \frac{1}{n} \sum_{i=1}^{n} E_i \tag{4.5}$$

where μ_{E_k} is the average of the n recent frames and E_i is the energy level of the i:th frame.

The maximum energy level of the n recent frames is calculated according to Equation 4.6

$$E_M = \max_{E_j \in (A_n \cup \{E_k\})} E_j \tag{4.6}$$

with E_m being the maximum energy value of the n recent frames, E_j is the energy level of the j:th frame, E_k is the energy level of the current frame and A_n is the set of the n recent energy values according to $A_n = \{E_1, E_2, \ldots, E_n\}$.

A normalised energy level M_k is then calculated using μ_{E_k} and E_m as can be seen in Equation 4.7.

$$M_k = \max\left[\frac{E_k - \mu_{E_k}}{E_M}, 0\right] \tag{4.7}$$

The normalised energy level is then compared to a preset threshold M_{thresh}. If the value is higher than the threshold, $M_k \geq M_{thresh}$, the current frame is considered as containing activity.

4.1.3 GMM-based background modelling

To see if more complex methods such as Gaussian mixture models, hidden Markov models or support vector machines would yield good results, one such algorithm was decided to be implemented. As stated in Section 2.2.6, the different complex, multimodal approaches have different pros and cons. Neural networks, SVM:s and HMM:s all require training data which results in some problems.

Firstly, the gathering or selection of training data is a problem. In order to train a model, training data is required. There are two options, either to record own samples or to use existing sound banks and neither option is very appealing. Recording and labelling own material would require recordings from multiple different environments to get versatile training data. Explicit knowledge of what sounds that are of interest and record them would also be necessary. To instead select sound banks would perhaps need permissions and licenses, with the problem still being to know what is of importance in surveillance recordings. With that knowledge, the sound banks must then contain such sounds and one would have to add those specific sounds to the training data.

Secondly, the pro mentioned in Section 2.2.6 of being effective on specific scenarios would be hard to implement since the cameras are intended to work in all different

environments. The goal is that the customer should not be needing knowledge in audio, so making the user decide scenario and environment on installation is not optimal.

Also mentioned in Section 2.2.6, using the more offline models limits the performance of background adaptation. Adapting to the background is important, since audio recordings typically can go on for weeks or even years.

Finally, when the offline model has calculated what the sound consists of, the algorithm would have to make a choice of what audio is of interest and what is not. This is not simple and would require excessive time and resources to consider and decide.

With this in mind, the GMM is one of the better options, since no training data is required. The GMM is also simpler and easier to calculate, and the con of being prone to model drifting is not a problem since the algorithm is intended to adapt to the background.

An overview the solution method can be found in Figure 4.1.

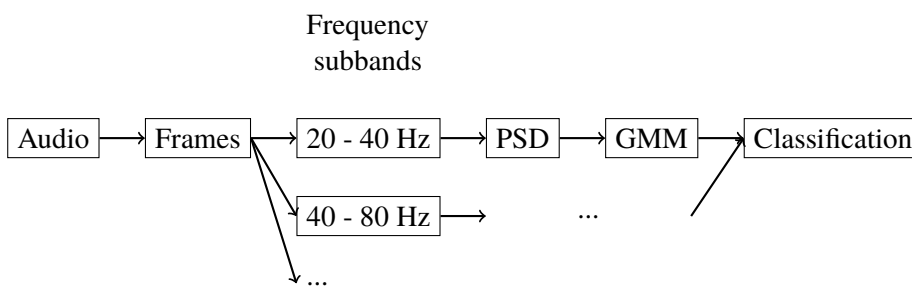


Figure 4.1: Graph of the basic overview of the system, from audio to classification. The audio gets divided into frames. The frames are then analysed and the frequency components extracted. A PSD value is then calculated for each subband which is used as a feature in the GMM. The results from the GMM are then used in the classification process.

Initial approach

The method developed by Cristani et al. [8] that is briefly explained in Section 2.2.3 was used as inspiration for the initial algorithm. The features used in the article is the power spectral density on individual frequency subbands which is explained in Section 2.2.5. The subbands are logarithmically spaced, where the limits of the subband are doubled at each band. This spacing method is used in the Cristani et al. paper, but were earlier used by Peltonen et al. in [20]. For each of the subbands, a GMM is created where a detection in a single frequency band results in the algorithm deciding the frame as containing an event.

There are different ways of updating the models and the initial approach follows the update method described by Cristani et al. in [8]. The following is done individually

for each of the frequency subbands.

The Gaussians are sorted according to descending order using the ω/σ value, where ω represents the weight of the Gaussian and σ the Gaussian's standard deviation. When receiving a new PSD value, it is decided as a successful match if it were to fall within 2.5σ of one of the components. If the signal would not be matched with any of the Gaussians in the mixture, the least likely Gaussian is replaced in favour of a new Gaussian with the mean equal to the current PSD value, with a high standard deviation and a low weight. If however the value were to be matched, Equation 4.8 is used to evaluate if the signal is to be labelled as background or not. If the sum of the ordered weights up until and including the matched Gaussian is bigger than a preset threshold, the frame is labelled as foreground.

$$\sum_{r=1}^{r_{hit}} \omega_r^{(t)} > T \tag{4.8}$$

r_{hit} is the index of the matched Gaussian, $\omega_r^{(t)}$ is the weight for the r:th Gaussian at time t and T is a predefined threshold. The idea behind this is that the background levels would appear frequently and thereby increase the weights and possibly lower the standard deviation. The common Gaussians would then be ranked first. If the matched Gaussian has a lower weight, it will be at the end of the ordered list. The summed weights would then be the sum of all coefficients and would thereby be greater than the threshold. If however the matched Gaussian would be the first in the ordered list, i.e. the most common, the sum would only consist of the weight of that Gaussian, which hopefully will be less than the threshold itself and the frame would be classified as containing no event.

The weights are updated according to Equation 4.9.

$$\omega_r^{(t)} = (1 - \alpha)\omega_r^{(t-1)} + \alpha M^{(t)} \tag{4.9}$$

Here, $\omega_r^{(t)}$ is the weight of Gaussian ranked r at time t, α is the fixed adaptive learning rate and $M^{(t)} = 1$ for the matched Gaussian at time t and $M^{(t)} = 0$ otherwise.

For the unmatched Gaussians, the means and standard deviations remain the same. For the matched Gaussian however, the parameters is updated according to Equation 4.10 and 4.11

$$\mu_{r_{hit}}^{(t)} = (1 - \rho)\mu_{r_{hit}}^{(t-1)} + \rho z^{(t)} \tag{4.10}$$

$$\sigma_{r_{hit}}^{2\,(t)} = (1 - \rho)\sigma_{r_{hit}}^{2\,(t-1)} + \rho(z^{(t)} - \mu_{r_{hit}}^{(t)})^2 \tag{4.11}$$

where $\rho = \alpha\mathcal{N}\left(z^{(t)}|\mu_{r_{hit}}^{(t-1)}, \sigma_{r_{hit}}^{(t-1)}\right)$, $\mu_{r_{hit}}^{(t)}$ is the mean of Gaussian ranked r_{hit} at time t, $z^{(t)}$ is the measured PSD value at time t and $\sigma_{r_{hit}}^{2\,(t)}$ is the variance (or squared standard deviation) for the Gaussian ranked r_{hit} at time t. The idea of this is to adapt the Gaussians according to the incoming values.

This does not work however. After much testing, it was realised that the sum of weights containing only one coefficient did at some points become larger than the threshold. This is the result of surveillance audio with the vast majority consisting of background audio.

Improved approach

The problem experienced was described by Moncrieff et al. in [18]. They conclude that there are two major problems with the method described in [8] by Cristani et al.

1. There is an implicit assumption that background audio does not dominate the audio stream. With portions of audio background bigger than T, background audio can be missclassified as foreground.

2. There are problems in the update methods resulting in an adverse effect on the adaption of the model. The model becomes unsolvable using the update approach and results in a reduction of the influence of the incoming observations.

To combat this, Moncrieff et al. proposes some changes to the method done by Cristani et al. Firstly, the ranking of the models ω/σ is altered to instead be ranked only using the weights, ω. For the visual case of GMM background modelling brought up by Stauffer and Grimson [26], the authors assume a decrease of the standard deviation over time. This is however not guaranteed for the case of audio due to its variability. Moncrieff et al. also proposes the foreground determination according to Equation 4.12

$$\sum_{r=1}^{r_{hit}} \omega_r < T \tag{4.12}$$

where a true condition results in the determination of foreground. This is very similar to what Cristani et al. used in [8], but with the inverse condition. For this to work, Moncrieff et al. uses the inverse ranking with the Gaussian with the highest rank having the lowest weight. The threshold T is also recalculated as $T_{Mon} = 1 - T_{Cri}$. With these changes, the first problem brought up in [18] was solved.

To deal with the second problem, Moncrieff et al. uses a reviewed weight update formula according to Equation 4.13.

$$\omega_r^{(t)} = (1 - \alpha M^{(t)})\omega_r^{(t-1)} + \alpha M^{(t)} \tag{4.13}$$

This update method is more passive and limits the effect of noisy audio as the weight are not as penalised when a non-background model is matched. The ρ is also calculated using a slightly modified formula as can be seen in Equation 4.14.

$$\rho = \alpha \frac{\mathcal{N}(z^{(t)}|\mu_{r_{hit}}^{(t-1)}, \sigma_{r_{hit}}^{(t-1)})}{\mathcal{N}(\mu_{r_{hit}}^{(t-1)}|\mu_{r_{hit}}^{(t-1)}, \sigma_{r_{hit}}^{(t-1)})} \tag{4.14}$$

This value of ρ is scaled to have a maximum value of 1 which prevents some issues, for example could the standard deviation get a negative value otherwise.

When deciding on which of the Gaussians to replace in the case of a non-matched value, a counter regarding how long ago the Gaussian was matched is used. The Gaussian with the highest valued counter i.e. the one matched the longest ago is selected.

With this approach, a set of one dimensional Gaussian distributions is used for the modelling of each PSD value separately. In reality, the frequencies depend on each other and should with that in mind be modelled together. To achieve this, a single multidimensional GMM with one dimension for each feature can be used instead. This is another option brought up by Moncrieff et al. and with their testing, they conclude that this option is slightly better. Because of limitations in time, this was never implemented since the one dimensional mixture models was implemented initially according to [8]. As a note, in [18], the formulae are adapted for the use of a multidimensional Gaussian while the equations in this report are specific for one dimensional Gaussians.

Value initialisation

The values of the Gaussians were initialised using the K-means clustering algorithm. Another common way of initialising Gaussians is the Expectation Maximisation (EM) approach. K-means was however selected instead of EM because it is faster and since the algorithm probably will be ongoing for a long while, it will matter little what the values are initialised as.

The cluster center initialisation decides the performance of the K-means algorithm and is therefore selected carefully. The first center is selected at random. For the rest of the centers, a distance to the nearest cluster center is calculated. The sample with the longest such distance is set as an initial cluster center.

The K-means clustering algorithm needs however to be altered in order for it to function in conjunction with the GMMs. With the algorithm, cluster centers are given. These are used as the mean values of the Gaussians. For the standard deviation and the weights, the method used as initialisation is described in [4]. The formulae used for calculating the standard deviation and weights can be seen in Equation 4.15 and 4.16

$$\sigma_k^2 = \frac{1}{|C_k|} \sum_{x \in C_k} (x - \mu_k)^2 \tag{4.15}$$

where σ_k is the standard deviation for the k:th Gaussian, also k:th cluster from the K-means clustering, C_k is the set of samples clustered to belong to the k:th cluster and μ_k is the mean of the k:th Gaussian.

$$\omega_k = \frac{|C_k|}{N} \tag{4.16}$$

ω_k is the weight for the k:th Gaussian and N is the amount of samples.

Although mentioned above that the initialisation will not matter so much because of the long time the algorithm will run, it has some effect on the standard deviation when a new Gaussian is formed. The papers studied only mention to assert the new Gaussian's standard deviation with a high value. The different frequency subbands have different orders of magnitude on their PSD values. Because of this, to choose a standard deviation fit for all frequency subbands, the standard deviations are chosen and stored independently. They use the calculated standard deviations for all Gaussians and estimate an average standard deviation. By multiplying the average with a fix constant bigger than 1, a high standard deviation is formed.

The K-means clustering, calculation and assignment of means, standard deviations and weights are all part of a start-up phase of a few seconds. After this point, the Gaussian mixture model for modelling the audio background can begin.

Optimisation strategy

In order to optimise the hyperparameters of the GMM approach, an automated grid search algorithm was implemented. There were mainly six different parameters to change in order to optimise the functionality, but if they were to be optimised manually, it is quite likely that only a local maximum point would be found. With grid search, a range for each of the hyperparameters is selected and the algorithm will run the program with each set of hyperparameters. The results are saved.

The strategy was to firstly run the grid search with logarithmically spaced hyperparameters to get a rough understanding of where the global maximum point is. This rough estimation is then used in a second grid search with linearly spaced hyperparameters around the estimation.

4.1.4 Voice activity detectors

The two VADs that are tested come from the G.729 codec and the WebRTC project. These particular VADs were selected because they are open source and because they can easily be tested as stand-alone functions. Since voice activity detectors are designed to detect speech it may turn out that they will not suit the problem very well. They are still interesting to test because they have a lot of work put into them and they are easy to test. It is also interesting to compare them to the other approaches, which have less work behind them but are more designed for surveillance scenes.

The G.729 VAD is tested in a MATLAB-script that reads an audio file. The audio file is split into frames of 60 ms where each frame gets analysed by the VAD. The VAD outputs a one if activity was detected within the frame and a zero if no activity was detected. For this VAD, there are no easily adjustable parameters so it is tested with its default settings. MATLAB is used for testing this VAD since it has an implementation in this language and can easily be used as an external function in a MATLAB-script.

The WebRTC VAD is tested in a similar way, except that it is done in Python. Python

is used here because there exists an implementation of this VAD in Python that is simple to import and test. The VAD is tested on 30 ms frames since that is the highest frame size it supports. As mentioned in Section 2.1.4, this VAD has a setting with four different levels of sensitivity and each of these levels is tested to see which level that produces the best result.

Both the G.729 VAD and the WebRTC VAD also have source code in C which can be used if one of these algorithms is used in the final implementation. They are however tested at first in MATLAB and Python respectively for simplicity reasons.

4.2 Compression

An uncompressed audio stream consumes a lot of data and results in a very high bitrate. This high bitrate is unnecessary since audio can be compressed and still maintain quality. To make the compression more efficient and reduce the bitrate even further, different frames of the audio can be encoded with different bitrates. If it is possible to find out which frames that are uninteresting and have low requirements on quality then these frames could get compressed with a really low bitrate. The ability to encode in different bitrates during runtime is needed and a codec that supports this is Opus. With Opus, the audio is compressed with lossy compression techniques to achieve significant reductions in data size.

The following subsections will describe the approach when compressing the audio, which depends on the output from the audio activity detection. First comes an explanation on why continuous transmission is used instead of discontinuous transmission. This is followed by an explanation and motivation for two particular strategies that could improve the results. The first one being a gradual bitrate decline which will provide a smoother transition when the bitrate switches from high to low. The second strategy is to use a post event classification which will help to capture and encode a complete event instead of parts of it. This section ends with a brief subsection on why the Opus codec fits well for the problem and how it is utilised to solve the problem.

Continuous transmission

When only considering bitrate, an optimal system would recognise every non-interesting audio frame and discard these frames from transmission. This would result in a bitrate of 0 bit/s during such sequences, for example during a night recording where nothing happens. However, when solving the problem it is preferred to not interrupt the transmission for two reasons. One is that the audio detection is not completely bulletproof, there is a risk that events will go undetected. The other reason is that a recorded audio stream that contains gaps will not provide the same credibility as a continuous recording, for example when it is used as forensic evidence. Instead of discarding frames that have been classified as non-activity, they are transmitted anyway but heavily compressed beforehand. Therefore, even if an event goes undetected it will still get saved

and provide the listener with information even though the quality will be worse for that piece of information than for the detected events.

4.2.1 Gradual bitrate decline

When detecting activity, the bitrate should be switched to high as soon as possible so that the quality of the activity will not get compromised. However, when the activity ends and non-activity is detected there is little use in an instant switch back to low bitrate. In the case of multiple events occurring with a short silence between them, instantaneous switching from high to low bitrate and back would result in a less pleasant audio recording. The proposed alternative is to use a gradual decline of the bitrate when non-activity is detected after a sequence of activity, see Figure 4.2. This gradual decline allows for a smooth transition when it is time to switch to low bitrate and the decline itself is controlled by two parameters:

1. Number of frames between each bitrate decrease.

2. Bitrate decrement for each decrease.

The bitrate is decreased according to these parameters until it reaches a lower bound or until activity gets detected.

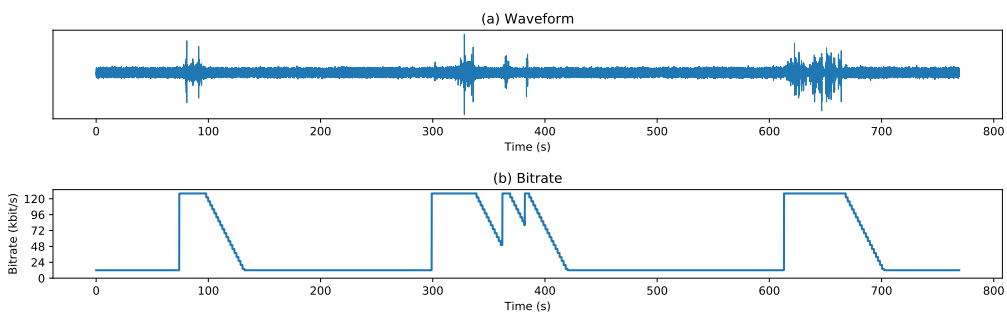


Figure 4.2: The upper graph shows the waveform of an audio recording with a few events. The lower graph shows the bitrate when compressing the recording. The bitrate instantly increases to its upper limit when activity is detected and is gradually decreasing towards its lower limit when the activity ends.

4.2.2 Post event classification

Some events naturally have small sequences of silence in them, for example in a voiced conversation where there usually are silent pauses between sentences. Such pauses

would result in a frequent switch of detecting activity and detecting non-activity which would cause a choppy result if the bitrate is updated accordingly. To solve this problem and to encode the entire event with the same bitrate a strategy called post event classification is used. The idea is that when any activity gets detected the classification of activity is prolonged for n frames, which is intended to make the program capture the entire event and encode it with high bitrate.

This is implemented by keeping a counter of the number of frames that is to be classified as activity after activity has been detected. If activity is detected within a frame the counter is reset to n. If activity is not detected within a frame and the counter is greater than zero, the frame is classified as activity and the counter is decremented. If activity is not detected and the counter is zero, the frame gets classified as non-activity. An overview of the algorithm containing this strategy as well as the gradual bitrate decline is presented in Figure 4.3.

4.2.3 Opus compression

The Opus codec is used to encode the audio using a lossy compression technique. Opus has an impressive flexibility when it comes to encoding in different bitrates. It has also been proven to produce great quality compared to other audio codecs [22]. By using Opus, the ability is given to switch between over 1200 different target bitrates between 6 and 510 kbit/s, all during runtime. This makes it possible to encode with low bitrate for non-activity frames and with high bitrate for activity frames and it also allows for the gradual decline mentioned in Section 4.2.1.

In Section 2.1.3, it is mentioned that Opus has the feature to encode in VBR which increases or decreases the bitrate depending on content. This is not too far from the goal of this thesis. The difference is that the approach in this thesis is significantly more aggressive than the VBR of Opus. When encoding with VBR in Opus, the quality of uninteresting sound is still high and the bitrate does not reach the low levels that are sought after during long sequences of non-activity.

4.3 Evaluation Strategy

This section will go through the strategy for evaluating the results. There are four aspects that are interesting to consider during evaluation. The first one is how well the activity detection performs and for this three different measurements are used: precision, recall and accuracy. The second one is the resulting quality on the audio after encoding. The third one is how much the data size is reduced. Lastly, the forth aspect is how well the algorithm handles noise. This section ends with a description and motivation for the test files that were used during development and testing as well as how these were obtained.

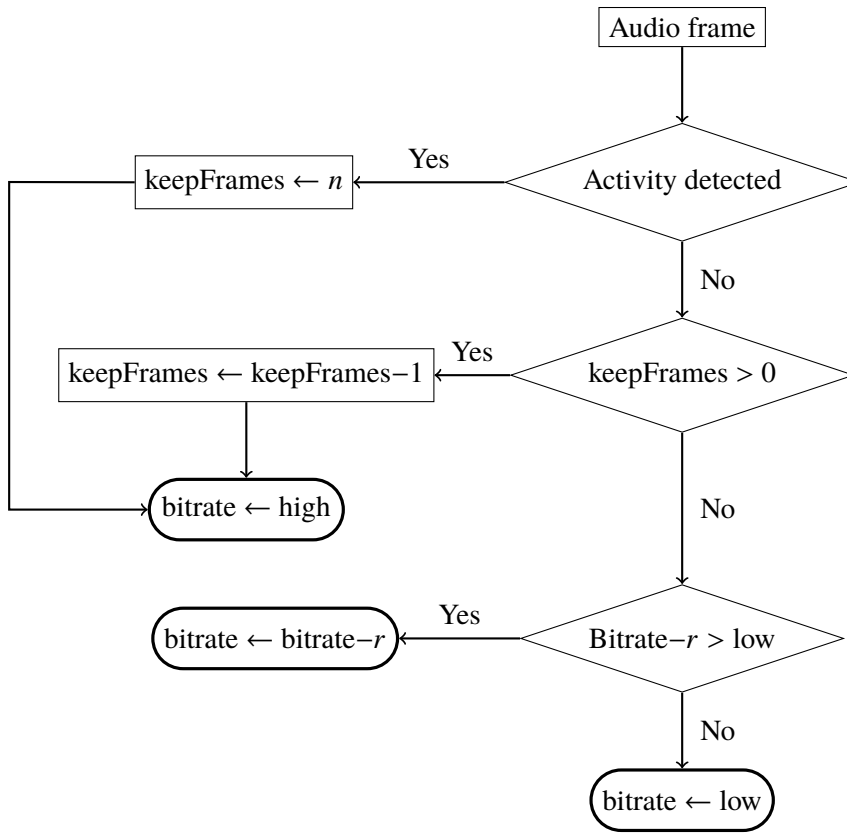


Figure 4.3: A scheme over the algorithm that controls how the bitrate is updated. The post event classification is handled by the 'keepFrames' variable, which is a counter that tells how many frames to keep after a detected event. The rounded rectangles are the end nodes, which are the possible decisions when setting the bitrate. It is either set to high or low or decremented by the rate r.

4.3.1 Precision, recall and accuracy

To calculate the measurements in this section, four values are used which are described below.

- TP is the amount of true positives, i.e. number of frames that are correctly classified as activity.
- TN is the amount of true negatives, i.e. number of frames that are correctly classified as non-activity.

- FP is the amount of false positives, i.e. number of frames that are incorrectly classified as activity.
- FN is the amount of false negatives, i.e. number of frames that are incorrectly classified as non-activity.

With these values, it is possible to calculate the three measurements used when evaluating the result. The first one is precision which tells how big portion of the activity classified frames that were correctly classified according to Equation 4.17.

$$Precision = \frac{TP}{TP + FP} \tag{4.17}$$

The second measurement is recall, which tells how big portion of the activity that was detected, see Equation 4.18.

$$Recall = \frac{TP}{TP + FN} \tag{4.18}$$

The third measurement is accuracy, which tells how big portion of all the frames that was correctly classified according to Equation 4.19

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.19}$$

Accuracy is often used as a measurement when comparing classification results but it does not tell if the inaccuracy comes from false negatives or false positives. By using recall and precision it is possible to compare such information. Both recall and precision are reflected in the accuracy, a 100 % accuracy also means that the recall and precision are 100 % since there are no false negatives or positives in that case. The goal is to get high scores on all of these measurements but some have higher priority than others.

It is important to detect as much activity as possible since one purpose of surveillance is to record data. If the data gets compromised then the surveillance system has lost its usage. For this reason it is crucial to get a high score on recall, a 100 % score on recall means that all activity was successfully detected. Precision is important for another reason, a higher precision means that less data is incorrectly classified as activity which means that the data can be compressed to a further extent. Since it is more important to not lose any valuable data than it is to reduce the data size it is better to get a high score on recall at the cost of losing some precision. The precision cannot be too low though, there is no purpose to have 100 % recall if the precision is very low because that just means that all the audio frames get classified as activity no matter what content they hold. If all of the non-activity frames get incorrectly classified then everything will be compressed with a high bitrate and the audio activity detection is no longer beneficial. It is therefore preferred to get a high accuracy together with a high recall so that the majority of inaccurate decisions stems from frames that are incorrectly classified as activity and not the other way around.

4.3.2 Audio quality

As mentioned in the Section 1.2 describing the problem, the intention is to compress the audio while keeping the audio quality acceptable. In order to properly test this, excessive tests and surveys would be optimal. Because of the lack of both time and resources (e.g. test persons), this was solved by simply adjusting the bitrates of the output until the quality would be considered acceptable by us, the authors. The quality optimisation could however not be done without the resulting data size in mind.

4.3.3 Data size

Since the algorithm is intended to compress the audio file's storage size, an important measurement of performance is the data size of the output file. This is compared with the output file where no activity detection is used and everything is encoded with high bitrate. It is also compared to the ideal data size which is the data size of the output file when a perfect activity detection is used. The ideal data size can be calculated according to Equation 4.20

$$Datasize = BR_{high} * A + BR_{low} * (T - A) \tag{4.20}$$

where BR_{high} and BR_{low} is the high and low bitrate in kbit/s, respectively. A is the duration of activity in seconds in the input file and T is the duration of the input file in seconds.

4.3.4 Noise influence

The algorithm would ideally work no matter the noise level, and a test for measuring the SNR was created to investigate how the influence of noise affects the outcome of the algorithm. Usually, SNR is calculated on a system rather than on a file. In this case however, there are no reason to calculate the system's SNR, but instead how well the algorithm handles environments with different amount of noise is of interest.

Special measurements and calculations had to be done to adapt the SNR calculation methods to that of a file. The common formula for calculating SNR using voltages is shown in Equation 4.21

$$SNR(dB) = 20 \log_{10}\left(\frac{V_s}{V_N}\right) \tag{4.21}$$

which can be extracted from Equation 2.3. V_s denotes the voltage of the signal and V_N the voltage of the noise.

With digital signals in the audio files, some approximations were made. Firstly, the input signal was modelled as a sine wave. The voltage of the signal could then be approximated as

$$V_s = \frac{A_{max}}{\sqrt{2}}$$

where A_{max} is the maximum amplitude of the input signal. The voltage level of white noise is estimated as the peak-to-peak of the maximum noise levels divided by 6.6 [10].

$$V_N = \frac{V_{N,pp}}{6.6}$$

With these estimations, the accuracy, recall and precision was derived for multiple different SNR values.

4.3.5 Test files

In order to get a measurement of the above mentioned evaluation aspects, multiple test files were created to test different things. To measure precision, recall and accuracy, a file containing the ground truth was necessary for each of the test files. Whenever an event occurred, the ground truth file should contain a label of event. Such ground truth files were created by carefully listen to the audio and by manually label every event. The test files are sampled with 48 kHz since it allows reconstruction of frequencies up to 24 kHz, which includes the entire human hearing spectrum.

The first test file contains few events. This gives an idea of how well the algorithm handles quiet environments. The opposite, how well the algorithm handles environments with large amount activity is also of interest and the second test file contains this scenario.

The third and fourth test files contains combinations of parts with few events and parts with frequent events. The difference between the two is in which environment the recording was done in. If all files would be done in the same environment, the results would not have the same credibility because of the lack of diversity. Even if the two recording environments are in the same building, they were done on different floors as well as with different devices and during different dates and times.

The next two test files are equal to the third and fourth, but they have an increased noise level. The reason behind this is to get an understanding of how well the algorithm handles noise.

To get an even more accurate understanding of how the algorithm handles noise, a seventh and last test file was created with manually put in events and silence at the other parts.

There is however another test file that was used only during the hyper-parameter optimisation. This file has low to average noise level and contains both silent and eventful parts. Would the testing be done on the same files as the one used for optimising the hyper parameters of the algorithm, the result would be biased. This file was therefore not included in the final evaluation in order to avoid a biased result.

A list of the used test files can be found in Table 4.1 where the events in these files are typical surveillance events such as door shutting, footsteps and voice.

Table 4.1: Seven test files that are used for the final tests of the implemented methods.

File ID	Content	Environment	Duration (min)	Activity proportion
1	Little activity	1	60	7 %
2	Much activity	1	60	89 %
3	Combination of File 1 and 2	1	77	42 %
4	Moderate activity	2	60	29 %
5	File 3 with amplified noise	1	77	42 %
6	File 4 with amplified noise	2	60	29 %
7	SNR measuring file	-	32	10 %

Chapter 5

Results

The results presented here are the outcome when the methods in the previous chapter have been applied. This chapter follows the same outline as the previous chapter and it begins with a presentation of the results from the investigated audio activity detection methods. This is followed by a section with compression related results such as resulting data size.

5.1 Audio Activity Detection

The results for the three methods described in Section 4.1 are presented in the following subsections. The measurements mentioned in Section 4.3.1 are used to show how well the methods perform. The results come from applying the methods to the six individual test files described in Section 4.3.5. A mean of the results from the six first files are also presented and this mean is weighted according to the duration of the files.

5.1.1 Energy functions

Two approaches for the adaptive threshold were tested. The first follows Equation 4.3 and 4.4 and the second follows Equation 4.5, 4.6 and 4.7. The first approach produced the best results of the two. The chosen parameters for this approach are presented in Table 5.1 and the results for this setting are presented in Table 5.2. The method had best accuracy on the sixth test file but it only detected less than half of the events. It had best recall on the third test file but it also falsely classified much of the silence as

activity, resulting in a poor accuracy. Taking both accuracy and recall in consideration, the method had the best results on the second test file where it detected 86.86 % of the events and got an accuracy of 79.68 %. The overall score shows that the method manages to find 79.34 % of the events but it falsely classifies much of the non-activity which reduces the accuracy to 60.08 %.

Table 5.1: Parameters for the energy function.

Parameter	Value	Description
f	60	Frame duration (ms)
n	30	Post event classification (number of frames)
p	0.1	Learning rate
k	2	Scaling factor

Table 5.2: Results for the energy function on the six first test files.

	Precision	Recall	Accuracy
1	8.09 %	76.41 %	37.21 %
2	89.94 %	86.86 %	79.68 %
3	47.31 %	89.43 %	53.41 %
4	60.18 %	55.29 %	76.61 %
5	39.89 %	86.80 %	38.81 %
6	86.63 %	47.55 %	82.80 %
Mean	**50.02 %**	**79.34 %**	**60.08 %**

5.1.2 GMM-based background modelling

The chosen parameters for the improved GMM algorithm from Section 4.1.3 is shown in Table 5.3. These were the parameters that provided a good trade-off between recall and accuracy in the optimisation stage. The results when using these parameters are presented in Table 5.4.

For one of the parameters in Table 5.3, σ_{high}, the description is vague since the real explanation does not fit in the table and is instead explained in the following sentences. For the GMM algorithm, when a new Gaussian is created it is assigned a high standard deviation. In order to calculate the high standard deviation, the mean standard deviation from the initialisation (Section 4.1.3) is also used. The high standard deviation is cal-

culated as the product of the mean standard deviation derived during the initialisation and the constant σ_{high}.

The algorithm had a mean accuracy of 86.63 % while detecting 81.05 % of the events. The test file that had the best accuracy was number 3 with an accuracy of 93.43 % while correctly classifying 94.62 % of the activity. The test files that had the worst results were test file 5 and 6 which contained amplified noise.

Table 5.3: Parameters for GMM implementation.

Parameter	Value	Description
N_{SF}	50	Number of start frames for the initialisation
fb	8	Number of frequency bands
G	4	Number of Gaussians per GMM
f	60	Frame duration (ms)
n	30	Post event classification (number of frames)
T	0.0625	Threshold
α_1	0.001	Learning rate when updating the weights
α_2	0.0001	Learning rate when calculating ρ
σ_{high}	9.5	A constant used when replacing Gaussians
σ_{comp}	5.0	Amount of standard deviations from the mean considered a match
ω_{low}	0.001	The weight used when replacing Gaussians

Table 5.4: Results for the GMM approach on the six first test files.

	Precision	Recall	Accuracy
1	35.78 %	82.23 %	88.36 %
2	94.56 %	94.82 %	90.54 %
3	90.31 %	94.62 %	93.43 %
4	77.90 %	77.65 %	87.23 %
5	81.42 %	60.72 %	77.42 %
6	82.13 %	54.60 %	83.52 %
Mean	**84.82 %**	**81.05 %**	**86.63 %**

The parameters from Table 5.3 were determined according to the following descriptions. N_{SF} was decided to not be too long, but still enough to get an estimation of the environment. fb was set to a value of 8, according the article by Cristiani et al. [8].

The same article also uses 4 Gaussians per subband which is the reason behind the value of G. Although a higher value of the post event classification parameter n appears to be optimal according to Figure 5.2, the testing done prior to the results of this figure showed results that would benefit values of n close to 30 which is the reason behind this parameter.

With the previous mentioned parameter decisions, the last six parameters from Table 5.3, namely T, α_1, α_2, σ_{high}, σ_{comp} and ω_{low} was optimised using the optimisation method described in Section 4.1.3.

Noise influence

The influence of noise for the GMM algorithm is shown in Figure 5.1. By increasing the SNR, the algorithm got better at detecting events. From 0 dB SNR to 20 dB SNR, the algorithm went from detecting 22 % of the events to detecting 97 % of the events. The accuracy increased up to 94 % when the SNR went from 0 dB to 10 dB. For higher SNR than 10 dB, the accuracy was decreasing which is caused by falsely detecting more of the non-activity. Taking both accuracy and recall into consideration, the algorithm had best results for 20 dB SNR which had a recall of 97 % and an accuracy of 92 %.

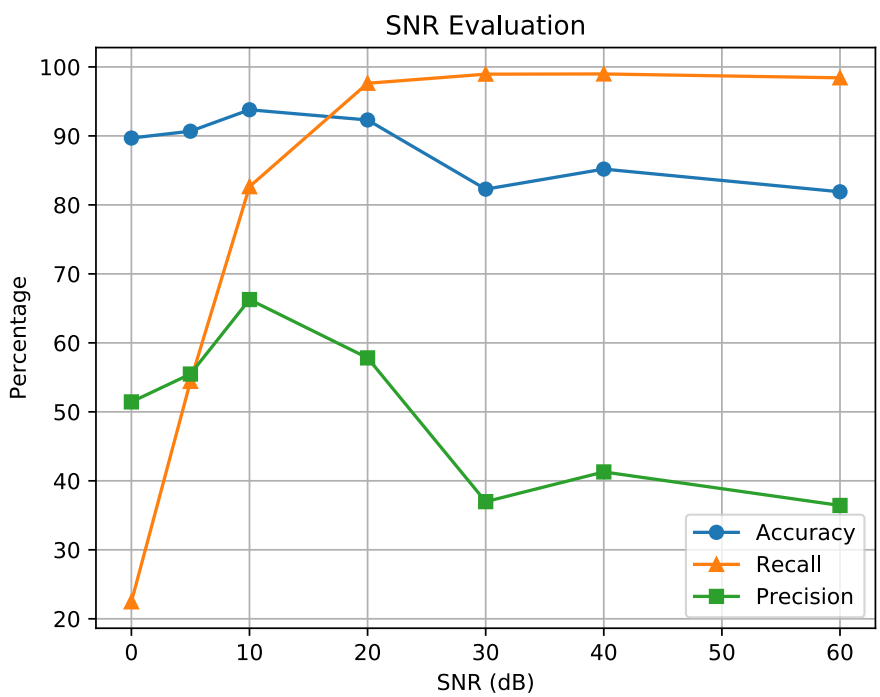


Figure 5.1: The graph displays percentages of accuracy, recall and precision for different SNR values.

5.1.3 Voice activity detectors

The chosen parameters for the G.729 VAD are shown in Table 5.5 and the results when using these parameters are presented in Table 5.6. The VAD had a mean accuracy of 77.85 % and a mean recall of 49.42 %, i.e. less than half of the events were detected. The algorithm did poorly on the noise amplified test files where only 29.59 % and 21.90 % of the events were correctly classified, which reduced the overall mean. The test file that had the best trade-off between accuracy and recall was the third file which got an accuracy of 83.32 % and a recall of 64.72 %. This method can not be recommended for this problem since it only detected half of the events. However, when the method did classify frames as activity, it were correct decisions in most of the cases which is reflected in the mean precision of 91.04 %.

Table 5.5: Parameters for G.729 VAD implementation.

Parameter	Value	Description
f	60	Frame duration (ms)
n	30	Post event classification (number of frames)

Table 5.6: Results for the G.729 VAD on the six first test files.

	Precision	Recall	Accuracy
1	57.96 %	40.23 %	93.74 %
2	96.19 %	65.83 %	67.30 %
3	93.95 %	64.72 %	83.32 %
4	82.55 %	37.25 %	79.68 %
5	90.21 %	29.59 %	68.71 %
6	74.93 %	21.90 %	75.42 %
Mean	**91.04 %**	**49.42 %**	**77.85 %**

The chosen parameters for the WebRTC VAD are presented in Table 5.7 and its results are shown in Table 5.8. All four sensitivity levels for this VAD were tested and the level that yielded the best result was 1. This resulted in a mean accuracy of 60.00 % and a mean recall of 85.71 %. The mean precision of 49.99 % tells that more than half of the activity classified frames were in fact non-activity that had been falsely detected. This means that the algorithm is rather aggressive and classifies too much of the audio as activity, which also has an impact on the good result for recall. This can be seen in the results of test file 5 and 6 especially. They both got a recall of 100 %, which is a perfect

score, but their accuracy were the same as the proportion of events in the files. This means that for these two test files, all frames were classified as activity which render the purpose of the algorithm meaningless.

Table 5.7: Parameters for WebRTC VAD implementation.

Parameter	Value	Description
f	30	Frame duration (ms)
n	60	Post event classification (number of frames)
a	1	Aggressiveness level

Table 5.8: Results for the WebRTC VAD on the six first test files.

	Precision	Recall	Accuracy
1	13.05 %	43.69 %	75.62 %
2	92.64 %	85.20 %	80.83 %
3	47.92 %	83.43 %	54.67 %
4	80.31 %	60.40 %	84.37 %
5	42.38 %	100.00 %	42.38 %
6	28.73 %	100.00 %	28.73 %
Mean	**49.99 %**	**85.71 %**	**60.00 %**

5.2 Compression

In this section, results regarding compression are presented. The audio classification algorithm which is tested in this section is the improved GMM approach. For compression, Opus encoding has been applied according to the classifications using a lossy compression technique. Audio that was classified as activity was encoded with 128 kbit/s and audio that was classified as non-activity was encoded with 14 kbit/s. The parameters used for these results are shown in Table 5.3. The resulting data sizes are presented in Table 5.9, where an instant bitrate switch is used on the six first test files from Table 4.3.5.

When encoding the files with 128 kbit/s, the resulting data size was 380.0 MB. When encoding classified activity with 128 kbit/s and classified non-activity with 14 kbit/s, the resulting data size was 169.6 MB. The six test files contained 39.9 % activity and if 39.9 % of the frames in the files were encoded with 128 kbit/s and the

rest with 14 kbit/s, the resulting data size would have been 176.2 MB. The data size reduction depends on the amount of non-activity in the audio which can be seen when comparing the results from test file 1 and 2. The first file contained 7 % activity and was reduced from 57.8 MB to 14.7 MB while the second file contained 89 % activity and was only reduced from 57.8 MB to 52.2 MB.

Table 5.9: The ideal data size and the resulting data size when setting bitrate according to activity detection, compared to data size without activity detection where the files get encoded with a constant high bitrate of 128 kbit/s.

	Ideal data size (MB)	Resulting data size (MB)	Data size without activity detection (MB)
1	9.9	14.7	57.8
2	51.9	52.2	57.8
3	36.1	37.2	74.4
4	21.1	20.7	57.8
5	36.1	29.1	74.4
6	21.1	15.7	57.8
Sum	**176.2**	**169.6**	**380.0**

5.2.1 Gradual bitrate decline

This subsection shows the resulted data size of the output Opus files. For all cases, the parameters used are shown in Table 5.3. The highest bitrate, set when activity was detected was 128 kbit/s. The lowest bitrate that could be set was 14 kbit/s.

The results are shown in Table 5.10. In the first column, no gradual bitrate decline was used. After an event had occurred and the post event buffer time had passed, the bitrate was instantly decreased to its lower bound. For the second column, the bitrate was decreased by 2000 bits every fifth frame. This resulted in a total decline time of 17.1 seconds. In the third column, the bitrate was decreased by 2000 bits every tenth frame. This resulted in a total decline time of 34.2 seconds.

The resulting data size when using a decline duration of 17.1 seconds was 216.6 MB compared to 169.6 MB of the instant decline. When increasing the decline duration to 34.2 seconds, the resulting data size became 232.9 MB. Increasing the duration even further will also increase the resulting data size, up to the limit of 380.0 MB which is the resulting data size when everything is encoded with the high bitrate of 128 kbit/s.

Table 5.10: The data size of different gradual bitrate decline settings for the six first test files.

	Data size (MB) with instant decline	Data size (MB) with 17.1 s decline	Data size (MB) with 34.2 s decline
1	14.7	24.5	29.0
2	52.2	56.7	57.2
3	37.2	44.2	47.4
4	20.7	27.3	29.9
5	29.1	41.5	44.2
6	15.7	22.4	25.2
Sum	**169.6**	**216.6**	**232.9**

5.2.2 Post event classification

The idea with the post event classification is to improve the chances of capturing entire events and produce a less choppy result. For that reason it is interesting to see how well the program is doing without the post event classification compared to with the post event classification. The result when using different sizes on the post event classification on the improved GMM approach is presented in Figure 5.2.

When a size of 0 frames was used, the algorithm only detected less than half of the activity. When increasing the size from 0 to 40 frames, the recall improved immensely but the precision slightly decreased. The accuracy peaked at 30 frames with a value of 86.63 % but it did not differ much between 20 and 60 frames, During that span however, the recall and precision went in opposite directions. This means that different settings can be used for different scenarios and still have a high accuracy. If it is prioritised to detect activity then a post event classification size of 60 frames can be used, where 87.16 % of the activity is detected. If it is more important make precise decisions then a size of 20 frames can be used, where 86.95 % of the activity classified frames truly are activity.

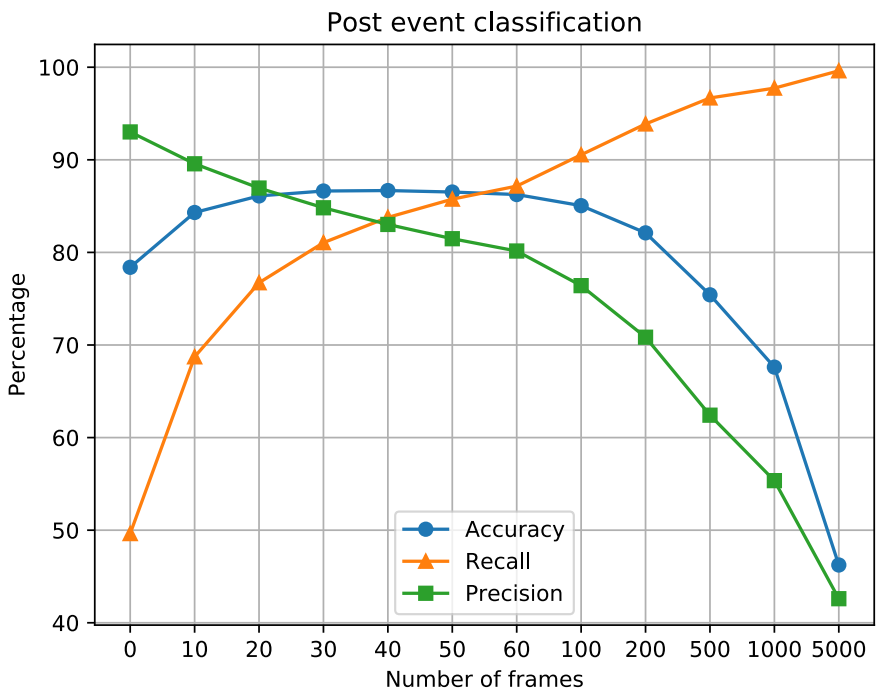


Figure 5.2: The figure shows how the accuracy, recall and precision are affected by different lengths on the post event classification, where each frame is 60 ms.

Chapter 6

Discussion and Conclusions

In this chapter, the results are discussed. The conclusions of this work, as well as ideas for future work are also brought up here.

6.1 Discussion

This section discusses the results from the previous chapter. Some reasoning behind certain decisions are also explained.

6.1.1 Precision, recall and accuracy

The mean result of all test files for each of the methods are displayed in Table 6.1 for convenience.

Table 6.1: Means of precision, recall and accuracy for the tested algorithms.

	Precision	Recall	Accuracy
Energy	50.02 %	79.34 %	60.08 %
GMM	84.82 %	81.05 %	86.63 %
G.729	91.04 %	49.42 %	77.85 %
WebRTC	49.99 %	85.71 %	60.00 %

The results of the GMM approach clearly has the best accuracy of the algorithms. Regarding the recall field, the WebRTC VAD outperforms the others including the GMM algorithm. The reason behind this is not because the WebRTC algorithm is better at detecting events, but because it in the fifth and sixth test file with increased noise, label everything as activity. This can be seen in Table 5.8, where the recall for both of the files are increased to 100% which is the reason why the mean recall is so high for that algorithm. In Table 4.1, the amount of activity for each of the files are described. By comparing those amounts to the accuracy of the last two test files in Table 5.8, they are identical. This is because when everything is labelled as event, the accuracy of the method is equal to the amount of activity.

For the G.729, the precision is higher than that of the GMM. The results however show that not even half of the events are recognised by the G.729 VAD. Since both the accuracy and especially the recall is higher for the GMM algorithm in comparison with the G.729, the GMM should be considered the better event detection algorithm.

It is also worth to notice that the GMM method outperforms the energy based algorithms in all fields. Another point to notice is the result of the added noise. For all algorithms it appeared to heavily decrease the accuracy of the algorithms, where GMM still outperformed the other results. The added noise did however lower the mean of each of the measurements for the GMM algorithm substantially.

6.1.2 Noise influence

Because of the decrease in performance with added noise, the test regarding different noise and SNR levels was created to get a better understanding of when the algorithm performs at its best. As can be seen in Figure 5.1, it appears that the optimal SNR would be somewhere between 10 and 20 dB. At this point, the recall has almost increased to its maximum and the accuracy of the algorithm is still high. An interesting result is that the algorithm appears to have a harder time to label silence correctly with decreased noise. This could be the result of the algorithm's inability to handle very low amplitude signals. With the optimisation of the algorithm, it might have been optimised for cases with SNR around 20 dB which can be another reason for the strange results.

It should also be mentioned that the constructed file contained only 10 % activity. It is most likely the reason behind the algorithm performing so well on even really low SNR levels. Since the algorithm labelled the vast majority of the file as containing no events, the accuracy was high. This is also reflected in the very low recall for those cases.

6.1.3 Ground truth errors

The ground truth labelling technique does also require some discussion. The way the ground truth files were created was simply by listening through the audio and marking

whenever an event occurred as described in Section 4.3.5. There are some dangers around doing this.

Firstly, as the authors of the article, it is in our best interest to have as high results as possible. This could in theory mean that the test files and ground truth files were created with the intention to perform exceptionally well on the created algorithm.

A second danger with this is that the ground truth files could contain the wrong labels. By only listening to the audio and manually label the events, there could be events that we could not hear through the headphones and speakers. Parts where the algorithm classified the audio as activity could for example be an unnoticed event, resulting in errors in the measurements of precision, recall and accuracy.

The third danger with labelling the activity manually is that the start and end of events might be incorrectly labelled. To accurately on the tenth of a second label whenever an event starts or ends is difficult. There might also be time periods between words in sentences or in between close events where the audio that should be labelled as event is labelled as silence.

The dangers of this method of ground truth labelling is clear. There are however no good options that could provide all desired functionality. In order to get a score on how well the algorithm performs in detecting activity, there had to be some sort of truth. There are no machines or algorithms to do this automatically, which is the reason behind the thesis. Would an algorithm be used to label the audio automatically, the result would also be biased towards algorithms similar to that machine since they would function similarly. For the thesis, it is also desirable to use audio recorded with the camera in which the algorithm was implemented. It is therefore beneficial to avoid other labelled data found online since that would not reflect the microphone in the camera accurately. The best option appeared then to be to manually label the audio.

6.1.4 Optimisation

During the optimisation of the GMM, several contending parameter options were found. They had different measurement values and performed better in different categories. To write the result of every combination of hyper-parameters in the report is not possible. In the end, the GMM algorithm chosen was one with a slightly higher accuracy but with a slightly lower recall. For the camera implementation, a GUI is intended to give the option to chose a few settings regarding the algorithm. One intended option is how "paranoid" the algorithm should be. With this setting, the developers would not have to decide on one specific set of hyper-parameters. With increased recall and lowered accuracy, the algorithm would not be able to compress the audio as much as with the current setup, but would instead catch more of the events. This is then a trade-off the customer would have to take and not the developer. Different customers also have different preferences regarding the trade-off, therefore it is better to let them decide.

As can be seen in Table 5.3, the final GMM uses some interesting hyper-parameters. One of the interesting hyper parameters is σ_{comp} which is set to a value of 5.0. This

goes against what was used in the related work, where 2.5 was the common value to use. Why this value is used is simply because it yielded the best results from the grid search optimisation.

6.1.5 Time and resources

Another important factor to discuss is the limited time and resources. It should be noted that most of the time has been spent on implementing and improving the GMM, which partly is the reason that some of the results have been extracted only for that algorithm. Only a small amount of work has been put into development of the energy and VAD algorithms, which might be a reason for why the GMM appears better. The other algorithms do however not compete very well in complexity and modification ability. The energy function is very limited because of its simplicity while the VAD is complex algorithms that do not give much availability for modification. This is the prime reason that most of the time has been dedicated to the GMM algorithm. Another reason is that the GMM approach resulted in better initial results.

6.1.6 Data size

The aim of this project is to reduce the data size of encoded audio streams. The magnitude of reduction is discussed with the presumption that audio normally is encoded with a constant bitrate of 128 kbit/s. According to Table 5.9, there is a possibility to reduce the data size of the test files from 380.0 MB down to 176.2 MB if the activity detection is 100 % accurate. This is a reduction of 31.0 MB per hour of recorded material, but since the reduction depends on how much non-activity there is in the audio it is also interesting to mention a data size reduction per hour of non-activity. Since the difference between the high bitrate (128 kbit/s) and the low bitrate (14 kbit/s) is 114 kbit/s it means that there is a theoretical reduction rate of 51.3 MB per hour of non-activity. With this rate, a surveillance scene where there typically are 12 hours of activity and 12 hours of non-activity each day could save 615.6 MB of data each day per recording device.

With an instant bitrate decline on the test files that together contained 39.9 % activity, the resulting data size in the experiments was reduced from 57.9 to 25.8 MB per hour on average. This is a reduction rate of 32.1 MB per hour of recorded material in relation to constantly encode the audio with the high bitrate. One reason for the resulting data size being lower than the ideal data size is that the algorithm fails to detect some of the events in the test files. This results in that sequences that should have been encoded with 128 kbit/s instead get encoded with 14 kbit/s. This can be seen in Table 5.4 where test file 5 and 6 got rather poor values on recall. These two test files were the test files that had their noise levels amplified so bad results on these were expected. Another reason for the lower data size could be that the audio frames are encoded with VBR. When setting a bitrate with VBR, it is just a target bitrate which means that the bitrate could actually end up being lower or greater than the specified rate.

Gradual bitrate decline

When gradual decline is used instead of instant decline, the average bitrate increases which means that the data size reduction becomes lesser. The duration of the decline impacts how much the data will be reduced. A longer decline results in a lesser reduction which can be seen in Table 5.10. When a decline duration of 17.1 seconds was used the test files were reduced from 57.9 to 33.0 MB per hour, i.e. a reduction rate of 24.9 MB per hour. When the decline duration was increased to 34.2 seconds the test files were reduced from 57.9 to 35.5 MB per hour, i.e. a reduction rate of 22.4 MB per hour.

The decline has the advantage of providing a smooth transition from the high bitrate to the low. Besides the increase in average bitrate, a long duration on the gradual decline has the disadvantage that the bitrate could potentially rarely reach its lower limit in certain situations. One such situation is when multiple events are occurring shortly after each other. Another situation is if the activity detection is not functioning properly. If some frames in a silent sequence get falsely classified as activity shortly after each other, the bitrate will not have the time to adapt. This will result in the whole sequence being encoded with an unnecessary high bitrate and the activity detection will no longer be beneficial, see Figure 6.1 for an example. Such occurrences have been observed during testing, where just a couple of frames get wrongfully detected shortly after one another in a long sequence of silence.

6.1.7 Post event classification

The reason for the post event classification is that it helps detecting complete events by prolonging the classification of activity when an event gets detected. As can be seen in Figure 5.2, the recall increases with 19 percentage points just by keeping 10 frames after detected activity. The figure also shows that the accuracy peaks around 30 frames. The greater number of frames that are used the more will recall and accuracy drift apart from each other and eventually everything will get classified as activity. This can be seen in the figure, when a post event classification of 5000 frames is used, which is 5 minutes, the recall reaches 100 % and the accuracy reaches the same percentage as the portion of activity in the test files.

It could be argued that the gradual bitrate decline and the post event classification serve the same purpose. To some extent they do, for example are both of them algorithms that keep the bitrate higher than it otherwise should after an event has occurred. The reason behind them are however somewhat different. The gradual bitrate decline's main usage is to improve the listening experience with a gradual shift in quality. The main usage of the post event classification is instead to ensure that some unnoticed events get caught. An event is more likely to appear after another and with this implementation, the algorithm will perhaps catch otherwise unnoticed events.

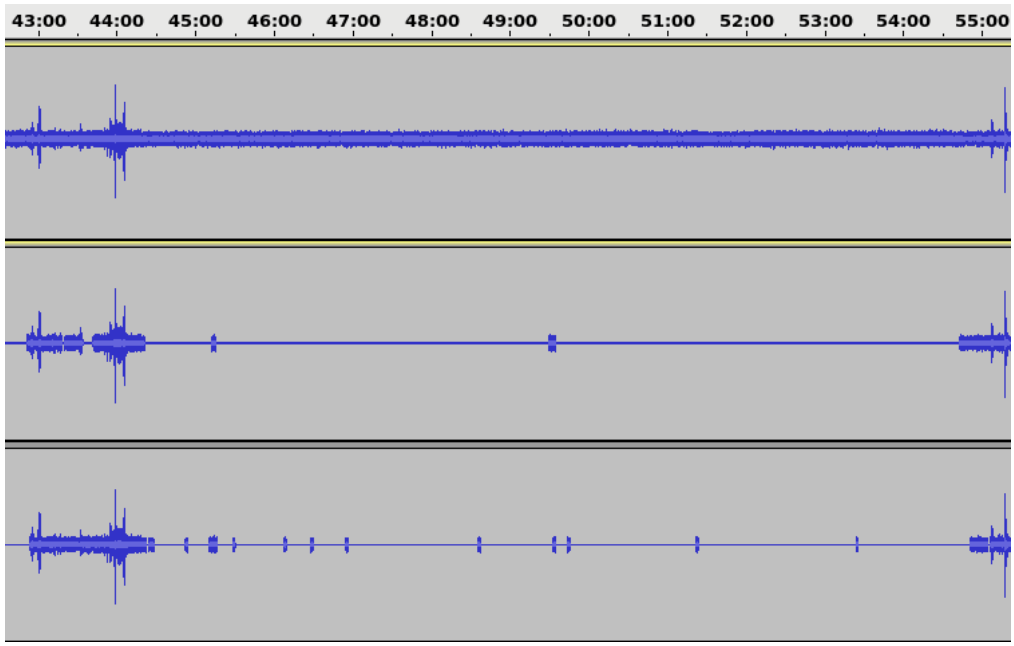


Figure 6.1: The first waveform shows a sequence from one of the test files. The second waveform is the corresponding ground truth and the third waveform is the resulting classification from the activity detection. The third waveform shows that the algorithm in some cases incorrectly detects activity. The parts that get falsely detected are very short so it is still possible to reduce the data size quite a lot if a quick decline in bitrate is used. However, if a long gradual decline was to be used the bitrate would rarely get low enough to make a major impact in data reduction since these spikes of false detection come shortly after each other and are repeatedly raising the bitrate.

6.2 Conclusions

The thesis is evaluated according to the problem formulation and concluded with ideas of future work.

The purpose of the thesis was to investigate possibilities to reduce the amount of data generated by audio in surveillance recordings and develop a program accordingly. This has been fulfilled. With the GMM algorithm, the data size was reduced from 57.9 to 26.8 MB per hour for the used test files with in total around 40 % activity and 60 % non-activity. Initially, several options of detecting audio events were investigated and three different approaches were implemented. With these algorithms, the data size of

the audio recordings is reduced.

The sub-problem of reducing the size without sacrificing important parts of the recording is partially achieved. While most of the audio that is compressed more heavily contain no event, there is no guarantee that some events considered important are certain to be kept at high bitrate. The main algorithm, the GMM approach, resulted in a recall of 81.05 %. Without the noisy test files, this number is higher. This means that 18.95 % of the time where an event is happening, the algorithm classifies the audio as background. What was considered an event in the test files might however not reflect an accurate description of an interesting event. Some of the events are for example mouse clicks and only in very few scenarios will that be considered as an interesting event. By being more sparse in the consideration of interesting events, the recall is increased even further. Since there are no real guarantee that the interesting audio is kept at high bitrate, the lowest bitrate that could be set is not zero but 14 kbit/s. At this bitrate, speech for example is still understandable.

Interesting audio as classified by the algorithm is compressed with a high bitrate according to the sub-problem of keeping interesting audio such as speech at a high quality.

The algorithm also work without any knowledge needed by the customer, which was desired as well. Instead, the algorithm functions indefinitely without any maintenance and the setup process only requires some silence during installation.

With this concluded, the problem is almost entirely solved with the exception of the demand that no interesting audio should be sacrificed. Without a perfect score of 100 % recall on a variety of test files, this cannot be guaranteed no matter the algorithm. Therefore, the assignment should be considered a success and to have solved the desired reachable goals.

6.2.1 Future work

This section concludes the report with some ideas of what could be done to further improve the algorithm or alternative ideas that were never tested because of time or resources.

Longer test file

It would be interesting to see how well the GMM algorithm performs on longer test files, for example recordings of weeks or months. As it is intended to be run on a camera, it would preferably work indefinitely. With limited time and resources, no longer test file was created since the labelling of such file would take a very long time.

With a longer test file, a test could be constructed to find out if for example shifts in background noise is handled correctly.

Silent startup

As mentioned in Section 4.1.3, the K-means initialisation requires a startup period when everything is classified as activity. During this time period, the high standard deviation for each of the subbands is calculated. This is not a problem in itself but if that time period contains foreground events rather than background events, the algorithm will be much more aggressive in its activity classification which can make some foreground events being interpreted as background. When this was realised, there was no time to fix the issue. One way of solving the problem would be to periodically reinitialise the high standard deviation for each of the subbands. Another way would be to slightly alter the high standard deviations over time with some sort of slow update method. Both these solutions would solve the issue. This would also solve the potential issue with altering backgrounds of long test files.

Gaussian mixture model augmentation

The implemented algorithm contained eight subbands where each of the subband contained an independent set of Gaussian distributions. As mentioned in Section 4.1.3, there exist an option to use a multidimensional GMM instead. In this alternative method, there would only be a single set of Gaussians and the subbands which in reality depends on each other would be modelled together. This approach has proven to give better results than to use several one dimensional Gaussian mixtures [18]. Although this was briefly tested, the algorithm gave poor results which was most likely due to an incorrect implementation. More experiments and implementation in this field of study are therefore encouraged.

The GMM could also be improved with more features. A feature mentioned but not used in the implemented algorithms is the ZCR. This feature is a good indication of speech. By using ZCR together with the PSD values of the sub-bands could yield an interesting result.

Training based classification algorithms

For this report, there were only time to implement a few different classification algorithms. With more time and resources, it would be interesting to see how for example the GMM algorithm would be compared to a method based of neural networks. It would also be interesting to see how well alternative methods such as HMM and SVM would classify events. With these methods, it could also be possible to classify what kind of event the recorded audio is, for example to determine if the event contained glass breaking or footsteps.

Alternative environments and microphones

Only two microphones with recordings from two different environments were tested for this report. This is mostly due to limitations in time and resources. How the GMM algo-

rithm would work with other microphones and with recordings from different environments would be interesting. For example could recordings from airports with constant activity show interesting results.

Outside compatibility

While the project early on was decided to only affect cameras placed indoors, a brief recording was once tested with a lot of wind background noise. The GMM algorithm labelled in this case almost every frame with wind as activity, and disregarded some frames with conversation as background noise.

An interesting thought would be to instead focus the development on handling of such wind noises to enable the placement of the microphone outdoors with still having an adaptive audio compression algorithm.

Bibliography

[1] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

[2] RG Bachu, S Kopparthi, B Adapa, and Buket D Barkana. Voiced/unvoiced decision for speech signals based on zero-crossing rate and energy. In *Advanced Techniques in Computing Sciences and Software Engineering*, pages 279–282. Springer, 2010.

[3] R Becker, G Corsetti, J Guedes Silveira, R Balbinot, and F Castello. A silence detection and suppression technique design for voice over ip systems. In *PACRIM. 2005 IEEE Pacific Rim Conference on Communications, Computers and signal Processing, 2005.*, pages 173–176. IEEE, 2005.

[4] Johannes Blömer and Kathrin Bujna. Simple methods for initializing the em algorithm for gaussian mixture models. *CoRR*, 2013.

[5] Selina Chu, Shrikanth Narayanan, and C-C Jay Kuo. A semi-supervised learning approach to online audio background detection. In *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 1629–1632. IEEE, 2009.

[6] Opus Codec. Comparison - opus codec. `https://opus-codec.org/comparison/`. Accessed: 2019-02-05.

[7] Dan Cossins. Q&a: Samuel morse invented the morse code, but how did he do it, how long did it take, and how long before it was accepted? *History Extra*,

2018. https://www.historyextra.com/period/victorian/qa-samuel-morse-invented-the-morse-code-but-how-did-he-do-it-how-long-did-it-take-and-how-long-before-it-was-accepted/.

[8] Marco Cristani, Manuele Bicego, and Vittorio Murino. On-line adaptive background modelling for audio surveillance. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, volume 2, pages 399–402. IEEE, 2004.

[9] Marco Crocco, Marco Cristani, Andrea Trucco, and Vittorio Murino. Audio surveillance: A systematic review. *ACM Computing Surveys (CSUR)*, 48(4):52, 2016.

[10] Matt Duff. Amplifier noise principles for the practical engineer. https://ez.analog.com/cfs-file/__key/telligent-evolution-components-attachments/00-630-01-00-00-04-96-33/AmplifierNoisePrinciplesforthePracticalEngineer.pdf, 2009. Accessed: 2019-05-27.

[11] The Physics Factbook. Frequency range of human hearing. https://hypertextbook.com/facts/2003/ChrisDAmbrose.shtml. Accessed: 2019-02-04.

[12] The Physics Factbook. The nature of sound. https://physics.info/sound/. Accessed: 2019-02-25.

[13] Issam Feki, Anis Ben Ammar, and Adel M Alimi. Audio stream analysis for environmental sound classification. In *2011 International Conference on Multimedia Computing and Systems*, pages 1–6. IEEE, 2011.

[14] Teofilo F Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

[15] Ruimin Hu, Bo Hang, Ye Ma, and Shi Dong. A bottom-up audio attention model for surveillance. In *2010 IEEE International Conference on Multimedia and Expo*, pages 564–567. IEEE, 2010.

[16] A ITU. A silence compression scheme for itu-t g.729 optimized for terminals conforming to itu-t v.70. *ITU-T Recommendation G*, 729, 1996.

[17] Sébastien Lecomte, Régis Lengellé, Cédric Richard, Francois Capman, and Bertrand Ravera. Abnormal events detection using unsupervised one-class svm-application to audio surveillance and evaluation. In *2011 8th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 124–129. IEEE, 2011.

[18] Simon Moncrieff, Svetha Venkatesh, and Geoff West. Online audio background determination for complex audio environments. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 3(2):8, 2007.

[19] Stavros Ntalampiras, Ilyas Potamitis, and Nikos Fakotakis. Probabilistic novelty detection for acoustic surveillance under real-world conditions. *IEEE Transactions on Multimedia*, 13(4):713–719, 2011.

[20] Vesa Peltonen, Juha Tuomi, Anssi Klapuri, Jyri Huopaniemi, and Timo Sorsa. Computational auditory scene recognition. In *2002 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 2, pages II–1941. IEEE, 2002.

[21] John G Proakis and Dimitris K. Manolakis. *Digital signal processing: principles algorithms and applications*. Pearson, 2014.

[22] Anssi Rämö and Henri Toukomaa. Voice quality characterization of ietf opus codec. In *Twelfth Annual Conference of the International Speech Communication Association*, 2011.

[23] David Reinsel, John Gantz, and John Rydning. The digitization of the world - from edge to core. Technical report, IDC, November 2018.

[24] Kirill Sakhnov, Ekaterina Verteletskaya, and Boris Simak. Approach for energy-based voice detector with adaptive scaling factor. *IAENG International Journal of Computer Science*, 36(4), 2009.

[25] Björn W Schuller. *Intelligent audio analysis*. Springer, 2013.

[26] Chris Stauffer and W Eric L Grimson. Adaptive background mixture models for real-time tracking. In *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149)*, volume 2, pages 246–252. IEEE, 1999.

[27] VoIP Supply. Cisco's hd voice solution overview. `https://www.voipsupply.com/cisco-hd-voice`. Accessed: 2019-02-25.

[28] J Valin, K Vos, T Terriberry, and A Moizard. Rfc 6716: Definition of the opus audio codec. *Internet engineering task force (IETF) standard*, 2012.

[29] Ben Waggoner. *Compression for great video and audio: Master tips and common sense*. Focal Press, 2010.

[30] WebRTC. Webrtc. `https://webrtc.org/`. Accessed: 2019-03-14.

[31] WebRTC. Webrtc vad source code. `https://webrtc.googlesource.com/src/+/refs/heads/master/common_audio/vad/vad_core.c`. Accessed: 2019-03-15.

[32] Wikipedia contributors. 4 bit linear pcm — Wikipedia, the free encyclopedia. https://commons.wikimedia.org/wiki/File:4-bit-linear-PCM.svg, 2019. Accessed: 2019-02-25.

[33] Wikipedia contributors. 44,100 hz — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/44,100_Hz, 2019. Accessed: 2019-02-25.

[34] Wikipedia contributors. Bartlett's method — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Bartlett%27s_method, 2019. Accessed: 2019-02-13.

[35] Wikipedia contributors. Data compression — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Data_compression, 2019. Accessed: 2019-02-04.

[36] Wikipedia contributors. Fast fourier transform — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Fast_Fourier_transform, 2019. Accessed 15-March-2019.

[37] Wikipedia contributors. Lossy compression — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Lossy_compression, 2019. Accessed: 2019-02-04.

[38] Xiph. Opusfaq. https://wiki.xiph.org/OpusFAQ. Accessed: 2019-02-21.

[39] Dong Zhang, Daniel Gatica-Perez, Samy Bengio, and Iain McCowan. Semi-supervised adapted hmms for unusual event detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 611–618. IEEE, 2005.