UNIVERSITY OF LEEDS – SCHOOL OF COMPUTING

Automatic audio segmentation based on spectral change

MSc AI – Final year project

James Marshall

2011- 2012

[The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others understand that failure to attribute material which is obtained from another source may be considered as plagiarism]

Abstract

This report presents a method to locate and extract audio thumbnails, which represent key features found within a piece of audio. The primary intended use for this system regards soundscape composition. The aim was to aid electroacoustic composers automatically find and extract key moments in an unedited recording.

The motivation for this project came from our human ability to be able to visually analyse a spectrogram image and locate the areas of interest. We first show how it is possible by using image analysis techniques; we can segment audio about areas of change. We then further develop this into a working prototype, which utilises spectral change as a feature vector. The report concludes with a series of tests, which show that our system successfully manages to extract segments from 10 separate tagged audio files with a high accuracy.

TABLE OF CONTENTS

TABEL OF FIGURES

TABLE OF TABLES

1. CHAPTER 1 – INTRODUCTION

1.1.MOTIVATION

Soundscape artists have spent many an hour trawling through hours, even days of recorded audio in order to find interesting snippets they can cut out to add to their soundscape. Throughout this project we propose a way of automatically finding these 'interesting' extracts.

The literature regarding non-speech sound recognition appears to be a lot less developed that of speech/vocal recognition, and seems to be clustered into a few key areas of research. These areas of research include security (Clavel, Ehrette, & Richard, 2005) (Gerosa & Valenzise, 2007) and music identification (Typke, Wiering, & Veltkamp, 2005) .The problem however, as (Dennis, Tran, & Li, 2011) points out is that most of these methods tend to use adaptations of techniques used in speech recognition systems. These methods tend not to be appropriate as they are focused towards a different goal.

The method we propose is inspired by techniques used in computer vision and our ability to automatically identify key areas of interest within a picture. It is possible using a time-frequency representation of an audio signal to obtain an image, which fully represents that signal, these images are known as spectrograms. We believe that by preforming image analysis on the spectrograms it will be possible to extract features, which we can use to train, a classifier in order to recognize interesting extracts from a large audio clip. While most previous extracted features have an acoustic motivation, if we look at them in their time-frequency domain interesting patterns occur in the visual domain. The below figure shows the waveform and spectrogram of a pop song, from this we can immediately identify the possible betas, note onsets and areas of the song which contain change. Although figure 1 shows a pop song the technique is still relevant within non note-based music as transitions of tone and timbre will still be present. We can therefore say that the visual information held within these spectrograms can provide a good source for feature extraction.

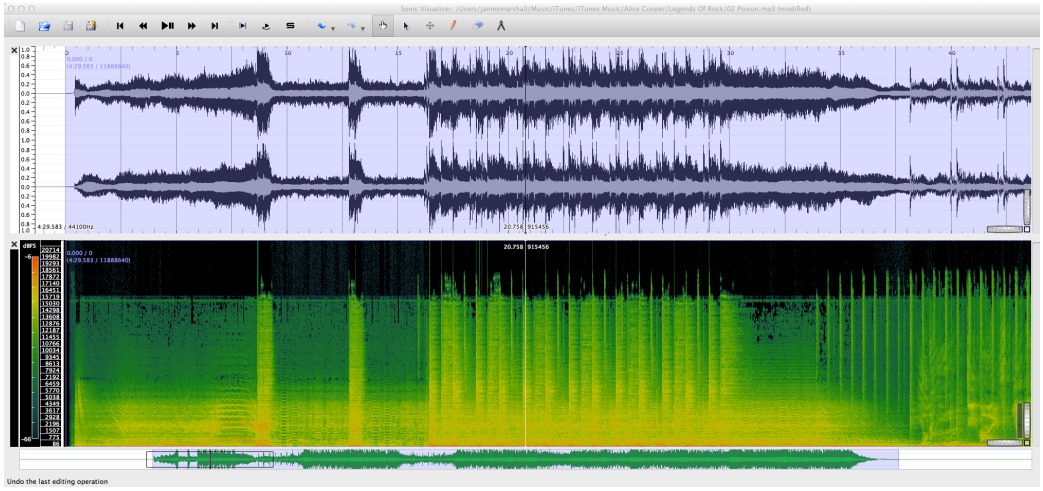


FIGURE 1 - WAVEFORM AND SPECTROGRAM OF A POP SONG

Although the intended use for this program will be to automatically pick out salient audio features to be used within soundscape composition. The idea could also be used to help with security, sound recognition, instrument identification and compression.

This project is relevant to the field of Artificial Intelligence (AI) as it draws from techniques used throughout many fields of AI including computational modelling, computer vision and machine learning. This system is designed to exhibit the intelligence of a human in such a way that it can automatically recognize interesting features of audio that a human would instinctively be able to pick out, saving time and hopefully helping us better understand how the human visual and auditory systems function.

1.2.OVERVIEW

1.2.1. AIM

The aim for this project is to analyse techniques to detect change within a sonic environment such as a soundscape. Once the change has been detected we will then extract representative segments in order to provide an audio summary of the different timbres present.

Possible applications for such project include producing an audio diary. One can imagine setting a microphone in a room of importance for a week and recording the day-to-day activity. Our system will be able to take that input and summarize the key events into a manageable amount.

It could also be useful in soundscape composition. Being able to extract the key components of a sonic environment could be an invaluable tool to an electroacoustic composer. It has the

potential of saving hours of work, and introducing a new composition element previously unavailable.

Another aim is to see if by analysing the images produced from spectrograms we can obtain results, which are comparable to, if we were to analyse the raw data that initially produced the spectrograms.

1.2.2. OBJECTIVES

From the aim of our project it is possible to highlight five key areas, which can correspond to our key objectives and minimum requirements these are listed below.

-Audio input handling

-Techniques to analyse characteristics of audio

-How to detect audio change

-Feature extraction

-Summarization of a given sound source by a lease of short sound segments

1.2.3. REQUIREMENTS

The minimum requirements for the project are as follows:

1) *Implement a means of handling audio data*

The program will require audio data in order to function; we therefore will need a means of inputting audio into our system. We will be initially using the software MATLAB for rapid prototyping and testing, we will therefore research, survey and test MATLAB's audio handling capabilities.

2) *Design and implement a spectra based technique to analyse a audio signal*

There are many techniques, which we will analyse however we will concentrate on a spectra based technique in order to analyse the audio signal. Again we will be using the software MATLAB to facilitate rapid prototyping and testing.

3) *Design and develop a technique to detect audio change*

In order to produce a summary of sonic environment we will need to develop a means of detecting audio change. During this section we will explore our interest of spectrograms to see if we can use these to automatically segment and detect audio differences.

4) *Extract the key features of a sound segment*

Once we have been able to detect a noticeable audio change we require a way of extracting the key features, which can accurately describe the sound. These features could include spectral envelopes, minimum/maximum frequencies, zero-crossing boundaries or harmonic components.

5) *Produce a summarisation of an audio track using short sound segments.*

Finally we are required to produce a summarisation of the given audio track. The summarisation will comprise of a list of short audio segments, which can accurately represent the sonic environment using the minimum amount of segments.

1.2.4. ENHANCEMENTS

A few potential enhancements are listed below.

- *Automatically produce a composition based on the extracted material*

Once we have extracted the summarised audio, it may be possible to auto generate a electroacoustic soundscape composition. It could allow the user to tune certain parameters, add effects and automatically apply digital signal processing techniques. The result would certainly be interesting and could add a whole new dimension to electroacoustic composition,

- *Look at ways of making sure we do not record the same material more than once (similarity measure).*

A problem I foresee is that once we have distinguished a boundary between two sounds, how are we going to know if that sound then re appears later on? For example given the sequence ABCA we might wish to return A B C however return A A B C. To stop this we might be able to use a measure of similarity or store the results to compare with any newly recognised segments.

- *Create a user interface to allow the user to easily access extracted segments.*

A UI can really improve the usability of the system; it can help turn an idea into a potential product. A UI may also be able to help during the validation stage providing us a means to easily compare segments.

1.2.5. DELIVERABLES

The deliverables for this project will contain a final written report and a data CD containing a demonstration of the system. The audio files on the data CD will show the system results for a number of audio clips used in the validation.

1.3.METHODOLOGY + SCHEDULE

1.3.1. PROJECT METHODOLOGY

The project assumes an iterative and incremental development approach (IID). The idea behind this method is to develop a system through repeated cycles of planning, analysis, implementation and evaluation (figure 2). The alternative strategy is to plan everything from the outset and develop the entire system with a 'big –bang' integration at the end (Cockburn, 2008).

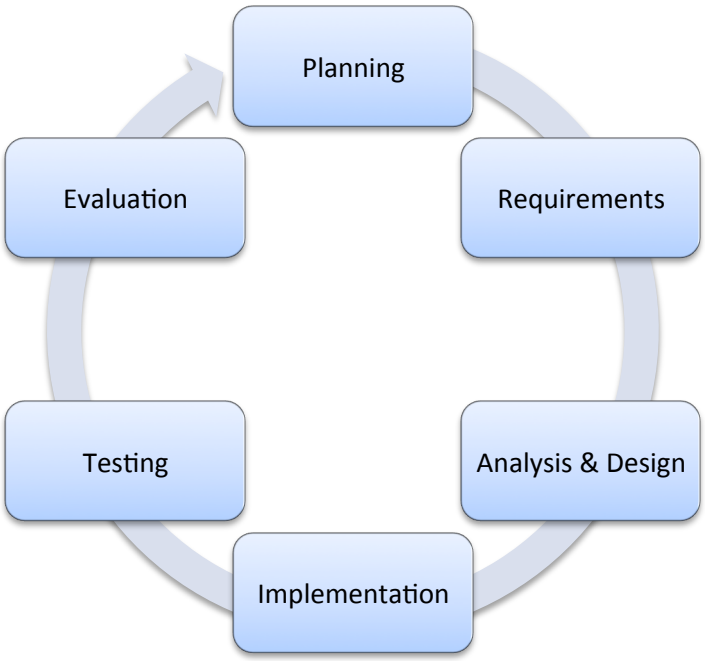


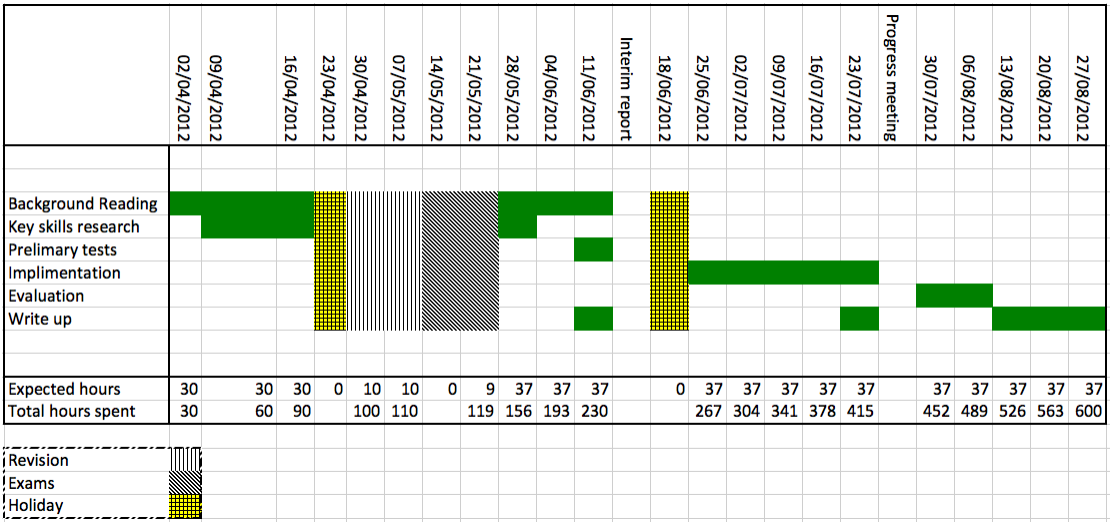
FIGURE 2 - IID METHODOLOGY

Another reason we have opted for the IID approach is, for our software development we face

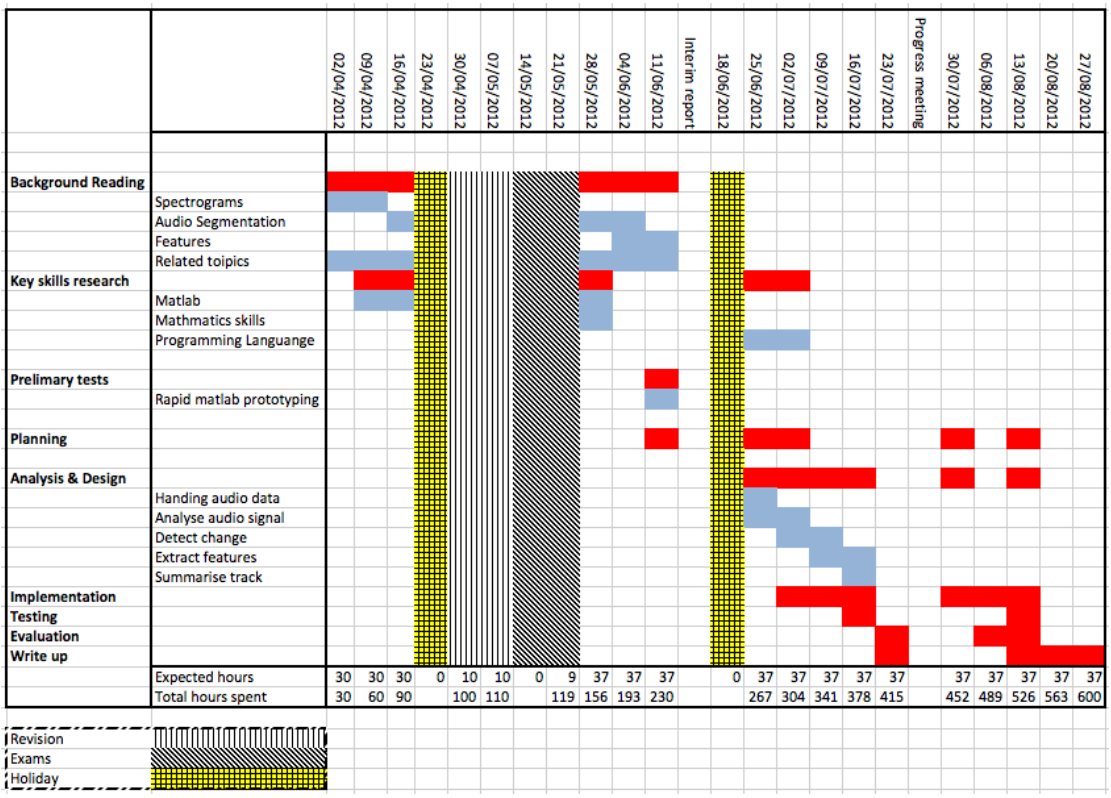
uncertainties and un-predictable changes, the literature recommends the use of iterative and incremental development as it enables fast reaction to change (Larman, 2003).

1.3.2. SCHEDULE

The initial schedule for the project is given below:



As you can see this gives a brief overview though detail needed to be added. Below shows a revised schedule based on the IID methodology and project aims.



Following on from a project meeting, a final detailed schedule was created (shown below).

		Progress meeting	30/07/2012	06/08/2012	13/08/2012	20/08/2012	27/08/2012
Planning	Look at mean-shift alogorithm						
Analysis & Design							
	Extract audio						
	Validation - Create and impliment a validation test						
	General Bug fix						
	Create Basic UI using HTML						
Implementation							
Testing	Obtain results for the validation test in week 13th						
Evaluation							
Write up	Aim to have first draft for 23rd						
	Printing and binding						
	Expected hours		37	37	37	37	37
	Total hours spent		452	489	526	563	600

2. CHAPTER 2 – RELATED BACKGROUND

2.1.SPECTROGRAM AND AUDIO CLASSIFICATION

We started the background following on from our motivation of using spectrograms as a means of audio classification. This lead us to a very limited supply of literature. (Deshpande, Singh, & Nam, 2001) seems to be one of the first references to audio classification in the visual domain. Deshpande attempts to automatically classify music into three broad categories: rock, classical and jazz. They do this by extracting features from spectrograms and use texture-of-texture models to generate feature vectors. They attempt to classify new sounds using a variety of classifiers including support vector machines and k nearest neighbour.

During the data preparation stage they use the MP3 compression standard to pre-process the data. By doing this it significantly reduces the file size and removes a lot of redundant information. Although MP3 compression is lossy (permanently removes data), it is generally considered that MP3 compression preservers the perceptual quality of the music. Therefore only the irrelevant features are removed. This said it could be the case that although it is perceptually similar to untrained ears actually the removed data might provide extra useful features in order to help us extract the salient points. The advancement in computing power over the past decade could provide us with a way to train and use the raw WAV file.

(Deshpande, Singh, & Nam, 2001) achieved reasonably good results however, there is a sever lack of training and test data and as such they found that a few misclassified results skewed their results. The other problem they found is that they were only using a short except of each training instance, which was randomly selected out of the track. It could have been the case that

although the track is generally a rock song there might be a small orchestral section, which the random sample could have picked. This of course might lead to a misclassification even though the algorithm is preforming well. Other authors also note (Yu & Slotine, 2009) that the recursive-filtering algorithm that they apply does not seem to fully capture the texture like properties of audio signals in the time frequency representation and therefore limits performance.

(Yu & Slotine, 2009) use a simple block matching technique on a grey-scale spectrogram in order to classify sounds. They achieve this by treating the sound spectrograms as texture images. The results they obtain appear to be extremely high obtaining an average accuracy over 8 classes of 87.5 percent. Although Yu and Slotine's algorithm worked successfully with solo instruments it would be interesting to see how the algorithm preformed with 'real life' sounds such as bird song, traffic or weather. The other possible problem is the introduction of polyphonic sounds. Currently their system was only tested on monophonic instruments and although they used instruments from the same family (violin, cello) to avoid too much over simplification there was little noise. It would be interesting to see by how much and if, the results degrade when tested with a noisy signal.

The most recent occurrence of using spectrograms for classification tasks come from the work of (Costa, Oliveira, Koerich, & Gouyon, 2011) and (Dennis, Tran, & Li, 2011). Both of these authors motivation again comes from the fact that spectrograms can produce recognisable images, which can be identified by a human reader. (Dennis, Tran, & Li, 2011) method concentrates on trying to make the classification robust to noise. They first map the spectrogram into a higher dimensional space by quantizing the dynamic range into different regions. This is very similar to the colour mapping used in image processing where they assign different intensity's of a grey-scale image RGB values. Their system then extracts the central moments of the partitioned monochrome intensity distributions as features. This produces a noise robust feature, as the spectrogram of the signal is sparse. The diffuse noise intensity is mostly located in the lower regions of the spectrograms dynamic range and the higher regions remain dominated by the strongest sound component.

(Costa, Oliveira, Koerich, & Gouyon, 2011) on the other hand present a paper, which use spectrograms in order to classify musical genre. Their approach uses a 'zoning' mechanism to preform local feature extraction. They noticed that by analysing the spectrogram images, the textures are non-uniform so instead of preforming a global feature extraction it is more relevant to preform a local one. This is accomplished by using a 'zoning' technique, which simply splits the spectrogram into, sections (figure 3).

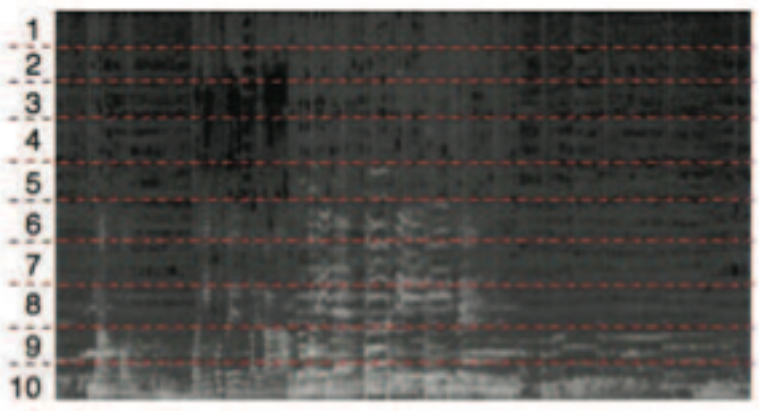
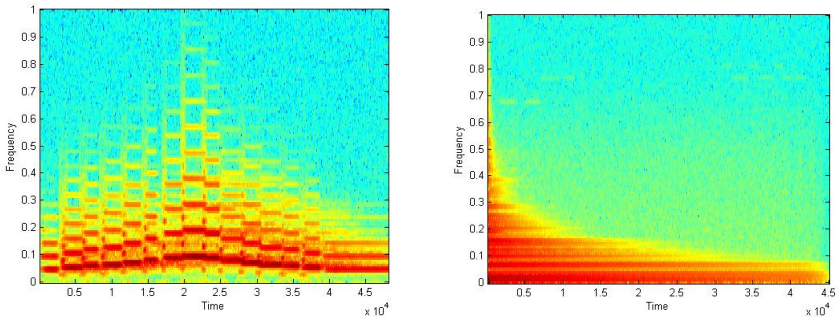


FIGURE 3 - ZONING TECHNIQUE (COSTA, OLIVEIRA, KOERICH, & GOUYON, 2011)

The features, which (Costa, Oliveira, Koerich, & Gouyon, 2011) use, are the Gray-Level Co-occurrence Matrices (GLCM) texture descriptors. They claim that statistical techniques of texture recognition the GLCM have been one of the most used and successful. The techniques they use appear to be very successful and they note that if they also combine features extracted from low level characteristics of the music they obtain an average accuracy increase of 7%.

It appears that other authors have had the same motivation and have used different techniques in order to classify sounds. However our original motivation was shown in beat tracking and not classification. To test the validity of (Costa, Oliveira, Koerich, & Gouyon, 2011) claims we decided to produce spectrograms using MATLAB of 4 different classes of instrument. The purpose of this test was not only to validate their motivation but to also experiment with the audio handling capabilities of MATLAB.



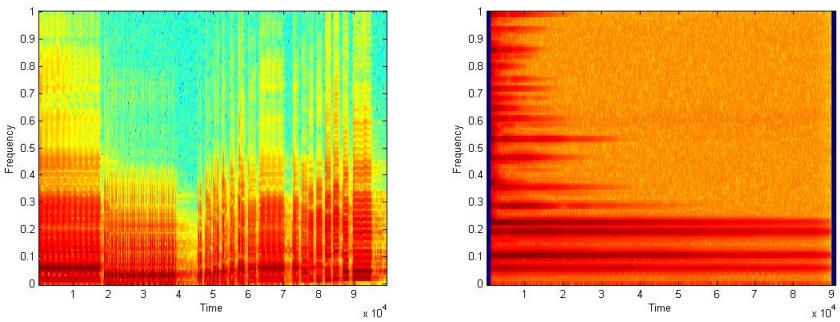


FIGURE 4 – CLOCKWISE FROM LEFT SPECTROGRAM OF FLUTE, GUITAR, TENOR SAX & TRIANGLE

Figure 4 shows that we can see that (Yu & Slotine, 2009), (Deshpande, Singh, & Nam, 2001), (Dennis, Tran, & Li, 2011) and (Costa, Oliveira, Koerich, & Gouyon, 2011) where correct and the texture images provided by the time frequency representation appear to provide us with a means of visually separating sounds.

2.2.AUDIO SEGMENTATION + FEATURES

Audio segmentation aims at extracting information and audio change from within a given audio excerpt. If we were looking at a 'pop' song this could be the song's structure, i.e. *verse, chorus, bridge* etc. However if we where looking at a soundscape recording this could be an area where something happens, e.g. a *police siren driving past.* By understanding the ways authors approach this and the features they use we hope to gain a better understanding how we can apply their techniques to our problem.

(Peiszer, Automatic audio segmentation: Segment boundary and structure detection in popular music, 2007) conducted similar research and compiled a table listing feature sets, relevant papers and the subtasks associated with these papers over the past decade. This table can be seen below with a description of the features in the following section. Note that the references for Peiszers table are presented in the same order as figure 5.

(Rhodes, Casey, Abdallah, & Sandler, 2006)
(Peiszer, Automatic audio segmentation:Algorithm, experiments and evaluation, 2007)
(Paulus & Klapuri, 2006)
(Ong, 2005)
(Maddage, Xu, Kankanhalli, & Shao, 2004)
(Lu, Wang, & Zhang, 2006)
(Logan & Chu, 2000)
(Levy, Sandler, & Casey, 2006)
(Goto, 2003)
(Foote, Automatic audio segmentation using a measure of audio novelty, 2000)

(Cooper & Foote, 2003)
(Chai, 2005)
(Casey & Slaney, 2006)
(Bartsch & Wakefield, Audio thumbnailing of popular music using chroma-based representations , 2005)
(Bartsch & Wakefield, To catch a chorus: using chroma-based representations for audiothumbnailing , 2001)
(Aucouturier & Sandler)
(Abdallah, Noland, Sandler, Casey, & Rhodes, 2005)

TABLE 1 - REFERENCES FOR PEISZERS TABEL

Feature sets	[ANS+05]	[AS01]	[BW01]	[BW05]	[CS06]	[Cha05]	[CF03, FC03]	[Foo00]	[Got03]	[LSC06]	[LC00]	[LWZ04]	[MXKS04, Mad06]	[Ong05]	[PK06]	[PBR02]	[Pei07]	[RCAS06]
MFCC		✓									✓		✓	✓	✓		✓	
CQT	✓									✓		✓			✓		✓	✓
Chromagram			✓	✓	✓	✓			✓				✓					
Other		✓[a]			✓[b]			✓[c]					✓[d]			✓[e]	✓[c, f]	
Subtasks																		
Segmentation	✓	✓				✓	✓	✓		✓		✓	✓	✓	✓	✓	✓	✓
Recurrent structure, musical form	✓	✓			✓[g]	✓	✓			✓		✓[g]	✓		✓[g]	✓	✓	✓
Audio summary				✓		✓	✓					✓				✓		
Key phrase / chorus detection			✓						✓		✓							
Semantic labels												✓	✓					
Vocal / instrumental section detection												✓	✓					

[a] Linear prediction of spectral envelope; Discrete Cepstrum
[b] Log-Frequency Cepstral Coefficients
[c] Spectrogram
[d] Octave Scale Cepstral Coefficients
[e] "dynamic features"
[f] Rhythm Patterns, Statistical Spectrum Descriptors
[g] concentrate on repeated segments, parts of songs may be left unexplained

FIGURE 5 - OVERVIEW OF FEATURE SETS AND SUBTASKS USED IN INDIVIDUAL PAPERS (PEISZER, AUTOMATIC AUDIO SEGMENTATION: SEGMENT BOUNDARY AND STRUCTURE DETECTION IN POPULAR MUSIC, 2007)

From this table (figure 5) we can see that Pieszer only found two papers that used spectrograms as their feature sets (Foote, Automatic audio segmentation using a measure of audio novelty, 2000) & (Peiszer, Automatic audio segmentation:Algorithm, experiments and evaluation, 2007) one of which was the authors masters thesis which used the techniques in Foote's paper.

2.2.1. MFCC - TIMBRE RELEATED FEATURES

Pieszer claims that many authors do not rely on the ubiquitous Mel-Frequency Cepstrum Coefficients (MFCCs) in the task of audio segmentation, as they are well known to describe the timbre content. This of course is a problem in popular music for example if we look at a generic rock song, the chorus could consist of a distorted guitar riff and another riff lacking a guitar, therefore as these sounds differ a lot, a problem could be encountered. However (Ong, 2005) claims that the MFCC method is particularly useful for analysing complex music due to its low dimensionality smooth version of the log spectrum and ability to discriminate between different spectral contents. MFCC calculations can be done via the following steps (Rabiner & Juang, 1993);

1. Convert signal into short frames

2. Compute discrete Fourier transform of each frame

3. The spectrum is converted to the log scale

4. Mel scaling and smoothing the log scale spectrum

5. Discrete cosine transform is calculated (to reduce the spectrum to 40 coefficients)

2.2.2. CONSTANT Q TRANSFORM (CQT)

The CQT transform is used to transform a data series to the frequency domain similar to that of the Fourier transform. It can however be used to map frequency to the western 12 tone scale (Peiszer, Automatic audio segmentation: Segment boundary and structure detection in popular music, 2007) e.g. mapping the frequency 440Hz to the note A. CQT refers to a time-frequency representation where the frequency bins are geometrically spaced and the Q-factors (which are the ratios of the centre frequencies to bandwidths of all bins) are equal (Schoerkhuber & Klapuri, 2010).

2.2.3. CHROMAGRAM

Chromagram, which is also known as 'Pitch class Profile' (Goto, 2003) can be thought of as a generalized CQT. The chromagram is defined as the restructuring of a spectral representation in which the frequencies are mapped onto a limited set of 12 chroma values in a many-to-one fashion (Pauws, 2004). Chromagram is specifically employed for musical signals; it combines the frequency components belonging to the same pitch class and results in a 12-dimensional representation, these correspond to the western 12-tone scale (Chai, 2005). As the chromagram is specifically designed for musical signals it may not be appropriate for us to employ it when dealing with soundscape environments. As we can see from figure 5 chroma based features appear frequently throughout the literature. (Goto, 2003) & (Bartsch & Wakefield, To catch a chorus: using chroma-based representations for audiothumbnailing , 2001) use similar methods to analyse relationships between repeated sections of popular song detecting boundaries between verse and chorus.

2.3.OTHER TIMBRE RELATED FEATURES

Many timbre-related features have been proposed for analysing audio however (Ong, 2005) claims that the most employed features are as follows:

Zero crossings – A measure of the number of times the sign of a signal changes. For example in figure 6 there is one zero crossing point at the point where the sine wave crosses the X access.

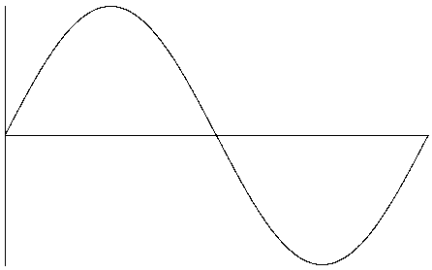


FIGURE 6 - SINE WAVE

Spectral Centroid - The spectral centroid indicates where the "centre of mass" of the spectrum is. (Grey & Gordan, 1978) claim that perceptually, it has a robust connection with the impression of "brightness" of a sound. It is computed as follows (Ong, 2005):

$$Spectral\ Centroid = \frac{\sum_K kX[k]}{\sum_k X[k]} \qquad (1)$$

where k is a correspond index to a frequency bin, within the overall measure spectrum, and X[k] is the amplitude of the corresponding frequency bin.

Spectral Rolloff – The spectral roll-off point is the frequency so that 95% of the signal energy is contained below this frequency (Peeters, 2004). (Ong, 2005) claims it is a measure of the 'skewness' of the spectral shape where the value is higher for right-skewed distributions. Spectral rolloff is calculated by:

$$Spectral\ Rolloff = 0.95 \times \sum_k X[k] \qquad (2)$$

Spectral Flux – Characterises the shape changes of the spectrum. It is a measure of spectral difference (Ong, 2005). It is calculated as the 2-norm between two normalised spectra and given by (Ong, 2005):

$$Spectral\ Flux = \|X[k] - X[k-1]\| \qquad (3)$$

where *X[k] is the spectral magnitude of a frame.*

2.4.DYNAMIC RELEATED FEATURES

2.4.1. SPECTRAL POWER

(Xu, Zhu, & Tian, 2002) present an algorithm for automatic music summarisation, which uses spectral power, amplitude envelopes and MFCC as features. Xu et al. first weight each frame with a Hann window h(n) (Xu, Zhu, & Tian, 2002):

$$h(n) = \frac{\sqrt{8/3}}{2}\left[1 - \cos\left(2\pi \frac{n}{N}\right)\right] \qquad (4)$$

where *N is the number of samples of each frame.*

The spectral power S(k), is then calculated as (Xu, Zhu, & Tian, 2002):

$$S(k) = 10 \log_{10}\left[\frac{1}{N}\left\|\sum_{n=0}^{N-1} s(n)h(n)\exp\left(-j2\pi \frac{nk}{N}\right)\right\|^2\right] \qquad (5)$$

2.4.2. RMS ENERGY

(Tzanetakis & Cook, 1999) use the root mean square (RMS) to produce features. The RMS is a measure of the loudness of the frame. They claim that RMS is important as new sound cue are often given by changes of loudness. It is given by:

$$RMS = \sqrt{\frac{1}{N}\sum_{k=0}^{N-1} x[k]^2} \qquad (6)$$

where N is the number of samples in each frame.

2.4.3. AMPLITUDE ENVELOPE

The amplitude envelope describes the energy change of the signal in the time domain and is generally equivalent to the attack, decay, sustain, release of a musical sound (Xu, Zhu, & Tian, 2002). Xu et al. compute the envelop (Xu, Zhu, & Tian, 2002)e using a frame-by-frame RMS as seen above and a low 3rd order Butterworth lowpass filter.

2.5. AUDIO SIMILARITY

From Pieszer's table we can see that Foote has used spectrograms as a means of obtaining features. We therefore decided to further investigate the paper. Foote describes methods, which can automatically locate points within audio that have significant change. They go about this be analysing local self-similarity. Foote claims that their method can find individual note boundaries or natural segment boundaries such as speech/music transitions and demonstrates multiple applications such as indexing, segmenting and beat tracking.

Foote's method computes the self-similarity for the past and future at each instant. Then a significantly novel point will have a high self-similarity in the past and future and low cross-similarity. The author notes that the notion of past and future can be changed depending on the task at hand. For example if we were looking to distinguish notes a shorter timeframe would be used compared to if we wanted to distinguish between verse and chorus.

Audio novelty can be represented by (Foote, Automatic audio segmentation using a measure of audio novelty, 2000):

$$N(i) = \sum_{-L/2}^{L/2} \sum_{-L/2}^{L/2} C(m,n) S(i+m, i+n) \qquad (7)$$

where *i is the frame number and L represents the width of the kernel, which is centred on 0,0.*

Once a novelty has been calculated for a piece of audio a graph can be made showing the novelty score against time. It is then possible to set a local or global threshold to extract segment boundaries.

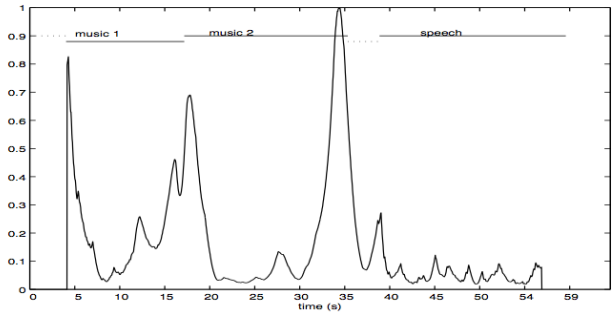


FIGURE 7- NOVELTY SCORE (FOOTE, AUTOMATIC AUDIO SEGMENTATION USING A MEASURE OF AUDIO NOVELTY, 2000)

Figure 7 demonstrates how the novelty score can be used to separate a piece of audio into sections and highlight areas of change. The graph shows the first minute of "animals have young" and it clearly shows three areas here of distinctive change, which Foote has labelled. Note that as Foote's approach uses the signal to model itself it does not require any training. Other authors have since built on Foote's work (Bartsch & Wakefield, To catch a chorus: using chroma-based representations for audiothumbnailing , 2001) built on the idea of a similarity matrix to create a 'time-lag' matrix whereby it was easier to extract repeating segments. Other authors such as (Ong, 2005) and (Chai, 2005) have also used the idea of a similarity matrix to improve their algorithms.

Foote's method could be extremely useful for us as it provides a way of distinguishing between old and new content. For example if we have content A-B-A the algorithm may separate that into three groups however we need a way of storing that we have previously seen A.

2.5.1. AUDIO SUMMARIZATION

(Cooper & Foote, 2002) present a method to automatically provide summary excerpts or thumbnails of a piece of audio. They do this by calculating the similarity of a parameterised piece of music, store these results in a 2-D matrix then simply sum the vectors. This gives a score of how similar the segment is to the whole piece of music.

Consider a simple audio file containing the following form: A (30 seconds) B (40 seconds) C (30 seconds).

First the signal is parameterized; Cooper and Foote use the MFCC analysis to do this. In (Foote, Visualizing music and audio using self-similarity, 1999), they argue that MFCCs have been shown to be better than spectral, pitch, and zero-crossing measures for discriminating between speech and music. However they do say that it can be argued for using MFCCs for music analysis as the MFCC discards pitch information as they contain the fine harmonic stricture

characteristics of the driving function (Foote, Visualizing music and audio using self-similarity, 1999). However MFCCs can be looked at as a smoothed representation of a sound's frequency spectrum, thus match similar timbres as a pose to exact pitches.

Once the signal has been parameterised a measure of dis-similarity can be measured by computing a distance matrix between pairs of parameter vectors *vi* and *vj*. (Cooper & Foote, 2002) propose two ways of doing this; one simple way is to use the Euclidean distance however a way they claim is more effective and produces a larger similarity score even if the vectors are small in magnitude is to calculate the scalar dot product of the vectors, which can then be normalized to give the cosine of the angle between the parameter vectors (Cooper & Foote, 2002):

$$d_c(vi, vj) = \frac{<vi,vj>}{||vi||\ ||vj||} \qquad (8)$$

The matrix S contains the similarity computed between all possible frame combinations such that:

$$S(i,j) = d_c(vi, vj)$$

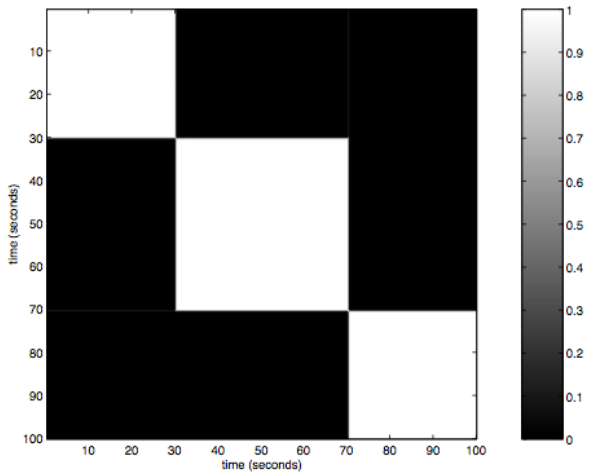


FIGURE 8 - SIMILARITY MATRIX

To visualize the similarity matrix a brightness proportional to the similarity $d_c(i,j)$ to each pixel (i,j) is mapped. This is shown in figure 8 where white is most similar and black is not similar. Note that the diagonal will always have the maximum values, as these regions are most similar to themselves. If we look at the diagram above we can see that during the first 30 seconds the same section is being played then it moves to a different section for the next 40 seconds then finally moves to a third section for the final 30 seconds.

2.5.2. CALCULATING THE AUTOMATIC SUMMARIZATION

For example given the sequence ABBBCC (assuming each letter has length one) and trying to find the most representative section of length three.

The similarity matrix can be created. Note, for simplicity Cooper and Foote chose the similarity measure to be one if the sequence matches zero.

$$S = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} & \boldsymbol{B} & \boldsymbol{B} & \boldsymbol{C} & \boldsymbol{C} & \\ 1 & 0 & 0 & 0 & 0 & 0 & \boldsymbol{A} \\ 0 & 1 & 1 & 1 & 0 & 0 & \boldsymbol{B} \\ 0 & 1 & 1 & 1 & 0 & 0 & \boldsymbol{B} \\ 0 & 1 & 1 & 1 & 0 & 0 & \boldsymbol{B} \\ 0 & 0 & 0 & 0 & 1 & 1 & \boldsymbol{C} \\ 0 & 0 & 0 & 0 & 1 & 1 & \boldsymbol{C} \end{bmatrix} \qquad (9)$$

Now for any subsequence, the average similarity between subsequences and the entire sequence can be found by summing each column. For example if we wanted to find a subsequence of length 3, the possible subsequences and their respective scores would be:

ABB = (1+3+3) = 7

BBB = (3+3+3) = 9

BBC = (3+3+2) = 8

BCC = (3+2+2)= 7

Therefore BBB would be the segment, which most represents this sequence of audio.

To generalize this to a s sequence of any given length starting at q and ending at r, Cooper and Foote given the following formula:

$$\bar{S}(q,r) = \frac{1}{N(r-q)} \sum_{m=q}^{r} \sum_{n=1}^{N} S(m,n) \qquad (10)$$

where *N is the length of the entire work (width and height of S).*

3. CHAPTER 3 – DESIGN AND DEVELOPMENT - PRELIMINARY

3.1.SOUND & ITS DIGITAL REPRESENTATION

Before we can start to look at developing complicated programs and high level technologies which manipulate and analyze sound we first have to go right back to examine what exactly sound is and how it is digitally represented. This section we will briefly describe the fundamentals of acoustics and then go onto describe how we can digitally represent sound.

3.1.1. NATURE AND CHARACTERISTICS OF A SOUNDWAVE

At a fundamental physical level, sound is simply a disturbance of molecules within a substance. This could be gas, solids or even liquids. However we need to look a lot deeper if we want to obtain a useful understanding of sound.

The best way to describe the nature of sound waves is to consider a model:

FIGURE 9 - SPRING MODEL OF SOUND PROPAGATION – (HOWARD & ANGUS, 2007)

The spring model descried by (Howard & Angus, 2007) shows a simple single dimensional model of a substance in which a sound wave could propagate. The golf balls in figure 9 represent the molecules/point masses within the substance and are connected by a series of springs representing the molecular forces. Using this model we can see that if the molecule and one end of the spring is excited, the springs next to it will first compress causing the next molecule to move. Within this model the region where the springs are pushed together is known as compressed and when the springs are being pulled as rarefaction. Sound waves therefore propagate by a series of compressions and rarefactions as longitudinal waves.

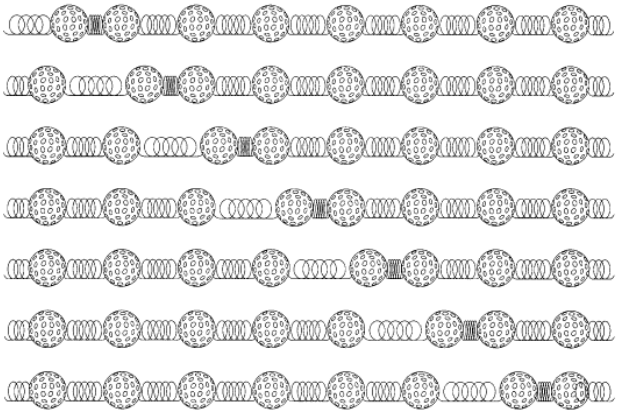


FIGURE 10 - COMPRESSIONS & RAREFACTIONS (HOWARD & ANGUS, 2007)

We can define three main characterises of sound waves which we can use to completely describe or reconstruct a signal. These features are described below:

Amplitude: The amplitude can be thought of as the amount of compression and rarefaction. This amplitude is directly related to loudness. (Rumsey & McCormick, 2006)

Frequency: The frequency of sound is determined by the rate, at which a sound source oscillates and is represented by the unit Hz. The frequency directly relates to the pitch of a sound. The higher the frequency the higher the pitch.

Wavelength: The wavelength refers to the distance between two adjacent peaks of compression or rarefaction. The wavelength is dependent on the speed of sound and its frequency. We can therefore define the wavelength as:

$$\lambda = v/f \qquad (11)$$

Where f= frequency (Hz) and v =velocity of sound (m/s)

3.1.2. DIGITAL REPRESENTATION

So far we have considered sound from its very fundamental physical properties, however we are concerned with sound in a digital environment, as such we need to understand how sound is converted and stored as a series of bits.

The main device used in digital recording is an Analogue-to-Digital Converter (ADC). The ADC captures many snapshots/samples of the electric voltage of the audio signal using a transducer most commonly the microphone. It is then represented as a digital number that can be sent and stored on a computer. By capturing this voltage thousands of times per second, you are able to get a very good approximation to the original audio signal:

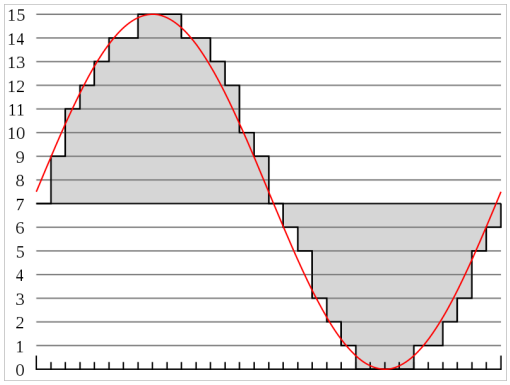


FIGURE 11- CONVERTING A SOUND FROM ANALOGUE TO DIGITIAL (HTTP://EN.WIKIPEDIA.ORG/WIKI/FILE:PCM.SVG)

Figure 11 shows a audio wave which has been sampled using a 4-bit ADC and as such has 16 possible levels. The red line represents the original signal and the black line the digitised version. From this diagram we can see that there are two major factors, which affect the quality and representation of the digital wave:

Sample rate: The rate at which the samples are captured or played back, measured in Hertz (Hz), or samples per second. A typical audio CD uses a sample rate of 44,100 Hz.

Sample format or **sample size**: The number of digits in the digital representation of each sample. An audio CD has a precision of 16 bits allowing a total number of 65536 levels.

The higher the sampling rate the more accurately we are able to record higher frequencies of sound. The sampling rate should be at least twice the highest frequency you want to represent otherwise aliasing problems can occur. As the range of human hearing is roughly between 20-20,000 Hz (Olson, 1967), 44,100 Hz was chosen as the rate for audio CDs to include all human frequencies.

The higher the sample size the more accurate the digital representation will be, however as we increase the sample size the size of the audio file will also increase. There is therefore a trade-off between sample rate, sample size and audio quality. The following table shows the size effects of increasing these components.

File Type	44.1 KHz	22.05 KHz	11.025 KHz
16 Bit Stereo	10.1 Mb	5.05 Mb	2.52 Mb
16 Bit Mono	5.05 Mb	2.52 Mb	1.26 Mb
8 Bit Mono	2.52 Mb	1.26 Mb	630 Kb

FIGURE 12 - AFFECTS OF INCREASING SAMPLE RATE AND SIZE (MARSHALL, 2001)

For the digital sound to be re-created into sound, the process described above is reversed, and the signal is then amplified and then sent to a loudspeaker that converts the voltage representation back into physical sound waves.

3.2. MATLAB + AUDIO

Throughout this project we are required to produce and create rapid prototypes of systems in order to test our hypotheses. MATLAB seems to be the obvious choice as it is a widely used environment for signal processing and analysis and offers many tools and toolboxes, which we can, utilize. However in order to confirm this a series of tests and experiments will be conducted to test the audio handling capabilities of MATLAB.

3.2.1. AUDIO INPUT + OUTPUT

Throughout the project we are going to be required to input and output audio files. We found that getting audio in and out of MATLAB was extremely simple. MATLAB provided simple functions, which could be called throughout scripts that handled most of our basic requirements. We have provided a list of the most basic functions below:

wavfinfo - Provides information about Microsoft WAVE (.wav) sound file such as sample rate and number of channels .

wavplay - Play recorded sound on PC-based audio output device.

wavread - Read Microsoft WAVE (.wav) sound file. This is probably one of the most useful to us as it allows us to import sound files. It can also return crucial information such as the sample rate and bits per sample. An example of a wavform imported using this method is shown in figure 13.

wavrecord - Record sound using PC-based audio input device.

wavwrite - Write Microsoft WAVE (.wav) sound file. Required when we are exporting the audio segments out of our system. It requires data to be stored in an array, as well as a sample rate

and bit depth. It is possible to resample a waveform using this function by changing the sample rate. Figure 14 shows an example of resampling the original waveform.

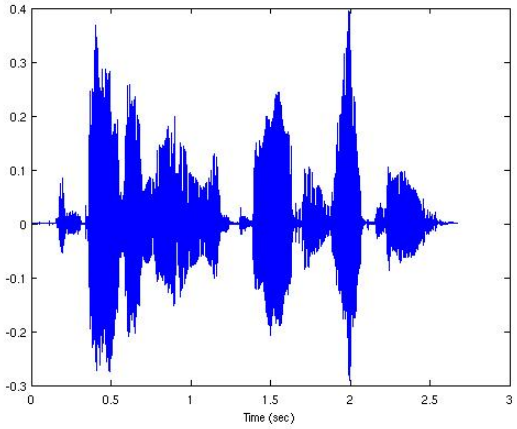


FIGURE 13 - IMPORTED USING 'WAVREAD'

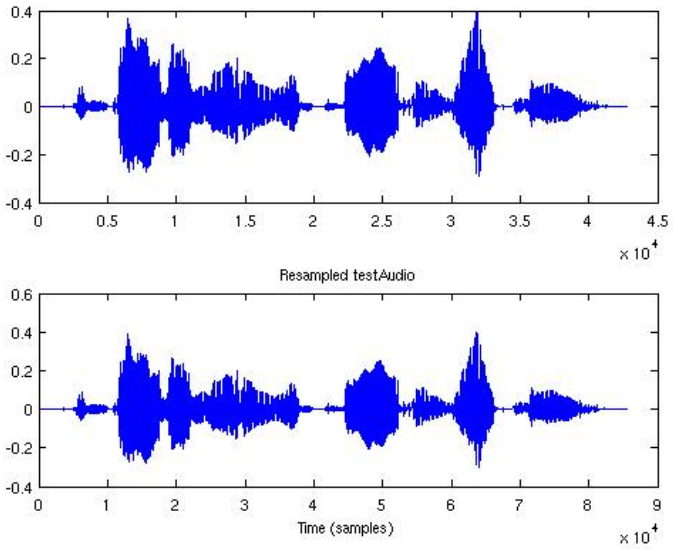


FIGURE 14 - RESAMPLED WAVFORM USING 'WAVWRITE'

3.2.2. FAST FOURIER TRANSFORM (FFT)

There are many other useful functions MATLAB has to offer however one, which may prove to be very useful and help save a lot of time is the FFT function. From our background reading we know that the FFT is extremely useful for analysing the frequency content of an audio signal. This will be crucial when creating the spectral features we need to extract.

The function Y = fft(x) returns the discrete Fourier transform of vector x computed with a FFT algorithm. We are then able to plot these results to show the magnitude & phase of the signal. Figure 15 shows the magnitude and phase of a triangle.

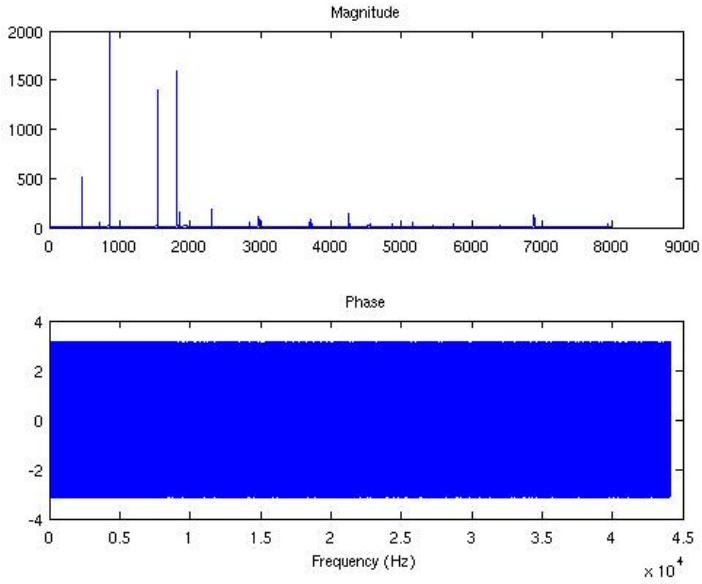


FIGURE 15 - MAGNITUDE & PHASE OF A TRIANGLE

4. CHAPTER 4 – DESIGN & DEVLOPMENT – IMAGE ANALYSIS

4.1.OVERVIEW

Following on from our motivation we decided that the first step in developing our system was to create a prototype which could take a spectrogram and try and extract features which may represent a significant change in the audio. Consider figure 16:

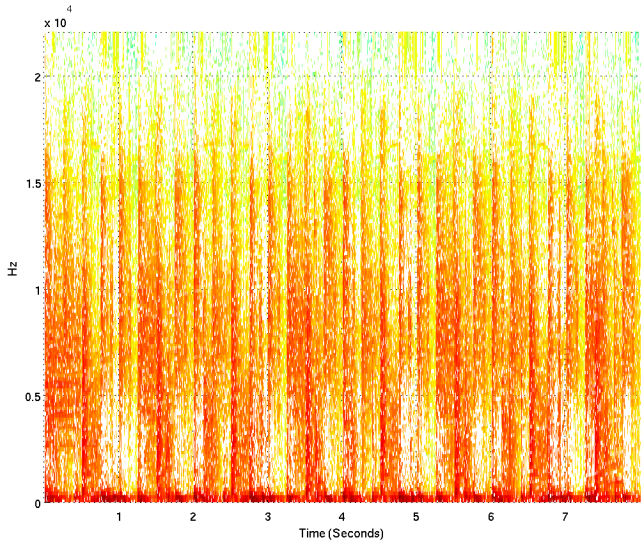


FIGURE 16 - SPECTROGRAM OF A POP SONG

Figure 16 shows a spectrogram of an audio file featuring a popular song. The song is categorised by a regular 4/4 beat with a heavy drum influence. This can be visibly seen on the figure. The darkest spikes represent the start of a bar where there is a heavy bass drum beat and the slightly lighter shade located higher up represent the subsequent beats played on the hi-hat and snare.

In figure 16 a regular beat is clearly recognisable however we are mainly going to be concerned with background environment recordings such as soundscapes or field recordings. A typical soundscape will not normally contain a regular beat and as such we will be looking for any recognisable change rather than repeating patterns. Figure 17 shows a short segment of a soundscape recording. During this extract a bus idles in an environment then sounds its horn four times at 8,9,10 and 11 seconds. This again can be clearly seen as a moment of sudden change in the soundscape. The horizontal orange band below 10kHz most likely represents the engine and background noise while the bus idles, and the sharp darker peaks at 8,9,10 and 11 represent the horn. It is a change such as this we are hoping to be able to automatically extract.

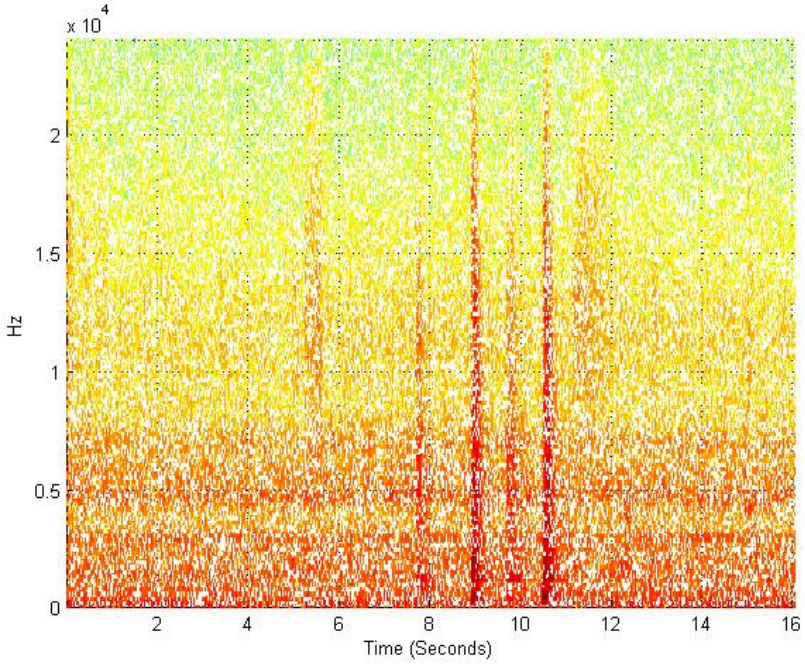


FIGURE 17 - SOUNDSCAPE RECORDING - BUS SOUNDING HORN

4.2.SPECIFIC OVERVIEW

From our motivation we will initially use image analysis techniques to try and extract the salient features present in the spectrogram. We will start by converting the image into a binary

representation. This will hopefully highlight the key (darker) areas. We will then preform a vertical projection count at each time step to obtain our feature vector.

The feature vector will represent the amount of pixels present at each time step, however we are looking for the most change as appose to the highest count. Although the highest count is likely to represent most of the key features within the audio such as beats or other significant events. It could be the case that a key feature in a soundscape is when the audio goes from a loud section to a quite section. As such we will want to find the highest degree of change. To do this we will differentiate the pixel count vector to find the highest areas of change.

Once the highest areas of change have been found, we will be able to segment the audio about these points.

4.2.1. SYSTEM DIAGRAM

Below shows the system diagram of our first prototype:

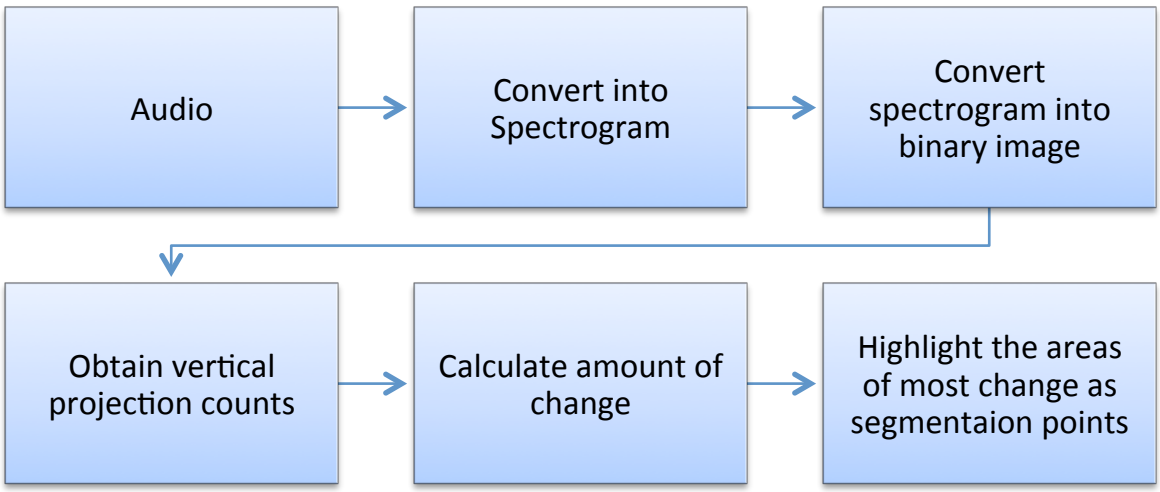


FIGURE 18 - PROTOTYPE ARCHITECTURE

4.2.2. INITIAL RESULTS

As this system was being developed as a prototype to check whether our motivation was valid, instead of using a full testing structure we decided to test the system on a range of audio files, which represented different features and change. The files that were selected included a short segment of speech and a popular song with a heavy beat. These files can be heard on the data CD.

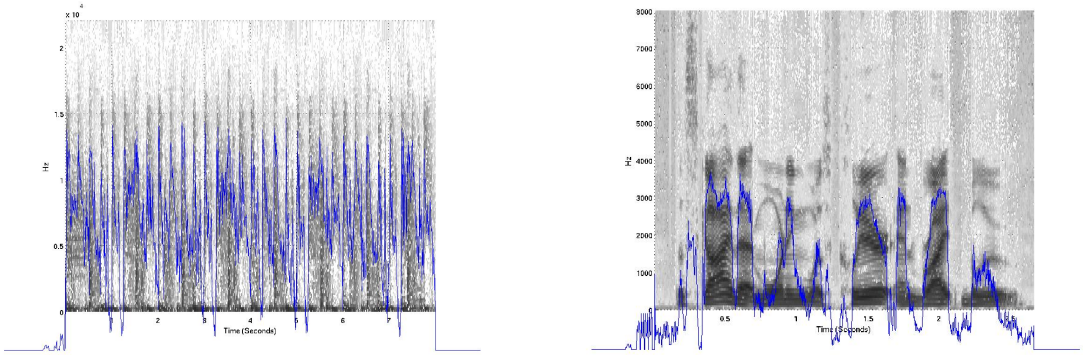


FIGURE 19 - PIXEL COUNTS (LEFT: POP SONG RIGHT: SPEECH)

The blue line in the figure 19 shows the initial vertical projection counts overlaid onto a grey scale spectrogram. Note these are taken after the image has been converted into a binary representation. The difference was then calculated to find the areas of most change. These where then added to a list which was sorted and the 'N' highest regions where selected. These regions represent an area of change and therefore a segmentation line could be created at that time. Figure 20 below shows the results for taking the top 10 regions.

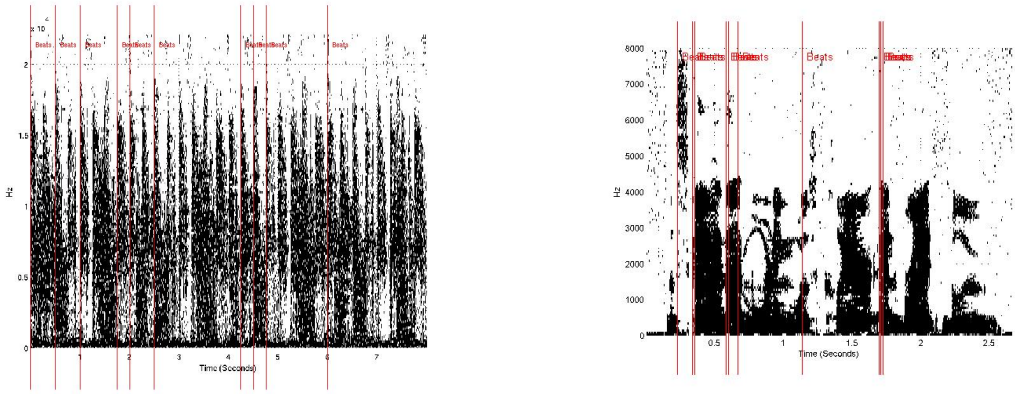


FIGURE 20 - SEGMENTATION INTO USING 10 SEGMENTS

We can see that in the pop song the program has successfully managed to identify beats. In the speech signal the program has managed to identify when a word begins and ends, however does not seem as reliable and sometimes identifies the same feature multiple times.

We also tried experimenting with a different feature vector. Instead of producing vertical projection count of the binary spectrogram we first applied an edge detector algorithm. We hoped that the edge detector would be able to locate the vertical lines and remove most of the unwanted noise. Below shows the results of applying multiple edge detector algorithms. In order to determine the required threshold the 'Otsu' automatic thresholding method was applied. This will be explained in more detail in section 4.3.3.

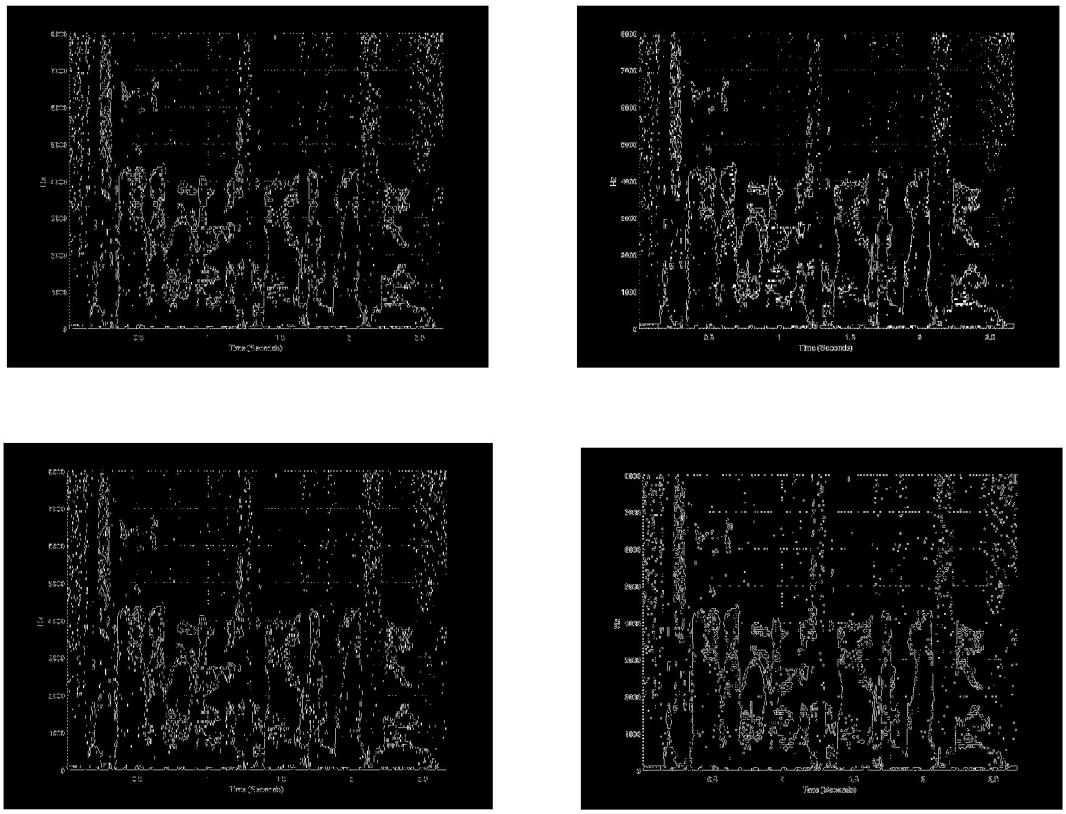


FIGURE 21 - TOP LEFT SOBEL, TOP RIGHT ROBERT, BOTTOM LEFT PREWITT, BOTTOM RIGHT CANNY - EDGE DETECTORS

We found that the Canny algorithm preformed best and by tuning its parameters it could be tailored to the recognition of edges/change in an audio file. However each audio file required its own parameter tuning, which is impractical for our use. We had hoped that the edge detector would be able to provide us with an alternative method of finding the areas of change, and it works to a certain degree. Figure 22 shows that both the edge detector and our histogram projection have similarities

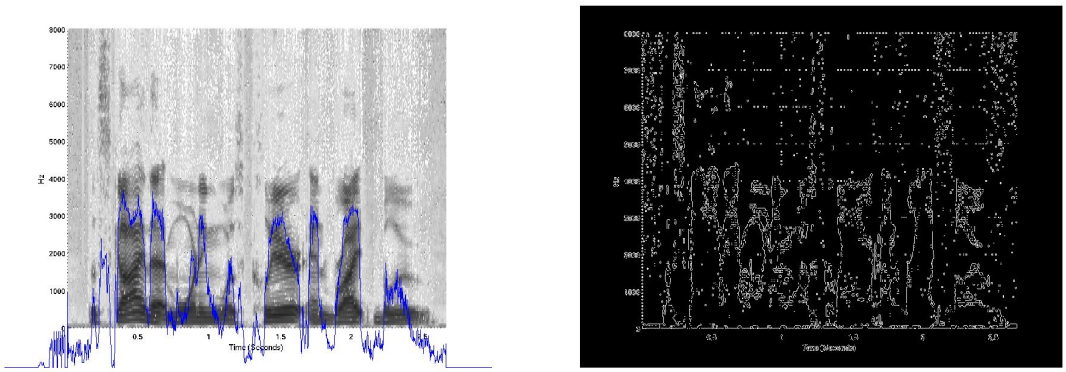


FIGURE 22 - COMPARING VERTICAL PROJECTION & CANNY EDGE DETECTION

For our purpose the histogram projection preforms better. It is also worth noting that in our final system we will not be preforming any image analysis, as such using an edge detector will not be a viable option.

4.3.IMPROVMENTS

This section we will briefly discuss some of the improvements and modifications which where made to our initial prototype. We will also discuss thresholding in more detail.

4.3.1. IMAGE PREPARATION – OFFSETTING

A problem that was encountered relatively early on was first noticed after inspecting some of the first results. When the spectrograms were exported as images from MATLAB, the spectrograms axis and white background were also exported. Figure 23 demonstrates this and shows the image; as MATLAB exports it, note a black border has been added to show the figure boundary.

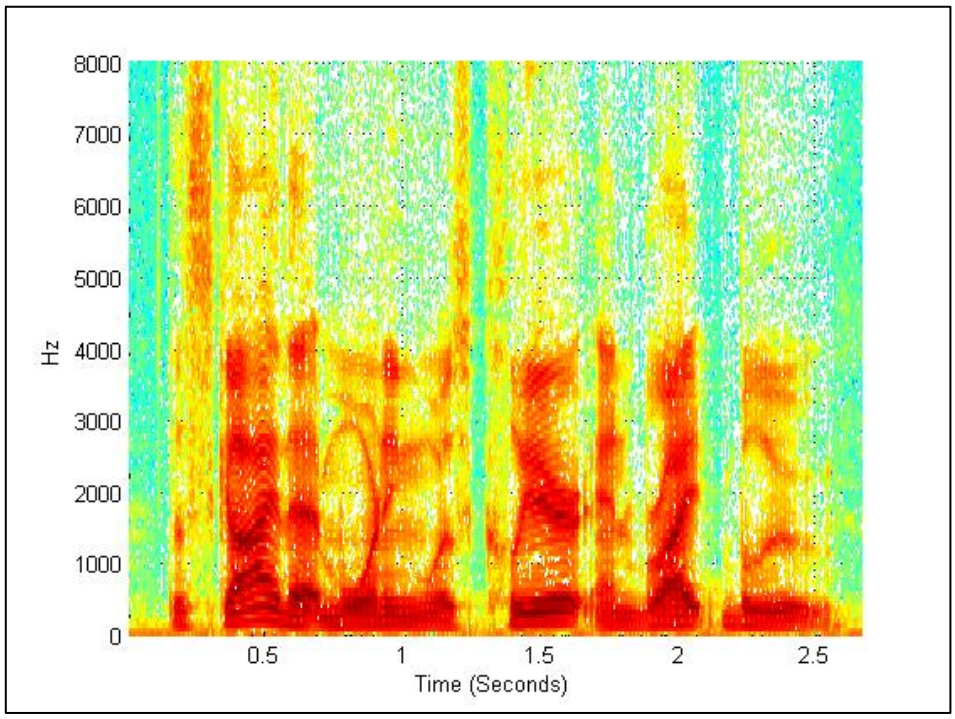


FIGURE 23 - EXPORTED SPECTROGRAM IMAGE

This created a number of problems. The first major problem is that the black axis counted towards our overall pixel counts. This meant that columns, which included labels such as the 'i' in 'Time', or any of the axis labels, would be biased. We can see from figure 24 how this problem affects the pixel counts.

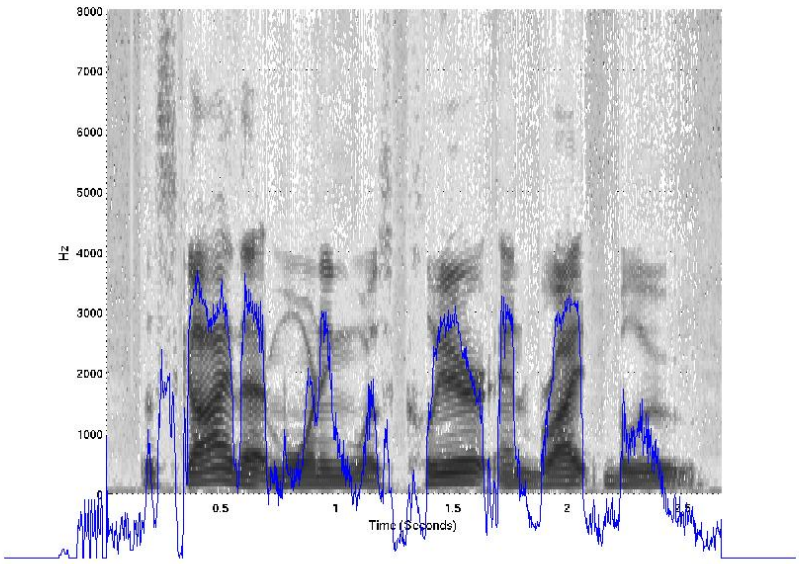


FIGURE 24 - SPECTROGAM WITH PIXEL COUNTS

The blue line represents the black pixel counts that are imprinted onto the grey scale spectrogram. If we look at time before 0 we can see minor fluctuations within the count, which represent the artefacts created by the black axis. This of course can be stopped by offsetting the counter to start at location (0,0) in the graph as appose to the picture. To work out this location we used the 'data curser' tool in MATLAB  . The data cursor enables us to read data directly from a graph by displaying the values of points we select on the image. In our case this will include the X Y location, the pixel index and the RGB value. Figure 25 shows how the data cursor was used to find the initial offset location.

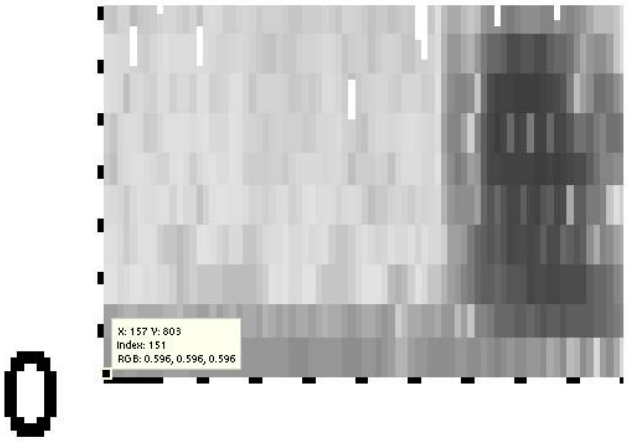


FIGURE 25 - FINDING THE OFFSET

Using this tool we were successfully able to remove the background and axis. The offset uses the following values:

Y = 69:803

X = 157:1088

As we can see from figure 26, we have successfully managed to remove the background and axis. The red line around the image has been artificially added and shows the image boundary.

FIGURE 26 - OFFSET SPECTROGRAM

4.3.2. IMAGE PREPARATION – REMOVING GRID LINES

Another major problem which was encountered was found during the testing stage where some unusual results where being noticed. We found that we were getting results, which showed a very uniform spacing; this suggested a beat was present. This looked extremely positive, however on comparison with a tagged audio file the results didn't actually correspond to beats.

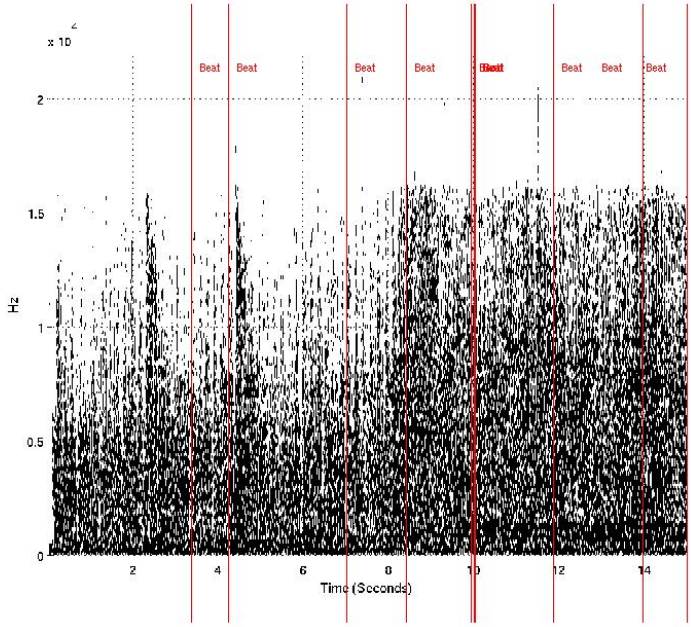


FIGURE 27 - FALSE MATCHES

If we look at figure 27, the lines the program has picked out seem to be rather uniform, which could represent a beat. However if we look closer we notice that these lines actually fall on grid lines MATLAB has created on the spectrograms. These vertical lines severely bias the counts so that it appears there is a feature present when actually it is an artefact of MATLAB. The reasons that these lines where not spotted earlier is that they are not present on every spectrogram. If we look at figure 28 no vertical/horizontal lines can be seen.

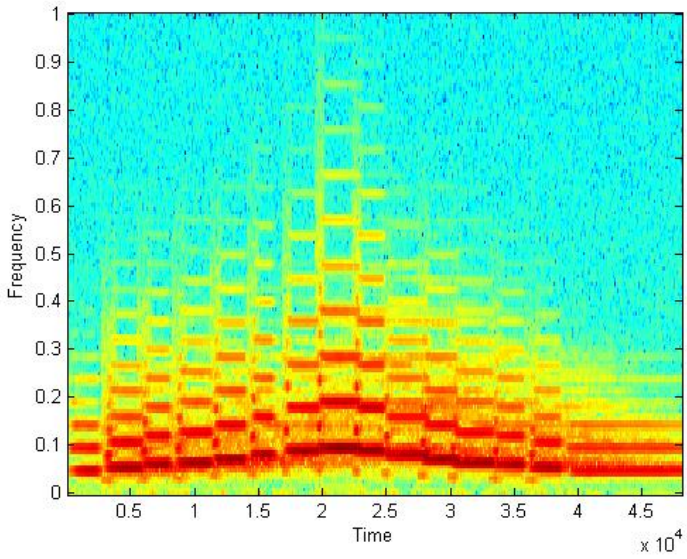


FIGURE 28 - SPECTROGRAM WITHOUT GRIDLINES

This presented a tricky problem, as we were required to remove black lines from within some spectrograms and not others. However a relatively simple solution was found. We decided that as our original spectrograms didn't contain any black pixels we could simply turn any black pixel in the image white. This would remove all the black line artefacts in the spectrograms but maintain all of the spectrogram makeup.

We again used the pixel analysis tool to find the exact pixel colour value. This value was RGB:0,0,0. We ran a simple loop, which located all the pixels of this colour and convert them into white (RGB: 255,255,255). The resulting image can be seen in figure 29.

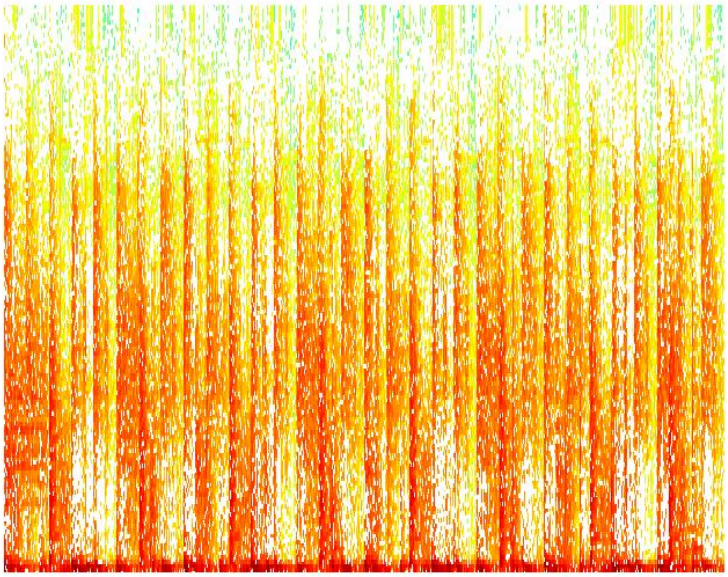


FIGURE 29 - SPECTROGRAM WITH BLACK PIXELS REMOVED

4.3.3. THRESHOLDING

One way we can extract features from the spectrogram is to take vertical histogram counts of pixels at each time step. This proved to be very difficult using the current colour spectrogram due to the multi-dimensional data obtained from the RGB image. We therefore decided to convert the spectrogram into a binary image consisting purely of black and white pixels. This allowed us to simply sum the amount of black/white pixels at each time step then measure the difference between them. The greatest differences will provide us with the areas in the audio clip where the greatest change occurs.

In order to convert the image into a binary representation we first converted the image into a grey scale representation and then preformed particular threshold analysis in order to determine which pixels are turned into black/white.

Figure 30 shows the original colour spectrogram and the grey scale counterpart

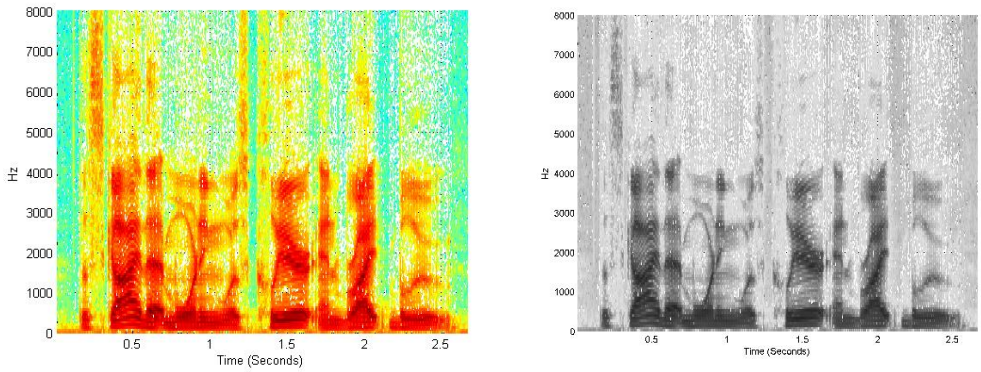


FIGURE 30 - CONVERTING THE IMAGE INTO GREYSCALE

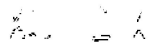
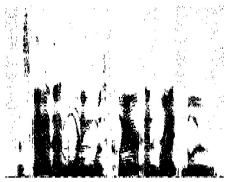
As we can see from these two images, the structure of the spectrogram as been preserved and it is still clear where audio changes might occur. To preform this conversion we used MATLAB's '`rgb2gray`' function. The '`rgb2gray`' function converts RGB images to grey scale by eliminating the hue and saturation information while retaining the luminance. The algorithm `rgb2gray` uses converts RGB values to grey scale values by forming a weighted sum of the *R*, *G*, and *B* components: (http://www.mathworks.co.uk/help/toolbox/images/ref/rgb2gray.html)

$$0.2989 * R + 0.5870 * G + 0.1140 * B$$

By preforming the grey scale algorithm we have converted the image into a single dimensional array with pixel values ranging from 0-255 where 0 represents a fully black pixel and 255 white. Although this could potentially provide us with a means of calculating change by simply summing all the values at each time step in each column, it would produce a lot of unwanted noise. By converting the image into binary representation we are able to reduce the noise and obtain counts, which are more representative of that time step.

MATLAB again provides us a means of doing this by using the function '`im2bw`'. The '`im2bw`' function converts the grey scale image to a binary image. The output image replaces all pixels in the input image with luminance greater than a threshold with the value of 1 (white) or 0 (black). Note that this threshold is scaled between 0-1 therefore a threshold of 0.5 is midway between black and white.

The problem for us is how we choose our threshold. We therefore decided to experiment with a range of thresholds and produce images for each and compare these to analyse which threshold is most suitable.

Threshold	Speech	Notes
0.1		This value of threshold is too low. The image is blank and therefore all the detail has been lost. All pixels have been assigned to white.
0.2		Some detail has now started to appear, however still does not represent the true spectrogram.
0.3		This is the first time we can potentially see key features starting to arise. Still no higher frequency components are observable.
0.4		Some of the higher frequency components become visible, however only on one or two pixels. The lower components are now becoming substantial and key features may be able to be extracted.
0.5		More of the higher frequency components are now becoming visible and there is further enrichment of the lower components.
0.6		The higher frequency components now play a dominant role, and the lower frequency section is further enriched.
0.7		Now the higher frequency components are becoming denser, however slight noise has started to appear. The features however are clearly visible.

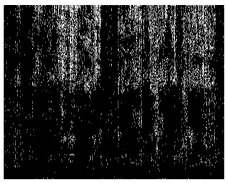
0.8		Noise has defiantly started to appear and now masks some of the detail in the spectrogram.
0.9		Further noise has been added and it is now very hard to distinguish any features with the eye. Most of the pixels have been assigned to black.

TABLE 2 - OBSERVING THE CHANGES TO A SPEECH SPECTROGRAM

From the above table that a threshold between 0.6 and 0.7 should be implemented as this provides the best representation with the least noise. Any higher that 0.7 will not be able to provide an accurate representation and lead to many false results. It may be possible to use much lower thresholds such as 0.3 or 0.4 to obtain the key areas, however as these miss lots of lower amplitude content and as such would act more as amplitude detectors.

To further investigate this issue a popular song was analysed in a similar manner.

Threshold	Pop Song	Notes
0.1		Like the speech, the 0.1 threshold provides us with no visible features. It would be impossible to locate the key features using this threshold.
0.2		We can now start to see parts of the spectrogram appearing, however again the information here is too little to extract any kind of features.
0.3		Key points now start to emerge. Beats can clearly be seen however are still not prominent.

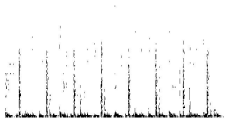
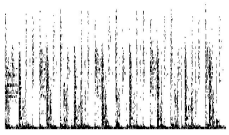
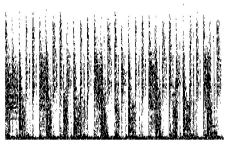
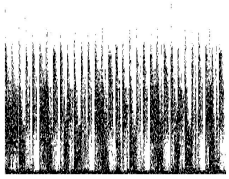
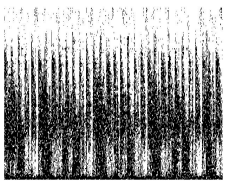
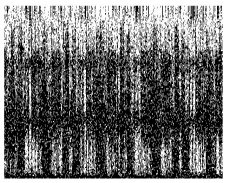
0.4		Quickly the main beats have become more prominent. With each increase in threshold, the higher components become more visible.
0.5		The beats again become more prominent however noise has also started to appear. More of the note onsets have also appeared in-between the main beats. It is easy to identify the tempo note onsets.
0.6		The features become more prominent, throughout the whole frequency spectrum
0.7		The features become more prominent, throughout the whole frequency spectrum, some of the extremely high frequency components are starting to show.
0.8		Although the beats are visible, noise is also heavily present which could lead to false features being identified.
0.9		Noise has now clouded the image. The beats have become indistinguishable.

TABLE 3 - OBSERVING THE CHANGES TO A POPULAR SONG SPECTROGRAM

We can see from this table a much lower threshold is required. Relatively early on (around 0.3) the main features and beats become recognisable. This shows that each style of audio requires its own threshold level.

4.3.4. OTSU METHOD

To try and find an automatic approach we decided to investigate Otsus method. This algorithm assumes that the image to be thresholded contains two classes of pixels for example foreground

and background. It then calculates the optimum threshold separating those two classes so that their intra-class variance is minimal (Otsu, 1979). The aim is to find the threshold value where the sum of foreground and background spreads is at its minimum.

Within-class variance defined as a weighted sum of variances of the two classes (Morse, 2000):

$$\sigma^2_{within}(T) = n_B(T)\sigma^2_B(t) + n_O(T)\sigma^2_O(T) \qquad (12)$$

where :

$$n_B(T) = \sum_{i=0}^{T-1} p(i) \qquad (13)$$

$$n_0(T) = \sum_{i=T}^{N-1} p(i) \qquad (14)$$

$\sigma^2_B(t) = the\ variance\ of\ the\ picels\ in\ the\ background\ (below\ threshold)$

$\sigma^2_O(t) = the\ variance\ of\ the\ picels\ in\ the\ foreground\ (above\ threshold)$

Using Otsu's method on both of our test cases we obtain the following results:

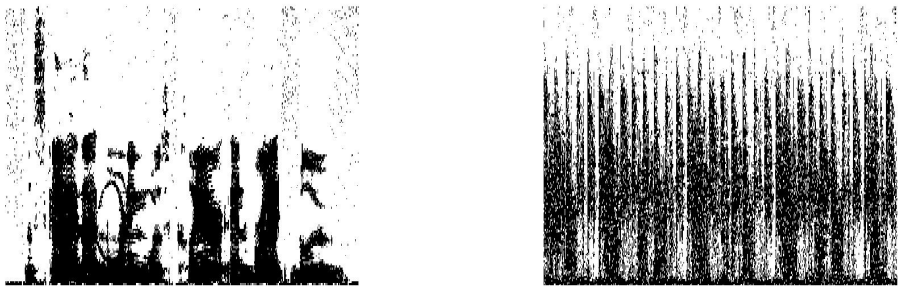


FIGURE 31 - SPECTROGRAM'S WITH OTSU THRESHOLD METHOD

4.3.5. FURTHER THRESHOLDING TESTS

Figure 32 shows the pixel count for the three test files (soundscape, speech and a pop song) against the threshold value.

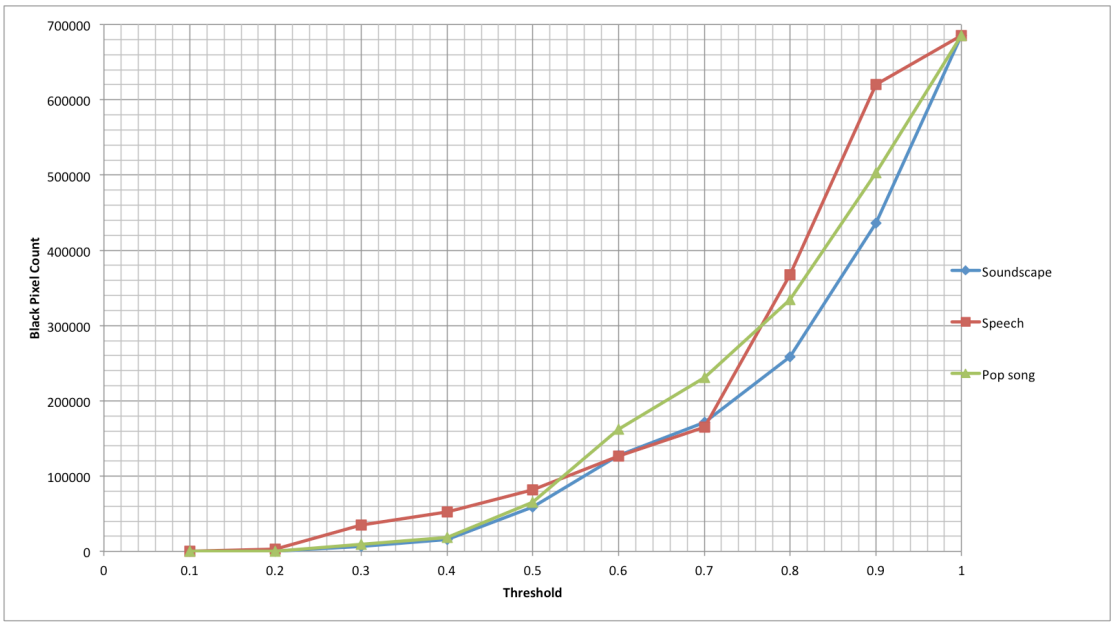


FIGURE 32 - PIXEL COUNT, THESHOLD COMPARISON

We can see from figure 32 that as we increase the threshold to one the amount of black pixel starts to increase at an expediential rate. It therefore becomes more sensitive to change at the later thresholds.

Currently we have only visually analysed the impacts the threshold has. To further test the impact the threshold wields we decided to test the full system using a range of thresholds to show us if the threshold is a crucial parameter. If our results remain the same throughout the tests i.e. the segmentation remains constant then we can conclude that impact of thresholding is a minor component however if the results differ between threshold levels then thresholding needs to be seriously considered. By looking at the figure 32 we decided to test the system between the range 0.2 to 0.8 as any less and the images don't contain any black pixels and any more the image becomes to noisy and unstable.

We will also compare two different audio files to look at how the style of the audio might effect and change our thresholding decisions.

Speech Signal:

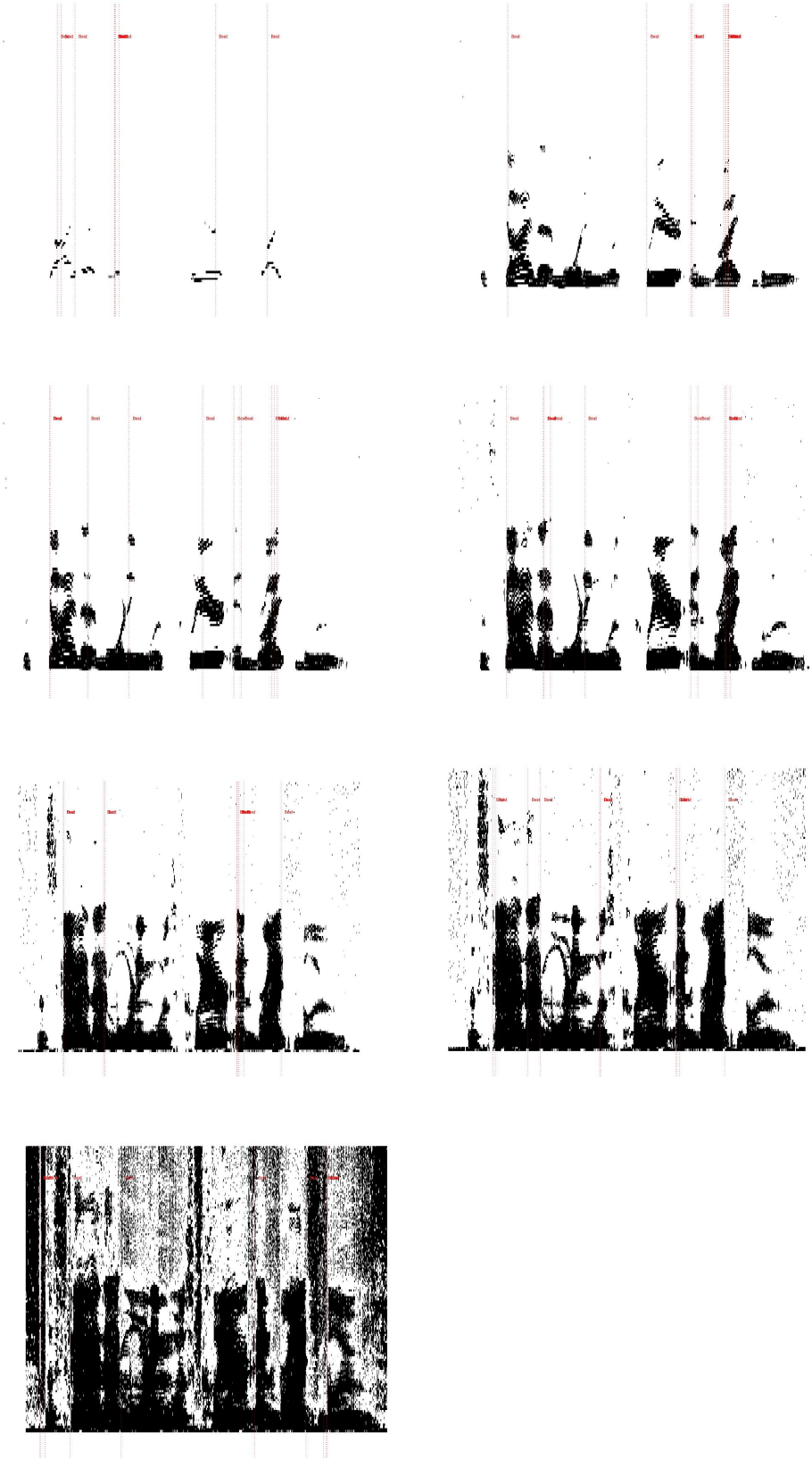


FIGURE 33 - SHOWING THE SEGMENTATION AS THRESHOLD IS INCREASED

From figure 33 we can clearly see that the change in threshold undoubtedly affects where the segments are created. The below images show the same test for the popular song:

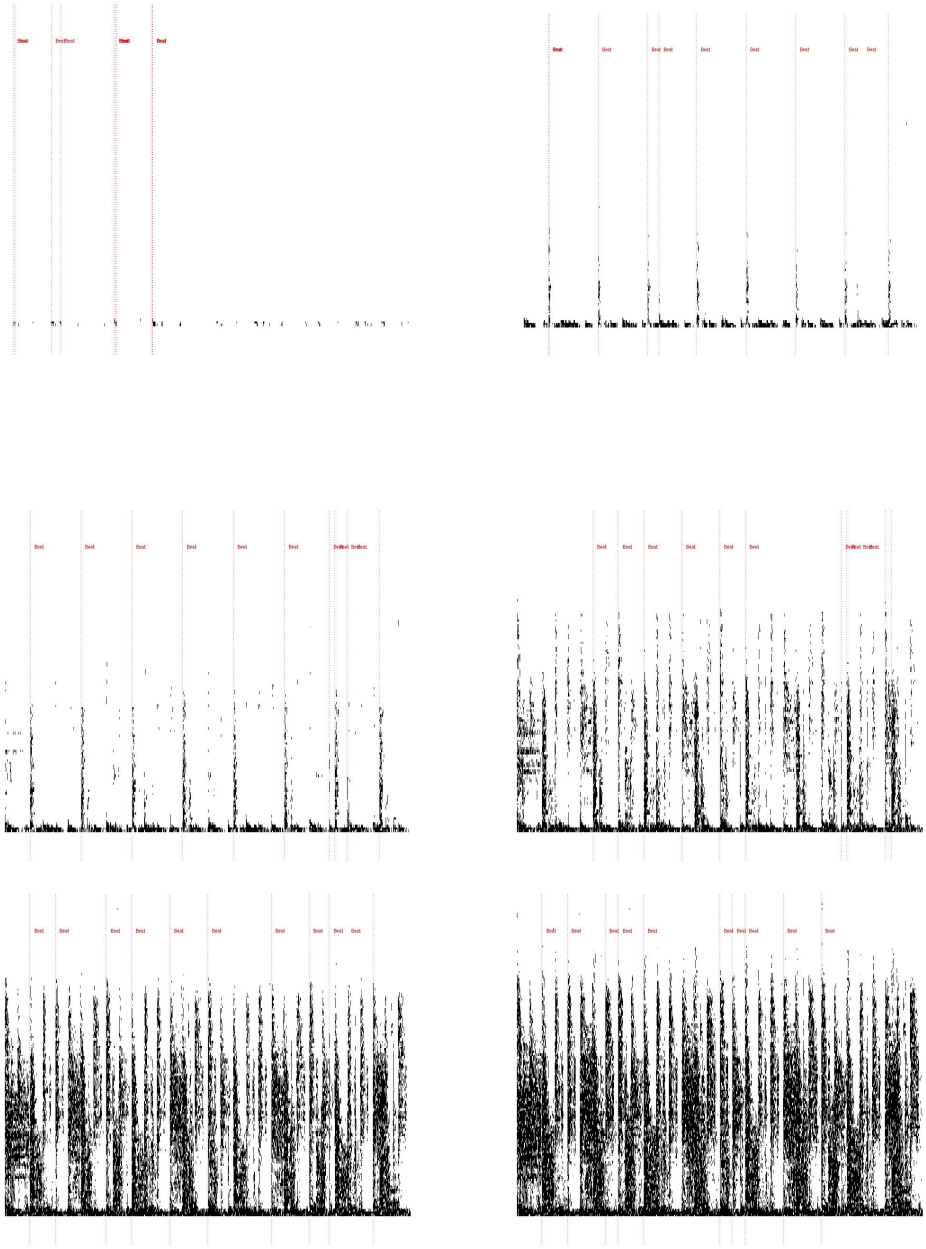


FIGURE 34 - SHOWING THE SEGMENTATION AS THRESHOLD INCREASES

In figure 34 we have analysed a section of pop music, which has a very consistent 4/4 beat. We can see that even with a very low threshold (0.3) the system has managed to locate the heavy bass beats of this piece of audio. We can also see that as we increase the threshold and more of the spectrogram becomes visible, the segment location lines remain relatively consistent. We noticed that with a lower threshold we are able to locate all the bass beats, but as we increase the threshold we also start to locate the offbeat snare, however more errors where also seen. As the system seemed to be able to accurately locate beats even right through to the later thresholds we decided to see if we could locate both the snare and bass by increasing the number of allowed segments. Figure 35 shows the segmentation with 25 possible segments. We

can now see that it manages to highlight almost every occurrence of the drums successfully with some slight error towards the last quarter. As we can now locate the beats of the piece this information can be used to work out the tempo.

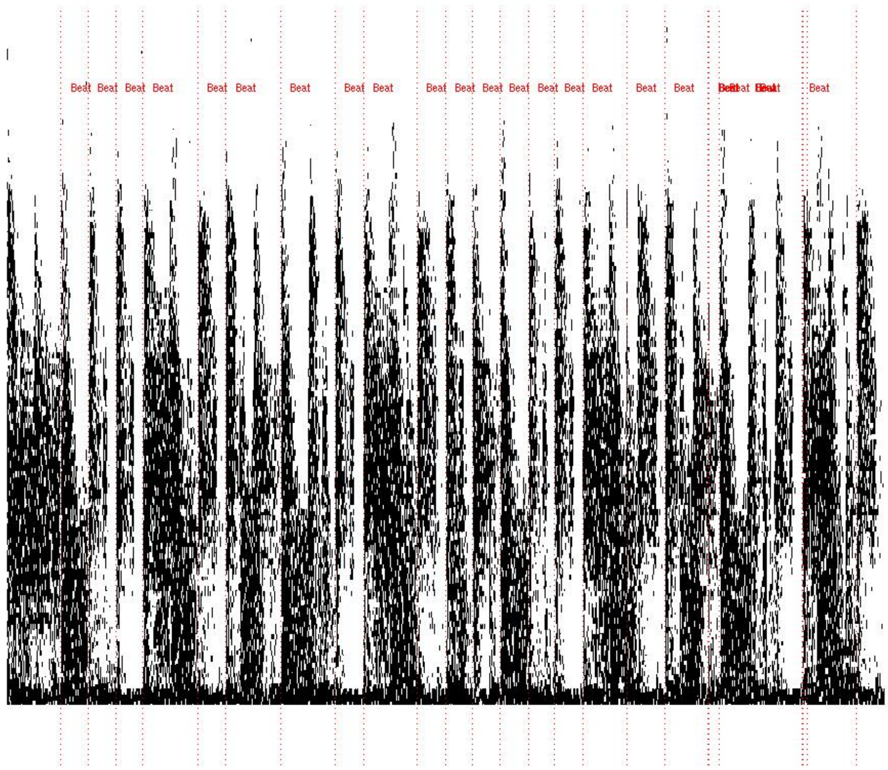


FIGURE 35 - SEGMENTATION USING 25 SEGMENTS

5. CHAPTER 5 - DESIGN & DEVLOPMENT – SPECTRAL ANALYSIS

5.1. OVERVIEW

In the previous chapter we saw how we could use the spectrogram to separate and segment key features from an audio file. This idea was formulated from our motivation and as such we used image analysis techniques in order to test this theory. This chapter we will further explore and develop the system to use spectral analysis techniques to separate key features, however we will now extract these features into audio thumbnails.

5.1.1. SPECIFIC OVERVIEW

Although our prototype in the previous chapter seemed to work reasonable effectively there is a fundamental flaw in it, which can be avoided. The prototype takes an audio signal, produces a

spectrogram, saves this spectrogram as an image and then preforms image analysis in order to obtain our feature. The problem with this is that we lose lots of information, which could be retained. Instead of converting and saving the spectrogram as an image we can instead use the mathematics and numbers that constitute the spectrogram to obtain our spectral features.

In our prototype we took vertical projections of pixel counts at each timestep, then differentiated these in order to obtain our feature vector. In this case however we don't have a binary image in which we can simply count the number of pixels, we will therefore look at the numeric values which make up each column in the spectrogram and measure the change between columns. Summing the results of our spectral analysis at each timestep, then measuring the change between neighbouring time steps will obtain this.

5.1.2. SYSTEM ARCHITECTURE

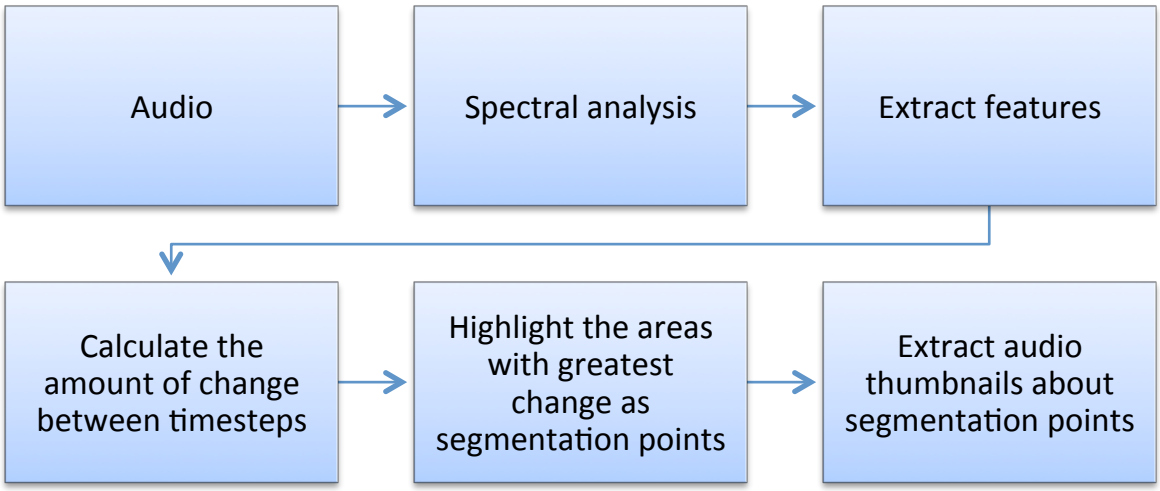


FIGURE 36 - SYSTEM ARCHITECTURE

5.2.DEVELOPING THE SYSTEM

5.2.1. SPECTRAL FEATURES

Currently we have looked at spectrograms with a black box approach, in that audio goes in and an image is returned. This was appropriate for our prototype and to test our motivation theory however as mentioned previously we lose lots of information, which could be retained. In this

section we will briefly talk about how spectrograms are produced and how we can use the mathematics and numbers that constitute the spectrogram instead of the raw image file.

As we know a spectrogram is a time-varying spectral representation that shows how the spectral density of an audio signal varies over time (Haykin, 1991). Looking at a spectrogram (figure 37) the horizontal axis represents time, vertical frequency and a third dimension indicating the amplitude. The amplitude is often represented by colour intensity however can also be represented in a 3D graph (figure 37).

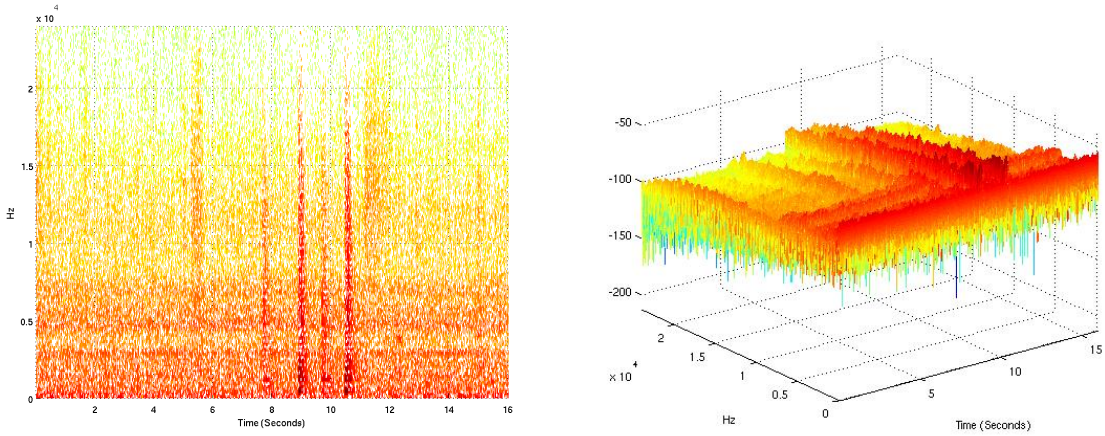


FIGURE 37 - 2D + 3D SPECTROGRAM

We are however interested in how these spectrograms are generated. Spectrograms can be created in a number of ways, however the most common and one we will be using uses the short-time Fourier transform (STFT). Another time frequency transform we could consider is the constant Q transform discussed in section 2.2.2.

To compute a STFT the data (audio file) is first dived up into a series of blocks equal to a window size, note that these blocks usually overlap each other to reduce boundary artifacts. A fast Fourier transform is then applied to each block of data, this obtains the frequency content of each block. The result is added to a matrix, which records magnitude and phase for each point in time and frequency. It can be expressed by:

$$STFT\{x[n]\}(m,\omega) \equiv X(m,\omega) = \sum_{n=-\infty}^{\infty} x[n]\omega[n-m]e^{-j\omega n} \qquad (15)$$

where: *x[n] = signal*

w[n] = window function

m is discrete and ω continuous

The spectrogram of a signal x(n) can be estimated by computing the squared magnitude of the STFT of the signal as follows (National Instruments, 2009):

$$Spectrogram\ (n,\omega) = |STFT(n,\omega)|^2 \qquad (16)$$

One of the main properties, which affect the spectrogram, is the window length. The window length affects the time resolution and frequency resolution of the STFT. A narrow window has a good time resolution but rough frequency resolution. A wide window results in a good frequency resolution but rough time resolution as the wide windows have a long time duration and narrow frequency bandwidth (National Instruments, 2009).

In figure 38 we examine the effects of modifying the window length. The figure shows a 20 second clip of audio comprising of four tones joined together in 5-second intervals the frequencies are (10,25,50,100 Hz).

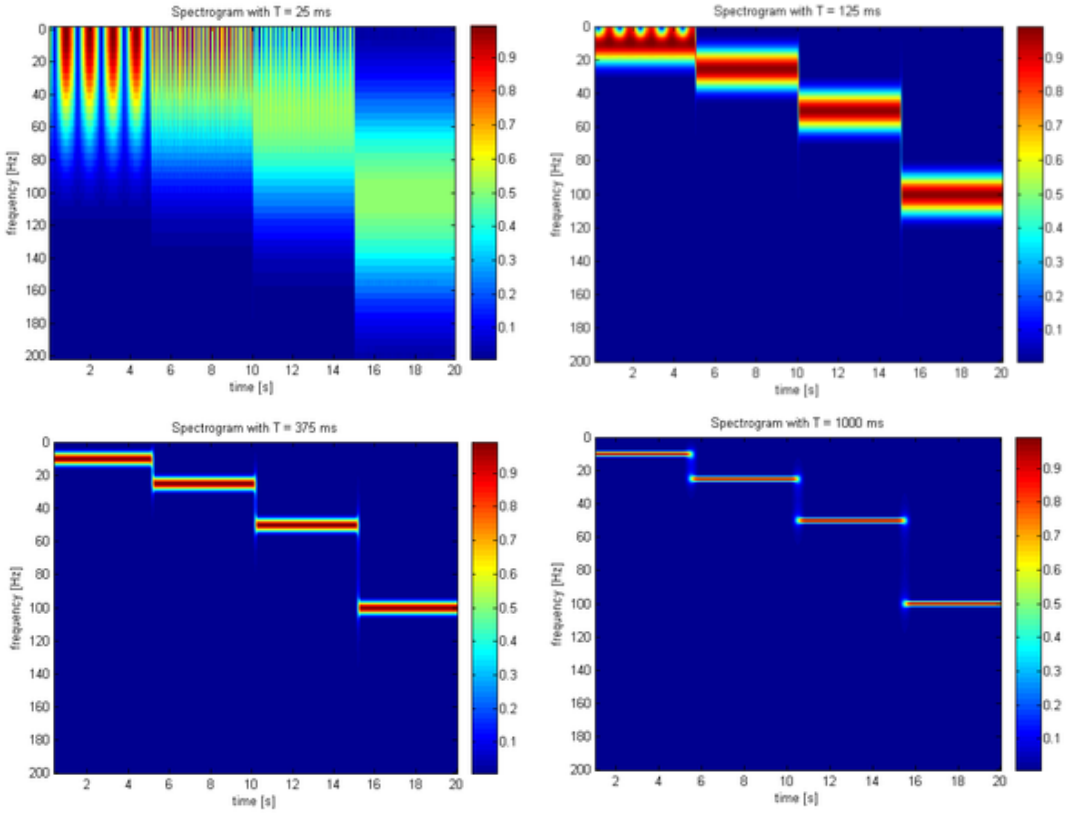


FIGURE 38 - CHANGING WINDOW LENGHTS – CODE OBTAINED FROM (ALEJO2083, 2010)

During our prototype we used MATLAB to produce our spectrograms, on further investigation we found that when the spectrogram function is used with optional outputs, it is possible to return the short-time Fourier transform of the input signal instead or as well as the visual display. We would then be able to use the returned information to calculate our features.

Although MATLAB provided us with a way of obtaining the STFT information we needed, we also created our own function to help us fully understand how the STFT worked, the effect of modifying the parameters and to compare and contrast the performance to the MATLAB spectrogram function.

The STFT information returned by MATLAB is such that each column contains an estimate of the short-term, time-localized frequency content of the signal. Time increases across the columns and frequency increases down the rows. As our prototype proved successful we decided to mimic it and sum each column to obtain a 1 dimensional feature vector equal to the number of timesteps specified in the STFT. Figure 39 shows this vector plotted against time. Note we have taken the absolute value for this graph as the vector comprises of complex values.

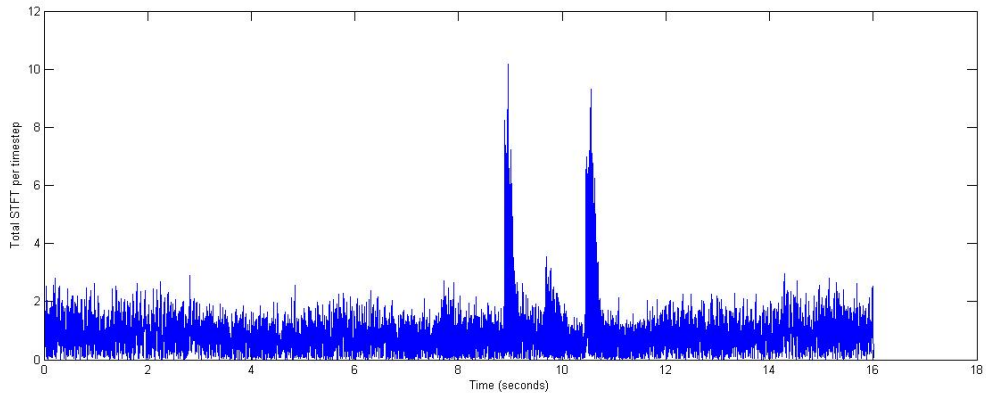


FIGURE 39 - ABSOLUTE VALUE FOR TOTAL STFT COUNT PER TIMESTEP - BUS EXAMPLE

Similar to our prototype we could apply a threshold to this and try and obtain our segments however we are not just simply looking for the areas with the most going on, we are trying to find the areas, which have the most change. As such we will examine each neighbouring time step and calculate the change between them. As we have our feature values stored in a 1 dimensional vector we are able to use MATLAB `'diff'` function. The `diff(x)` function calculates differences between adjacent elements of `x`. This was exactly what we required and the results are shown in figure 40.

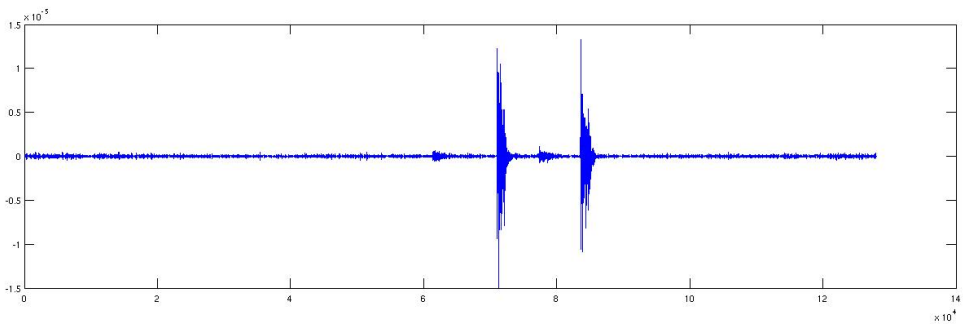


FIGURE 40 - AMMOUNT OF CHANGE- BUS EXAMPLE

From figure 40 we can see that we have obtained the amount of change against time. We can see that we have minimised the noise and accentuated the feature we wish to segment. Note that we have obtained both positive and negative results. The reason for this is that when the audio changes from a high power level to a lower power level we will have a negative change. For our purpose we want to find the sections within the audio that represent the most change, the fact that it is negative does not matter i.e. a noisy scenario which suddenly becomes quite is still interesting and a feature we wish to pick up. As such we will take the absolute value of the change vector to obtain our final feature. The result can be seen in figure 41.

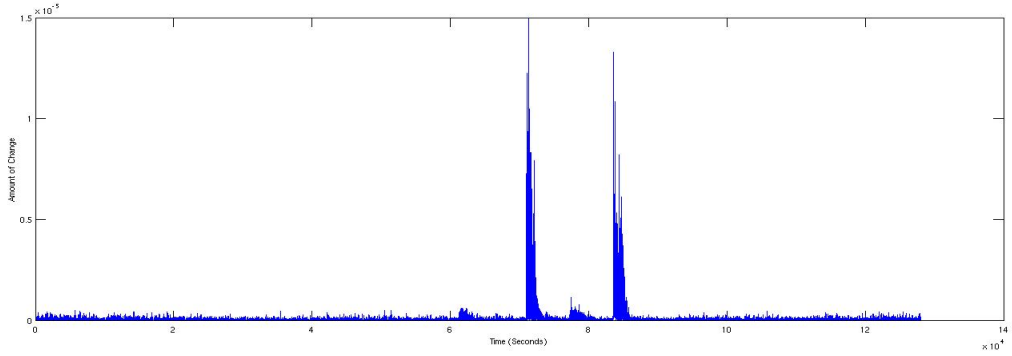


FIGURE 41 - ABSOLUTE VALUE CHANGE VECTOR - BUS EXAMPLE

5.2.2. IMPROVING THE SELECTION

In our initial prototype our system relied on human input to decide how many key segments their needed to be. We initially set a figure and it was iteratively changed until the correct amount of segments was found. This of course created many problems as we have seen in the previous section. Some tracks may only have a couple of key defining moments whereas others may have tens or even hundreds. It is also worth noting that our aim is to make an automatic system, which finds segments, and although by setting an initial amount we can achieve this, it still requires human tuning to find the optimum amounts. To improve this we decided to revisit the idea of thresholding, and introduce a threshold at which a segment could be classified as 'key'.

The first thing that was tried was to take the mean average of the absolute value of amount of change and define anything a certain amount above this it could be classed as 'key' and anything below not. The absolute value is used as we wish to find the areas of most change regardless if this is positive or negative.

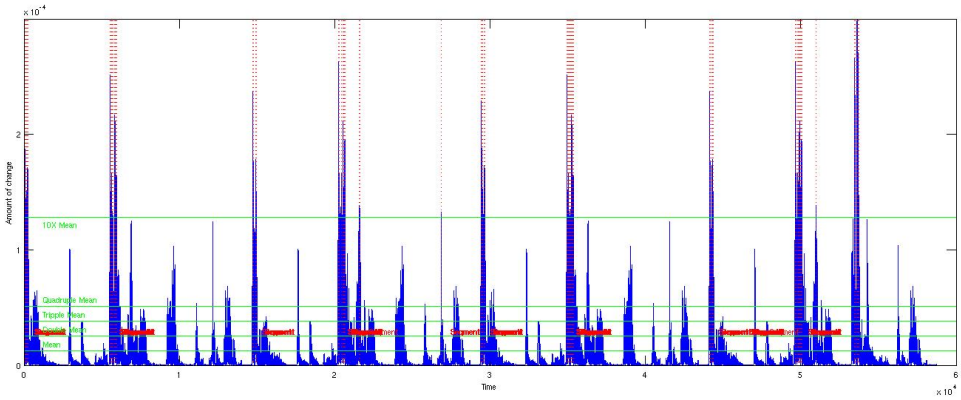


FIGURE 42 – MEAN THRESHOLD LINES – SEGMENTATION ABOUT 10X MEAN VALUE

Figure 42 shows the absolute value for the amount of change plotted against time with the mean value and subsequent threshold lines in green overlaid. The segmentation in this case has taken 10x the mean value; the red dotted lines show the sample where this segmentation takes place. Although this works reasonably well, we again find the problem that different audio files require different thresholds. For example if we consider figure 43 the segmentation occurs using 4x the mean.

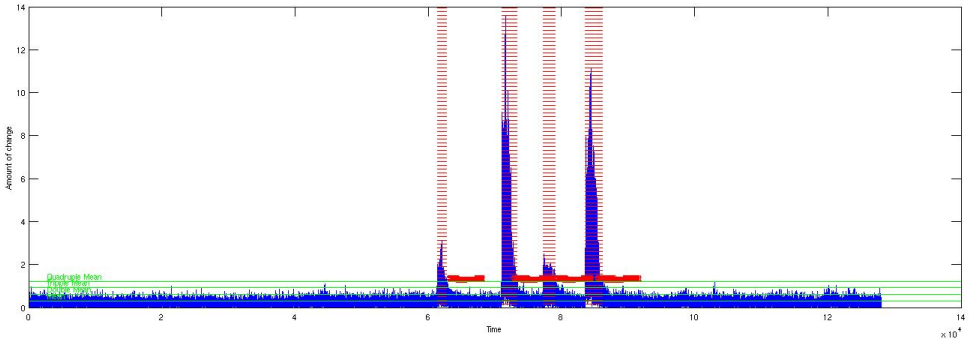


FIGURE 43 - SEGMENTATION ABOUT 4X MEAN VALUE

With so much variance, instead of just using a multiplication of the mean we decided to use the standard deviation as a base mark. The standard deviation shows how much variation exists from the mean value. A low standard deviation indicates that the data points tend to be very close to the mean whereas a high standard deviation shows the points vary a lot more from the mean. Standard deviation is given by:

$$\sigma = \sqrt{\frac{\sum(X - \bar{X})^2}{n}} \qquad (17)$$

Where: *X = each value in the data set*

$\bar{X}$ = mean of all values in data set

n= number of values in the dataset

The standard deviation measure helps us find how far typical values tend to be from the mean as such if we take a value which is 2-3 times outside that standard deviation we can class it as a change.

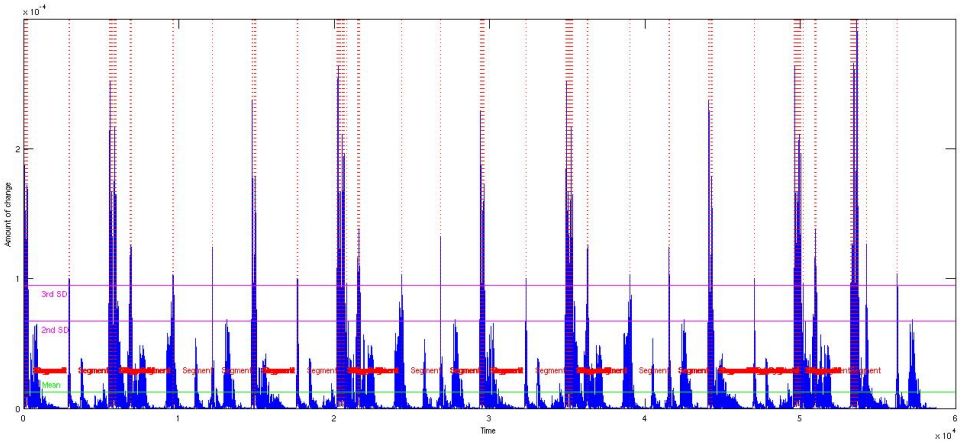


FIGURE 44 - STANDARD DEVIATION THRESHOLD LINES - SEGMENTATION ABOUT 3RD SD

5.2.3. EXTRACTING THE AUDIO

Currently our system highlights areas that contain the key segments we wish to extract. The way it does this however is by highlighting the samples with the most change. Ideally our program should only highlight the start and end sample for each segment however due to noise and segments, which contain change within them, this is not the case. If we take a closer magnified look at a single segment we can see the problem (figure 45).

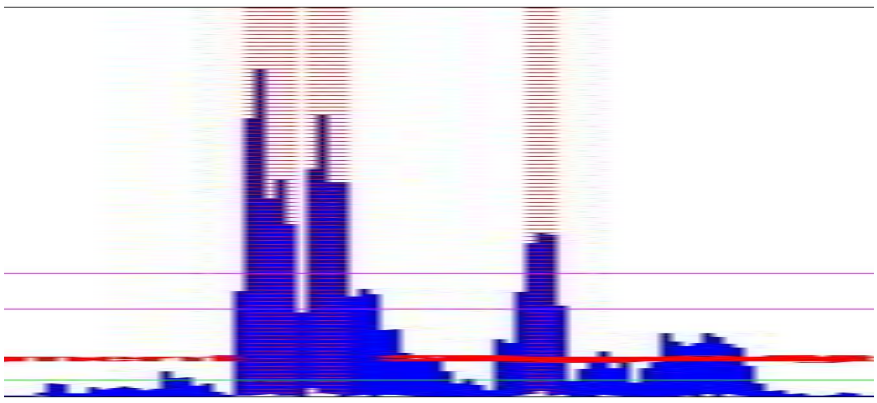


FIGURE 45 - MAGNIFIED SEGMENT (TIME ALONG X AXIS, AMOUNT OF CHANGE ALONG Y)

We can see from figure 45 that our program has been able to successfully locate the start and end sample however we have also located many samples in-between. The problem occurs in how we define the start and end of a segment. We initially said that each adjacent sample, which is above the threshold, is part of the same segment. This however proved to be unsuccessful as we ended up with multiple segments that represented the same key section of audio. For example in figure 45 although we only wish to extract a single segment we would actually extract 3.

One way of potentially solving this problem would be to use the mean-shift algorithm. Mean-shift is a simple interactive procedure that shifts each data point to the average of data points (Cheng, 1995).

Mean-shift introduced by (Fukunaga & Hostetler, 1975) is an iterative method, and starts with an initial estimate (x). Mean-shift treats the points in the feature space as a probability density function. Dense regions in the feature space correspond to local maxima. A kernel function is then given which determines the weight of the nearby points for re-estimation of the mean. Typically a Gaussian kernel is used given by:

$$K(xi - x) = e^{c\|xi-x\|^2} \qquad (18)$$

A possible problem we could encounter is selecting the size of the mean-shift kernel. The kernel size directly determines the size of the window within which sample weights are examined. This kernel scale is a crucial parameter for the mean-shift algorithm (Collins, 2003).

The weighted mean of the density in the window determined by K is:

$$m(x) = \frac{\sum xi\epsilon N(x)K(xi-x)xi}{\sum xi\epsilon N(x)K(xi-x)} \qquad (19)$$

Where *N(x) is the neighbourhood of x*

The mean-shift algorithm now sets x ← m(x), and repeats the estimation until m(x) converges (Comaniciu, 2002).

Looking at mean-shift at the high level we are effectively fixing a window around each point, computing the means of the data within that window then shifting the window to the mean and repeating until convergence (Thirumuruganathan, 2010). We will use this idea and apply it to our system.

Our system takes each point, which has been marked as a sample then checks to see if any other samples have been marked within a given range. If they have we will consider them as part of

the same segment. The range we are talking about can be thought of as the kernel function and as mentioned before the kernel size directly determines the window in which sample weights are examined. A large kernel (or range) will increase the size of the window and therefore decrease the amount of segments found. A small kernel size will mean that we only check samples, which are very close, this in turn will increase the amount of segments our system finds. To further examine this phenomenon we tested our system with a range of kernel separation sizes. The results for this can be seen in appendix C. A typical result is shown in figure 46.

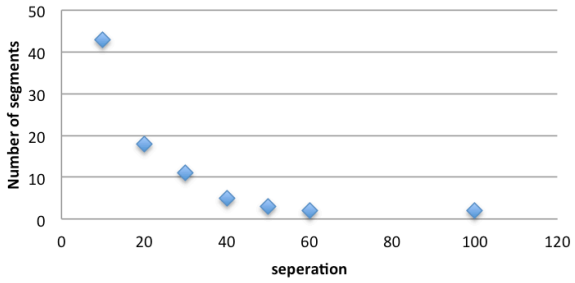


FIGURE 46 - AFFECTS OF ADJUSTING THE KERNEL SIZE

We can see from all the tests that the number of segments based on the separation (kernel) size follows an expediential decay. This backs up our theory that if we increase the kernel size the number of segments will decrease. We can also use these tests to set a global level for use in our validation. We noticed that in most of our examples by the time our separation value approached 1000 the amount of segments started to stabilise. There were however examples which did not stabilise until around a separation of 8000. Currently we are defining kernel size in terms of samples however this can be equated into seconds to help visualise the separation. With the current test parameters of the STFT set, 8000 samples relates to 1 second of audio.

The higher we set the kernel size the more computing power is needed and therefore the system takes longer to run. Also if we set the kernel too high we would end up classifying multiple segments into a single segment. For example if the kernel was set to 80,000 any features which are within ten seconds of each other would be classified as a single feature. From our tests many of the examples stabilise at 1000, however there were occurrences of files, which required a higher kernel. Setting the kernel at 8000 ensures that we capture the correct number of segments without classifying multiple segments into the same file.

5.2.4. CREATING THE AUDIO THUMBNAILS

Now that we have managed to successfully locate the start and end sample for each of the segmentations, the next step is to extract these key segments in audio form for the user to listen

to and use. The result will be a list of audio thumbnails which will allow the user to find the key defining moments as appose to listening to the whole audio file and manually segmenting them.

A possible way of achieving this goal was to use the Analysis & Resynthesis Sound Spectrograph (ARSS). ARSS is a program that analyses a sound file, coverts into a spectrogram and is able to synthesise this spectrogram, or any other user-created image, back into a sound (Rouzic, 2009). Examples of this are included on the data CD and have been taken from (Rouzic, 2009). On the CD we show two examples, these include Johann Strauss II's the Blue Danube (figure 47 left) and speech taken from Hal 9000 (figure 47 right).

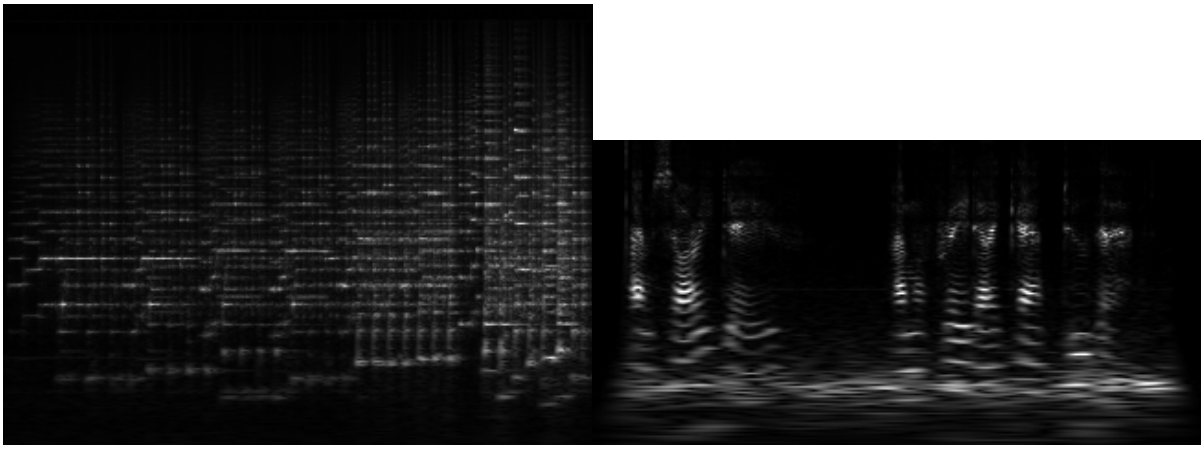


FIGURE 47 – SPECTROGRAMS TO BE RESYNTHESISED USING ARSS (ROUZIC, 2009)

However although the program is very impressive and fits very well with our motivation as far as we are concerned we want an exact copy of the audio our program segmented. Unfortunately as you can hear from the recordings lots of resolution is lost and the recordings lose quality. This is due to the same problem we encountered with our prototype in that going from audio to spectrogram loses lots of information, this is then increased when we go back into audio. On further testing we also found that the resolution of the spectrograms MATLAB produces are very low, adding to the loss of resolution.

Fortunately we found a simpler way, which does not require any additional programs and maintains all the audio quality. Our segmentation lines show the index of the STFT. This index relates to time and as such by looking at the minimum and maximum index in each segment we can find the start and end times for each segment. This returns the precise time in seconds. However although we have the start and end time for each segment, from our initial research in chapter 2 we know that the audio file is stored in samples appose to seconds. This is a relatively easy conversion as we know the sample rate, which represents the amount of samples per second. We can therefore work out the start and end sample by:

$$StarEnd\ Sample = StartEnd\ time\ (seconds) * sample\ rate \qquad (20)$$

The audio can now be easily exported using MATLAB's '`wavwrite`' function.

5.2.5. INITIAL TESTING AND MINOR IMPROVEMENTS

Following our IID methodology we where constantly improving and developing our system. This section we will look at some of the improvements and changes, which were added to the program throughout the development.

Although we have been able to successfully extract the audio segments, for the user to find, navigate and listen to them we require a simple user interface (UI). The UI needs to be able to list all of the segments with their appropriate index and allow the user to download these segments. With these requirements in mind we decided to create a simple HTML webpage, which can list all of the segments the program found and display them in a list. Each segment can then provide a hyperlink to the audio file where the user can download the audio file.

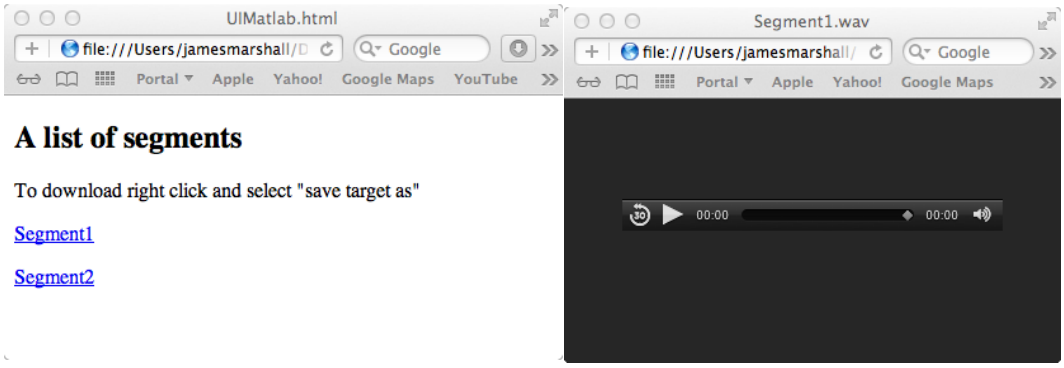


FIGURE 48 - BASIC UI

Figure 48 shows the result of UI. Although very basic it serves its purpose extremely well. It allows the user to go through each segment, decide whether or not to download it. However on testing a few problems where encountered. Our first UI required manual updating, which meant that the program operator had to manually input each segment reference i.e.:

```
<p><a href="Segment1.wav">Segment1</a></p>

<p><a href="Segment2.wav">Segment2</a></p>
```

However this is counterproductive as the reason we require a UI is to minimise the amount of effort the user needs put in. We therefore wrote a MATLAB script, which automatically updates and creates the HTML file to include all the segments each time the program is run. This solved

the problem and successfully updated the HTML file with the new segments and links on each iteration of the program.

After further examining some of our test files we noticed that although we where successfully able to extract the main essence of a key point we often missed some of the crucial sound at the attack and release (audio data CD).

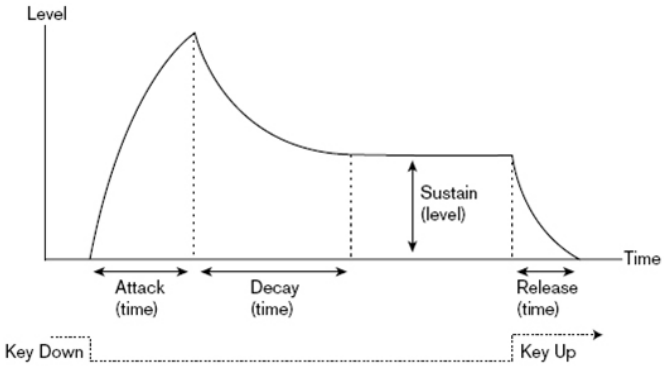


FIGURE 49 – SOUND ENVELOPE - ATTACK, DECAY, SUSTAIN AND RELEASE (HEADPHONES.COM, 2012)

Figure 49 shows the envelope of a sound. Every sound has an envelope and this describes its characteristics. For example a snare drum will have a very sharp attack, whereas a violin or piano with have a much smoother attack. There are four key properties, which describe the envelope; attack, decay, sustain and release. The envelope shown in figure 49 is typical of a piano key being pressed.

To further investigate we decided to look at the ordinal sound wave. Figure 50 shows a magnified image of the key feature we wish to extract. Note time is shown in x axis and amplitude in the y. From this we can see that the feature has a very sharp attack however relatively long release.

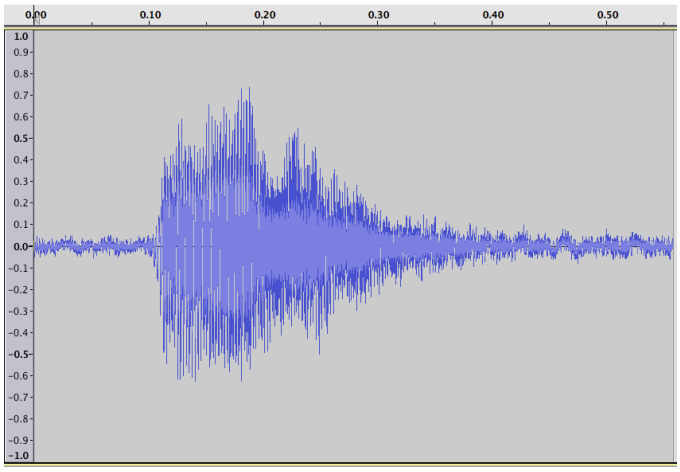


FIGURE 50 - MAGNIFIED AUDIO FEATURE

By looking at this we predict that our algorithm will have picked up the sharp attack extremely well, as there is a great amount of change however during the release the amount of change becomes a lot less and as such will have probably fallen below our threshold. To check this we can examine our MATLAB plots, which show the threshold and the amount of change (figure 51).

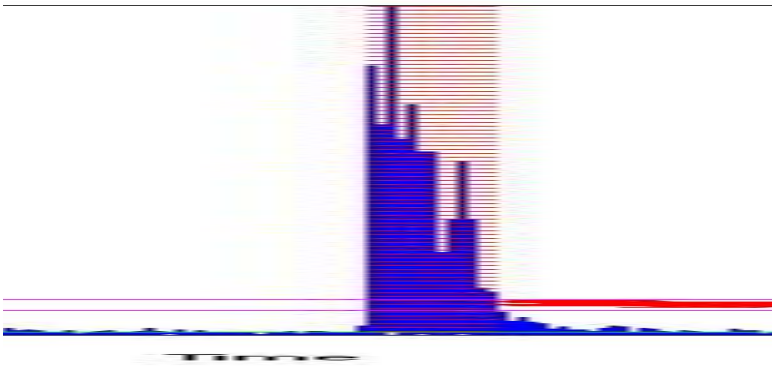


FIGURE 51 - AUDIO FEATURE - AMOUNT OF CHANGE

The vertical red lines in the figure above shows the timesteps where our system has identified enough change to mark a feature. We can see that although it picks out the initial attack as predicted, the features release falls below the purple threshold line and as such our system does not capture it. This explains why when we listen to the extracted feature, we miss some of the release information. The attack information lost can also be explained in the same way however this occurs across a series of hundredths of seconds and as such is just visible as the slight blip before the large spike. Our system can capture the end of the attack, sustain, decay and start of the release however misses the very beginning of the attack and end of the release.

To help stop this effect, we decided to add a tolerance into the audio extraction phase. Instead of extracting the feature about the exact start and end points, our system added a certain amount of time to the start and end, which is proportional to the length of the segment. This proved to be extremely successfully however on larger segments we noticed that an unnecessary amount of time was added. Instead we found that by simply adding 0.5 seconds to the start and end we were able to successfully capture the whole feature and minimise unwanted excess. The results of this addition can be heard on the data CD.

There was only one slight problem with this method that was discovered when testing on audio files which contained features very close to the start or end of the track. If the start of the feature is less than 0.5 seconds from the start of the track an error occurs as the program tries to extract audio, which does not exist. To stop this a conditional if statement was added to check and make sure the program does not try and extract audio which does not exist.

6. CHAPTER 6 – TESTING, FURTHER WORK AND CONCLUSIONS

This chapter will analyse the performance and functionality of the system. However before we can accurately analyse the performance we need to consider a use. In chapter 1 we discussed how our system has many potential uses, ranging from security to beat tracking. However we are mainly concerned in aiding soundscape artists.

A soundscape can be thought of a piece of music/audio, which contains a collection of sounds, which describe acoustic environments, often ones that we are subjected to throughout our lives (LaBelle, 2006). Soundscapes can provide the contextual references that contribute to our feelings, belonging and place (Mydlarz). They are comparable to the audio equivalent of a landscape painting documenting and describing a scene. An example could be a picture of a busy market scene. We would see customers hustling and bustling, vendors touting and depiction of the weather. This could be converted into audio, allowing the user to hear what is going on. Another popular use of soundscape is to provide a historical documentation of a particular location allowing a listener to revisit a scene back in time.

Soundscapes are generally created artificially by taking many or a long recording and combining the key elements into a single piece of music. For example a soundscape artist wishing to capture the sounds of a seaside town may place a microphone outside a window, leave it for a couple of days then manually go through the audio and pick out the key sections they wish to add to their composition. This of course takes a lot of time, and can be rather tedious. Our aim as discussed in chapter 1 is too automatically find and extract the key segments from within a piece of audio.

Although in the previous chapters we have seen we have successfully been able to accomplish this in our test examples, in order to validate and show we have not over fitted our system we derived a testing scheme which focuses on extracting moments of change from within a soundscape environment. In order to thoroughly test the system a range of audio files would be required. These would have to contain different information and different settings. We also wanted the clips to have varying amounts of background activity, which could confuse and impact the segmentation. With this in mind we came up with a list of 10 short audio segments. The audio files range in length between a couple of seconds to approximately 90 seconds. The reason for having short audio segments is to ensure that we can run tests multiple times in a reasonable amount of time with the computing power available. Another major reason for having short files is it allows us to ask a panel to tag the audio within a reasonable timeframe. The following files where selected for the validation:

1 - Urban Roadside: (Cars passing by, car pulls up and someone opens and closes the door, it then departs)

2 - Moving Objects: (Scrapes as someone repositions a wooden object on a floor)

3 - Urban church bells: (ambient recording with church bells chiming at regular intervals)

4 - Garden: (Birds singing in the trees, interrupted by a couple of loud crows)

5 - Operating a lock: (Quiet scene, which is interrupted by a reverberant lock mechanism)

6 - Opening a door: (Quiet scene, person fumbles keys and then opens a door)

7 - Explosion: (Distant recording of fireworks)

8 - Nature at night: (Constant background hum of insects with a young owl intermittently screeching, a dog barking can also be heard)

9 - Thunder: (Constant rain with occasional claps of thunder)

10 - Burglary: (Quiet scene interrupted by a shattering of glass)

The above audio files all contain at least one possible change however as the idea of change can be subjective a short survey was produced in order to tag the files. In the survey the participant was asked to listen to each file and note down the start and end time of each segment they heard. A blank survey is include in appendix D. The results of the surveys where then analysed to find the amount of segments, and their corresponding start/end times for each of the audio files. The full results for this can be seen in appendix E table.

A total of 20 surveys where taken, and within these surveys we found that most of the results directly correlated and there was little argument for how many segments were present and where these segments start and end. There where however a couple of clips which did have a bit of debate. The clips with debate, where generally the ones that had more background noise, and possible segments. Note that in the case of debate the mode average was taken.

If we take a closer look at clip 8, featuring a constant background hum of insects with a young owl intermittently 'screeching'. The most common segment extracted were individual 'screeches'. However some members of the panel decided that a segment could include multiple 'screeches' therefore there was a discrepancy between the amounts of segments present. We found that in general the starts of segments were clearly identified by the participants however the end varied. This was also true on clip 9 where thunder claps where the identified segment.

The participants all agreed when the segment started however the end became ambiguous. The table below summarises the number of segments present in each of the validation clips.

Clip Number	Amount of segments present
1	4
2	5
3	7
4	2
5	2
6	4
7	3
8	8
9	4
10	2

TABLE 4 - AMOUNT OF SEGMENTS PRESENT IN EACH CLIP

6.1. RESULTS

In the previous section we talked about how we obtained our testing data to use for our validation. This section we will look at and analyse the results of running the testing data through our system. We will preform a number of tests, in which we will first look at how many segments our system splits each clip into, then further analyse how it has split each clip.

As we need to make sure that we conduct a fair test, the system parameters will all be kept constant. They will be initially set to the conditions that where used and tested in the development stage:

Segmentation threshold: 3 standard deviations

Kernel Separation: 8000 samples

Attack + Release: Not enabled

Table 5 summarises the number of segments present in each of the validation clips and how many segments where detected by our system. The accuracy percentage for each clip and the whole system is also shown.

Clip Number	Amount of segments present	Amount Detected	Accuracy %
1	4	5	80
2	5	5	100
3	7	7	100
4	2	2	100
5	2	2	100
6	4	4	100

7	3	3	100
8	8	9	88.89
9	4	5	80
10	2	2	100
Total	**41**	**44**	**93.18**

TABLE 5 - VALIDATION ACCURACY, AMOUNT OF CLIPS PRESENT

From table 5 it is clear that our system has successfully managed to segment the audio into the correct amount of segments with an overall accuracy of 93.18%. Although this information suggestions that our system is successful we require a closer examination of these segmentations to validate the results. This is required as it could be the case that although we have been able to segment the correct number of segments they could be in the wrong location.

There are a number of options we have to analyse the position of the segments. One of the ways we can visually analyse the clips is to look at the amount of change and segmentation graphs. These graphs allow us to visually see exactly how much change is occurring against time. They also show the segmentation threshold, which allows us to see where our system has segmented the audio. The full results can be seen in appendix F, however for the purposes of analysis figure 52 shows the results for clip 3.

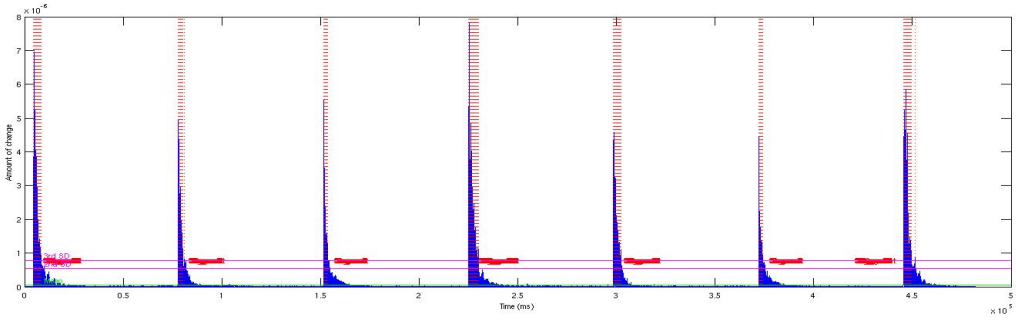


FIGURE 52 - CLIP 3 CHANGE & SEGMENTATION

We can see that for some clips where little background noise is present such as clip 3 in figure 52, the position of the segments can be easily visualised using these graphs. However when more noise is present and the segments are not so well defined, such as clip 8 (figure 53), these graphs become more complicated to read. If we look at this graph the segmentation becomes a lot harder to visualise. An easier method to quantitively assess the results of our system is to analyse the timestamps produced by our UI and compare these to the tagged results obtained from our survey.

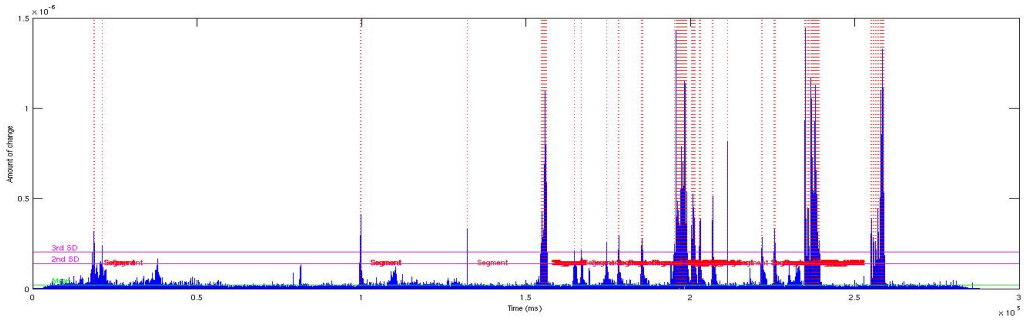


FIGURE 53 - CLIP 8 CHANGE & SEGMENTATION

The UI results are shown in appendix G. For each one of the clips the user is shown a list of numbered segments with their associated timestamps. We have then taken these timings and compared them to the user tagged files which can be seen in appendix E. Note that the results for the system segmented approach have been rounded to the nearest second. This was done as the time resolution of the media player that the participants used didn't allow for a higher degree of accuracy.

To accurately decide whether the segments found by our system correlate to the segments tagged, we manually compared the start and end time of each segment. We then noted down any missed time, missed segments and any additional segments the system found which where not originally tagged. The full results for this can be seen in appendix G. Table 6 below summarises the key findings.

	Quantity	**Accuracy percentage**
Total amount of segments found	44	N/A
Amount of tagged segments missed	5	12.2
Amount of segments tagged correctly	36	87.8

Amount of additional segments found	3	6.82
Tagged segments covered by multiple system segments	5	11.36
Amount of correct start times	27	65.85
Amount of correct end times	15	36.59

TABLE 6 - KEY FINDINGS

From table 6 above we can see that a total accuracy of 87.8 % was achieved. This figure shows that our system is preforming much better than if we where to randomly select short segments of audio. By taking a closer look at the results we noticed a few things. In general the system was better at finding the attack of the segment than the release. We also noticed that although the accuracy of this appears quite low in comparison with the total accuracy, human error in tagging could account for this as the resolution of the media player used only allowed for +/- 1 second. By looking at the results again and adding the human error tolerance we in fact only failed to locate 1 attack, achieving a total accuracy of 97.22%. If we added the tolerance to the release we manage to increase the accuracy from 36.59% to 69.44%, which is still a lot lower than the attack however greatly increased.

To investigate this we took a closer look at some of the clips, which featured constant missing releases. If we look at clip 3 (figure 52), we successfully managed to locate each attack however consistently missed the release. By looking at figure 52 we can see that the envelope of each segment is consistent and features a sharp attack but long decay. As we saw in our initial testing when this occurs the delay falls below the threshold and is therefore cut short. As each clip is different we kept the settings of the system consistent, however as we can see that the missing time is consistent, it would be possible to adjust the settings to compensate for the long release and tailor the system to the specific need.

Other interesting things we noticed is that in clips, which had more background noise, we sometimes located tagged segments with multiple system segments. This suggests that the kernel function was set too low as it split the tagged segment into multiple segments. Therefore segments with a higher background noise tended to need a larger kernel.

The final method for analysing our results is not quantitative and rather relies on human perception. Instead of looking at the specific times, we can listen to segments extracted and compare these to the original file. The initial results, which were used in the analysis above, can be heard on the data CD. From listening to these we can hear mixed results. In most of the clips we can clearly hear that our program is successfully locating and extracting the features present. Similar to our testing, it is missing lots of the sound envelope and not extracting the full feature. This is also accentuated when the feature has a very sharp envelope such as a explosion. In this case we are simply locating it rather than extracting any audio.

The data CD also contains the validation audio with the envelope function added. One of the major benefits of listening to the sounds with the added envelope function is that rather than just locating many of the sounds we can listen to them back and confirm they are a feature. An example of this is if we listen to clip 7, before the envelope function the segments were extremely short and barely anything was audible. It was nearly impossible to tell what feature was being extracted. After however, the fireworks captured become audible and the feature could be confirmed.

Although we cannot obtain any quantitative data from listening to the audio files, it was clear when listening to the files that the system was preforming correctly and always managed to find areas of change. By listening to the segmented audio it has drastically helped us improve the system. It is extremely hard to imagine the audio by simply comparing times. As we heard during our testing, the envelope of the sound is crucial, and failing to capture hundredths of seconds can really change how the audio segments sound. Comparing the unprocessed segments with the segments with the added envelope function also shows this.

6.2. FURTHER WORK

This section will briefly discuss some possible future directions of the project. We will also look at some concepts and ideas that have been implemented and are in development.

One of the limitations of the system is that it can only be run on systems with MATLAB. As such the system is not very portable. This means that if we want to produce summaries in real time, or distribute the system, we would need to recode the system. C is currently one of the most popular languages (LangPop, 2012), and it would give the greatest chance to allow people to carry on the work in this project. We would therefore highly consider porting the system to C. C also offers other advantages such as code portability and efficiency. Efficiency is particularly key if we were to implement the system in real time, or even when calculating the results for a

particularly large audio file. There are also freely available cross-platform audio library's, which can help with the handling of audio data such as OpenAL (CreativeLabs).

We saw in chapter 1 that there were alternative uses for our system. With more time we could start experimenting with using the system as a potential beat tracker. This would require testing the system against current solutions. During our validation stage we tested the system using soundscape audio, however the system would handle popular music and audio with a definite beat just as well. We believe that it could work as a beat tracker as during our prototype stage (chapter 4) we saw that it was successfully able to identify beats of a short pop song (figure 16).

One of the main things that we have noticed throughout this project is that although you can set very general parameters, which are able to achieve reasonable results. The system can always be modified and tailored towards a particular use and as such improve its accuracy. We believe a way of automatically identifying the style and type of audio, which is being processed, could be introduced. Looking back to our initial research we have seen authors such as (Deshpande, Singh, & Nam, 2001) classify music into categories. We could then use this information to set parameters within the program.

6.2.1. CLUSTERING

Another advancement we have in progress is applying a clustering algorithm to our features. By applying clustering to our features we can improve the functionality of the system.

The algorithm we used was 'K-means' as it is relatively simple to implement and can work on large datasets. K-means is an iterative process, which initially randomly assigns K centroid locations, assigns the nearest points to those locations then updates the centroid location with the mean value of the associated points. The algorithm then repeats until it converges. An example of a converged dataset is show in figure 54.

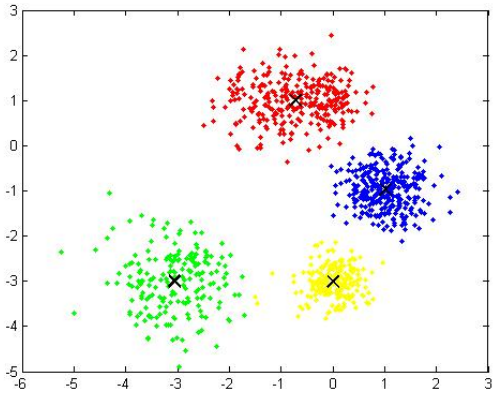
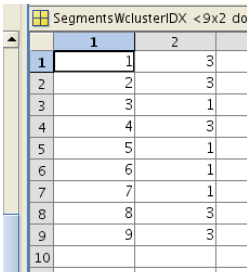


FIGURE 54 - AN EXAMPLE OF CLUSTING 2D DATA INTO 4 CLUSTERS

By sorting our data into K clusters we hope to develop a hierarchy of sounds, which summarise the audio. For example if we specify one cluster, the centroid location will provide a feature, which is most representative of all features present. As such we could obtain a hierarchy of sounds, which range from very general to very specific. However if K is increased too far we will over fit the data. The clustering algorithm will also help to reduce redundancy in the data, as it will sort clips that are most similar together. We can also use it to find similarity within segments. Figure 55 shows 9 segments, which have been clustered into 2 groups. Note the left column shows the segment number and right shows the cluster assignment.


SegmentsWclusterIDX <9x2 do

	1	2
1	1	3
2	2	3
3	3	1
4	4	3
5	5	1
6	6	1
7	7	1
8	8	3
9	9	3
10		

FIGURE 55 - SIMILARITY IN SEGMENTS USING K-MEANS

6.2.2. USER INTERFACE

The UI is under constant development and we are currently working on ways in which we can improve it. One of the first developments made was to include the segment timestamp. As we saw previously our segments can sometimes miss some of the onset / decay information the user requires. By adding a timestamp it will allow the user to manually go back into the file and locate the key section of audio highlighted by our program. The user can then manually extract the required attack and delay. As different users will have different requirements by adding a timestamp, we will improve the usability and functionality of our system. With regards to different user groups if we were to look at the application of security, the timestamp could be

more important than the audio segments. Instead of manually going through hours to days of security footage, by locating the time at which change occurs, security firms will be able to quickly and more efficiently hone in on the cause of such change. Figure 56 shows an example of the UI with the timestamp addition.

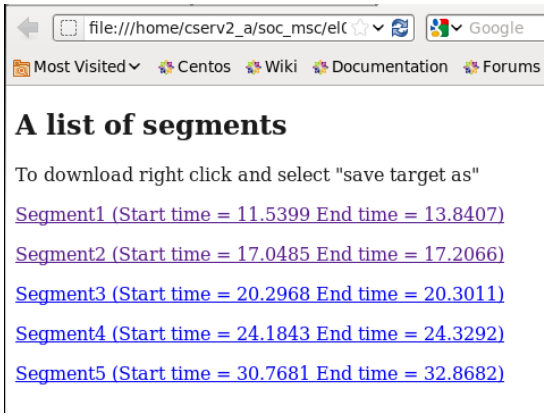


FIGURE 56 - UI WITH TIMESTAMP

You can see from figure 56 that we have include both the start and end time. We have also made sure that the time is given to the highest possible degree of accuracy.

Further updates to the UI could include the addition of user controls and parameters. By adding these it will allow the user to improve and adjust the performance of the system based on the required task. These could include:

- Spectrogram parameters (window length, time step & overlap)
- The segmentation threshold (reduce to increase the amount of segments)
- A cap on the amount of segments returned (switch between returning N amount of segments or segmentation based on threshold)
- The attack and delay added to segments
- The window size which dictates segment separation

6.3.CONCLUSIONS

The project started out with the motivation that if we visually analyse spectrograms we are able to located key sections and segments. We showed in section 1.1 that by looking at the spectrogram of a pop song we are able to locate beats and areas of change. Continuing from this motivation the aim of this project was to analyse techniques to detect change within a sonic environment such as a soundscape. Once the change was detected we would extract the representative segments in order to provide an audio summary of the audio file. This report has

presented the design and development of a system, which is able to accomplish this by using spectral change as a feature.

6.3.1. REVISITING MINIMUM REQUIREMENTS

Here we will look briefly back to our minimum requirements and point to the locations in the project where they have been covered:

1) ***Implement a means of handling audio data*** – This was first looked at during chapter 3 and continued throughout the project. We decided to MATLAB as a means to handle audio data. MATLAB was chosen as it allows a rapid prototyping facility with prebuilt libraries capable of accepting audio files.

2) ***Design and implement a spectra based technique to analyse an audio signal –*** Image analysis tools where initially used in chapter 3, however our final system used the Short-time Fourier transform which is discussed in section 5.2.1

3) ***Design and develop a technique to detect audio change –*** The way we detected audio change was first developed in the prototype in chapter 3. We measured the change between neighbouring timesteps, and then applied a threshold.

4) ***Extract the key features of a sound segment–*** When we looked at extracting the key features, which make up the sound segment we visited the idea of thresholds that are discussed in section 4.33 and 5.22. We also looked at methods such as the mean-shift algorithm in order to demine which samples constituted the segment.

5) ***Produce a summarisation of an audio track using short sound segments.*** – A basic UI was produced for this purpose and is discussed in detail in section 5.2.5. It was later improved and modified in section 6.2.2

6.3.2. FINAL THOUGHTS

Throughout this report we have shown how spectral change can be used to extract features in order to determine the areas, which hold the most change. Our system has successfully managed to locate and extract sound segments from 10-tagged audio files with an accuracy of 87.8%.

Our system favours sounds with sharp attacks & sharp release, as sounds with these envelopes provide areas of sudden change, which prove simple for our system to detect. The system can be

modified to extract segments, which are found in sounds with shallower envelopes though currently does not have any way of detecting them.

One of the most crucial finding is that each environment needs parameter tuning. During validation we set the parameter of our system to ensure fair testing however if this product was being marketed we would recommend that calibration tests are conducted before extracting final results.

Finally although we extract the audio segments into individual 'wav' files we believe that the most crucial information our system returns are the audio timestamps. As the system does not return the exact segment due to envelope problems, giving the approximate location of the start and end gives the soundscape composer the required information to speed up manual extraction. It also gives them the freedom to modify the segment. The audio thumbnail is still useful however as it allows the composer to quickly hear summaries of sections which they can then segment.

7. BIBLIOGRAPHY

Aucouturier, J., & Sandler, M. Segmentation of musical signals using Hidden Markov Models . *110th Audio Engineering Society Convention (AES)* , (p. 2001). Amsterdam.

Abdallah, S., Noland, K., Sandler, M., Casey, M., & Rhodes, C. (2005). Theory and evaluation of a Bayesian music structure extractor. *6th International Conference on Music Information Retrieval*, (pp. 420-425).

Alejo2083. (2010, 6 2). Wikimedia Commons. *STFT* .

ARSS. (2009, 2 23). *The ARSS*. Retrieved 7 20, 2012 from arss.sourceforge.net

Bartsch, M., & Wakefield, G. (2005). Audio thumbnailing of popular music using chroma-based representations . *IEEE Transactions on Multimedia*, (pp. 96-104).

Bartsch, M., & Wakefield, G. (2001). To catch a chorus: using chroma-based representations for audiothumbnailing . *IEEE Workshop on the Applications of Signal Processing to Audio and Acoustics .* New York.

Bello, J., Daudet, L., Abdallah, S., Duxbury, C., Davies, M., & Sandler, M. (2005). A Tutorial on Onset Detection in Music Signals. *IEEE Transaction on speech and audio processing , 13* (5).

Casey, M., & Slaney, M. (2006). The importance of sequences in musical similarity. *IEEE International Conference on Acoustics, Speech and Signal Processing.* Toulouse.

Chai, W. (2005). *Automated analysis of musical structure.* PhD Thesis, Massachusetts Institute of Technology .

Cheng, Y. (1995). Mean shift, Mode Seeking and Clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (pp. 790-799).

Clavel, C., Ehrette, T., & Richard, G. (2005, July). Event detection for an audio based surveillance system. *IEEE ICME* .

Cockburn, A. (2008, May). Using both incremental and iterative development. *CrossTalk: Defense Software Engineering* , 27-30.

Collins, R. (2003). *Mean-shit Blob Tracking through Scale Space.* Carnegie Mellon University.

Comaniciu, D. (2002). Mean shift: a robust approach toward feature space analysis. *Pattern analysis and machine intelligence, IEEE Transactions on , 24* (5), 603-619.

Cooper, M., & Foote, J. (2002). Automatic music summarization via similarity analysis. *IRCAM.* Pompidou.

Cooper, M., & Foote, J. (2003). Summarizing popular music via structural similarity analysis. *Workshop on Applications of Signal Processing to Audio and Acoustics*, (pp. 127-130). New York.

Costa, Y., Oliveira, L., Koerich, A., & Gouyon, F. (2011). Music Genre Recognition Using Spectrograms. *International Conference on Systems, Signals and Image Processing.* Sarajevo.

CreativeLabs. (n.d.). *OpenAL*. Retrieved 6 15, 2012 from Creative Labs: http://connect.creativelabs.com/openal/default.aspx

Dennis, J., Tran, H., & Li, H. (2011). Spectrogram Image Feature for Sound Event Classification in Mismatched Conditions. *IEEE signal processing letters , 18* (2).

Deshpande, H., Singh, R., & Nam, U. (2001). Classification of music signals in the visual domain. *COST G-6 on Digital Audio Effects.* Limerick.

Fukunaga, & Hostetler. (1975). The Estimation of the gradient of a Desity Function with Applications in Pattern Recognition. *IEEE Transactions on Information Theory*, *21*, pp. 32-40.

Foote, J. (1999). Visualizing music and audio using self-similarity. *ACM Multimedia 99*, (pp. 77-80). Orlando.

Foote, J. (2000). Automatic audio segmentation using a measure of audio novelty. *IEEE International Conference on In Multimedia and Expo*, *1*, pp. 452-455.

Gerosa, L., & Valenzise, G. (2007, September). Scream and gunshot detection in noisy environments. *EURASIP* .

Goto, M. (2003). A chorus-section detecting method for musical audio signals. *IEEE International Conference on Acoustics, Speech and Signal Processing , 5.* Hong Kong.

Grey, J., & Gordan, J. W. (1978). Perceptual effects of spectral modifications on musical timbres. *Journal of the Acoustical Society of America.*

Haykin, S. (1991). *Advances in Spectrum Analysis and Array Processing* (Vol. 1). Prentice-Hall.

Headphones.com. (2012, 8 11). Retrieved 8 11, 2012 from headphones: http://www.headphones.com/headphone-guide.html

Howard, D., & Angus, J. (2007). *Acoustics & Psychoacoustics.* Focal Press.

Lu, L., Wang, M., & Zhang, H. (2006). Extraction of high-level musical structure from audio data and its application to thumbnail generation. *IEEE International Conference on Acoustics, Speech and Signal Processing.* Toulouse.

LaBelle, B. (2006). *Background Noise : Perspectives on Sound Art.* Continuum International Publishing Group.

LangPop. (2012, 4 13). *Programming Language Popularity*. Retrieved 8 12, 2012 from LangPop.com: http://www.langpop.com/

Larman, C. (2003). *Agile and Iterative Development: A Manager's Guide.* Reading, MA.

Levy, M., Sandler, M., & Casey, M. (2006). Extraction of high-level musical structure from audio data and its application to thumbnail generation . *International Conference on Acoustics, Speech and Signal Processing.* Toulouse.

Logan, B., & Chu, S. (2000). Music summarization using key phrases. *IEEE International Conference on Acoustics, Speech and Signal Processing.* Istanbul.

National Instruments. (2009). *STFT Spectrogram VI*. Retrieved 6 20, 2012 from National Instruments: http://zone.ni.com/reference/en-XX/help/371361E-01/lvanls/stft_spectrogram_core/#details

Mydlarz, C. (n.d.). *IMPRINtS*. Retrieved 8 1, 2012 from Internet and Mobile technologies for a Public Role In Noise Surveying: http://www.csesalford.com/drumm/Charlie/Soundscape.htm

Maddage, N., Xu, C., Kankanhalli, M., & Shao, X. (2004). Content-based music structure analysis with applications to music semantics understanding . *12th Annual ACM International Conference on Multimedia*, (pp. 112-119). New York.

Marshall, D. (n.d.). *Implications of sample rate and bit size.* Retrieved June 20, 2012 from www.cs.cf.ac.uk/Dave/Multimedia/node150.html

Morse, B. (2000). *Brigham Young University*. Retrieved July 10, 2012 from http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/MORSE/threshold.pdf

Olson, H. (1967). *Music, Physics and Engineering.* Courier Dover Publications.

Ong, B. (2005). *Towards automatic music strucural analysis: identifying characteristic within-song excerpts in popular music.* PhD Thesis, University of Popeu Fabra.

Otsu, N. (1979). A threshold selection method from gray-level histograms. *IEEE Trans Sys, Man, Cyber* , 62-66.

Pauws, S. (2004). Musical Key Extraction From Audio. *Ismir* .

Paulus, J., & Klapuri, A. (2006). Music structure analysis by finding repeated parts. *1st ACM Workshop on Audio and Music Computing for Multimedia* , (pp. 59-68). Santa Barbara.

Peeters, G. (2004). A Large set of audio features for sound description.

Peiszer, E. (2007). *Automatic audio segmentation: Segment boundary and structure detection in popular music.* PhD Thesis, Institut für Softwaretechnik und Interaktive Systeme.

Peiszer, E. (2007). *Automatic audio segmentation:Algorithm, experiments and evaluation.* Master's Thesis, Vienna University of Technology, Vienna.

Schoerkhuber, C., & Klapuri, A. (2010). Constant-Q transform toolbox for music processing. *7th Sound and Music Computing .* Barcelona.

Rumsey, F., & McCormick, T. (2006). *Sound Recording: An Introduction.* Focal Press.

Rabiner, L., & Juang, B. H. (1993). *Fundamentals of Speech Recognition.* Prentice-Hall.

Rhodes, C., Casey, M., Abdallah, S., & Sandler, M. (2006). A Markov-chain Monte-Carlo approach to musical audio segmentation . *IEEE International Conference on Acoustics, Speech and Signal Processing .* Toulouse.

Tzanetakis, G., & Cook, P. (1999). Multifeature audio segmentation for browsing and annotation. *Applications of signal processing to audio and acoustics*, (pp. 17-20). New York.

Typke, R., Wiering, F., & Veltkamp, R. (2005). A survey of music information retrieval systems. *ISMIR* , 153-160.

Thirumuruganathan, S. (2010, 4 1). *Introduction to mean shift algorithm.* Retrieved 7 20, 2012 from https://saravananthirumuruganathan.wordpress.com/2010/04/01/introduction-to-mean-shift-algorithm/

Wold, E., Blum, T., Keislar, D., & Wheaton, J. (1996). Content-Based Classification, Search and Retrieval of Audio. *IEEE Multimedia* , 27-36.

Xu, C., Zhu, Y., & Tian, Q. (2002). Automatic music summarization based on temporal, spectral and cepstral features. *International Conference on Multimedia and Expo*, (pp. 117-120).

Yu, G., & Slotine, J. (2009). Audio Classification from Time Frequency Texture. *IEEE international Conference on Acoustics, Speech and Signal Processing* , 1677-1680.

APPENDIX A - PERSONAL REFLECTION

Coming from a background in music, acoustics and electronics, AI was a challenging and somewhat daunting challenge. However throughout the taught section of the course I managed to really get involved and learn the skills, which were needed for this project.

As I have a background in music, I decided that I would come up with my own project. I felt by doing this I would embark on a project, which I would enjoy, and potentially aid in my future career plans. After discussing my initial idea with my supervisor, we decided to combine aspects of my idea with an idea he had to create a title as presented in this report. Having that initial input into the project title was absolutely crucial as it helped motivate me throughout the project. It also helped keep my interest and drive through times of complication. I would recommend to future students above everything else, to pick a topic they enjoy. I found that the project drastically changed from my initial view, however as it was based around a topic I was familiar with and enjoyed I kept the motivation throughout.

The hardest part of the project was getting started. After finishing the exams in June it was very hard to build the motivation to get straight back into work and complete the background reading. Another problem I faced was simply being off put by complicated looking mathematics. I found myself on occasions simply staring at equations trying to get my head around them without any progress. A solution was found however, and it was simply 'throw yourself into the deep end'. Instead of just reading I found the best method was to create a program, change some of the variables and examine the output. MATLAB was a great aid in this, as often the equations were already built into functions and the output could be easily read.

Throughout the project I kept regular notes in the form of a lab book, I would highly recommend this to future students. The lab book allowed me to remember parts of the project, which I would have defiantly forgotten, during the write up. It also acts as a written backup in case of file corruption or loss (which happened on occasion). However even with the lab book I still often found it difficult in to write my ideas in a precise and comprehensible way. It was often very clear in my mind what I had accomplished however translating that to paper was hard. I fortunately stumbled on this problem early on in the project and found that the best way was to write up drafts of sections as I went along. This way I could note down the detail, which I may forget to include later on. It is also worth noting that the report has taken me roughly 200-250 hours to write, which is a large proportion of a 600-hour project.

I chose to use Microsoft Word to format my report, as I was not familiar with LaTeX. MS Word also offered many facilities that I would find useful when formatting my project. These facilities

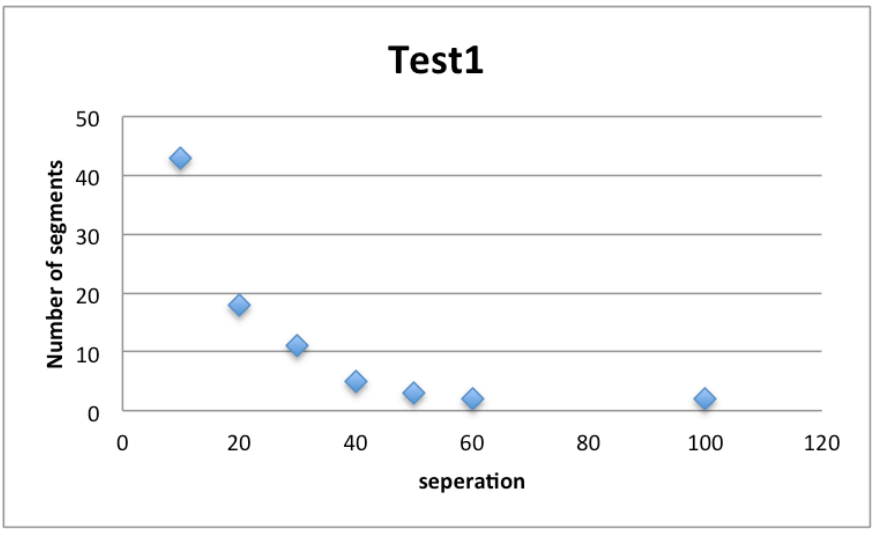
include automatic referencing tools, similar to 'Ref works', spell/grammar checkers and automatic chapter/section numbering. I slightly regret not using this project as a means to learn LaTeX however my familiarity with Word was defiantly useful and saved me a lot of time. I would defiantly recommend to students who are not familiar with LaTeX to use MS Word as it offers a great environment with useful tools which help to produce a professional looking report while saving many hours which would have been required to learn LaTeX.

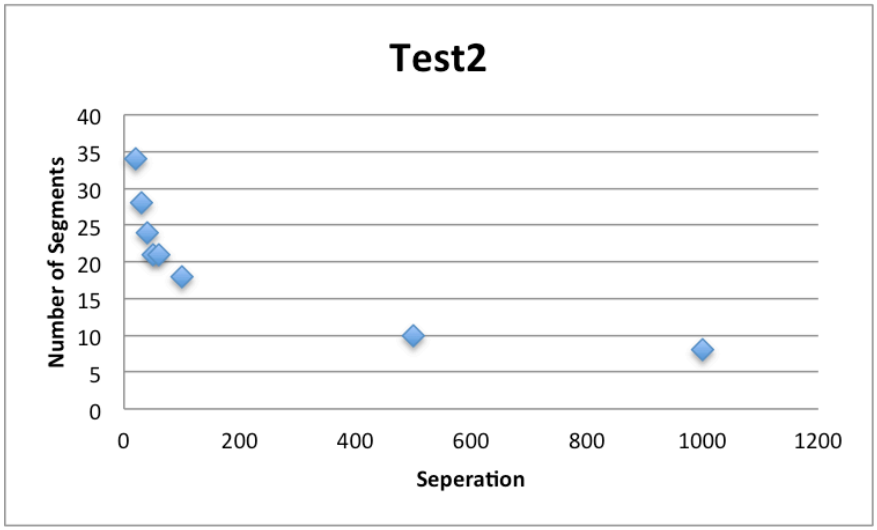
Finally I would like to say thank you to my supervisor for all the support and that I have really enjoyed this project. More specifically I really enjoyed the fact that the skills used in this project were developed during the taught section of this course. Although my background in music and acoustics helped with my basic my understanding of the terminology, my AI, modelling and analysis techniques where developed during the taught section.

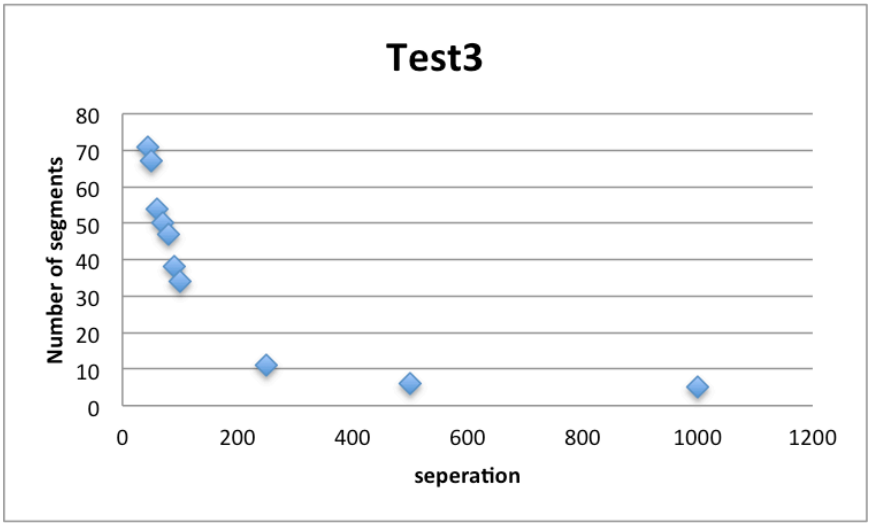
APPENDIX B – INTERIM REPORT

Please refer to attached document located at the end of this report.

APPENDIX C – SEGMENT SEPARATION TEST







Automatic Audio Segmentation based on Spectral Change

About this research

This research aims to create a way to automatically segment and extract moments of change from within an audio file. The main purpose of this project will be to help aid soundscape artists and audio documenters find and extract the moment whereby something different / interesting happens. This would also be of interest to security companies trying to locate anomalies and areas within a audio track which are different to the normal.

To achieve this a prototype program will be engineered and will be tested on a series of small audio clips which have been selected to represent a range of soundscape scenarios. The motivation for this project came from visually analyzing spectrograms (fig 1) . The spectrograms provide a visual representation of an audio file and we noticed that moments of change and difference could be clearly seen. We are therefore using spectral change to try and locate these differences automatically.

Illustration 1: Spectrogram

Your Contribution to the project

We need your help to tag audio test files in order to test the accuracy of our system

Please note that the data you provide to us will be kept anonymous.

We will provide a series of 10 audio clips ranging from a few seconds to approximately a minute. You are expected to listen the audio clips and write down the start and corresponding end time where audio change occurs (note that there may be **multiple** or even **no areas** in each clip). We expect you to write down your answers on this sheet in the table below. You will be able to listen to audio clip multiple times and have access to pause/rewind the clip as you deem necessary. Please feel free to add comments if you wish.

A example clip and answer will be provided

The experiment

Audio Clip name	Start Time	End Time
*"**Example**"*	*0:07* *0:10*	*0:09* *0:11*
Clip 1 Comments -		
Clip 2 Comments -		
Clip 3 Comments -		
Clip 4 Comments -		
Clip 5 Comments -		

Clip 6 Comments -		
Clip 7 Comments -		
Clip 8 Comments -		
Clip 9 Comments -		
Clip 10 Comments -		

We would like to thank you for your contribution.

APPENDIX E – TAGGED / SYSTEM SEGMENTS & TIMES

Tagged Clip Segmentation

Clip 1

Segment number	Start time	End time
1	00:10	00:14
2	00:16	00:20
3	00:24	00:25
4	00:32	00:36

Clip 2

Segment number	Start time	End time
1	00:00	00:01
2	00:03	00:04
3	00:06	00:07
4	00:08	00:10
5	00:11	00:12

Clip 3

Segment number	Start time	End time
1	00:00	00:04
2	00:10	00:14
3	00:20	00:24
4	00:30	00:34
5	00:40	00:44
6	00:50	00:54
7	01:00	01:04

Clip 4

Segment number	Start time	End time
1	00:03	00:06
2	00:07	00:09

Clip 5

Segment number	Start time	End time
1	00:02	00:04
2	00:07	00:08

Clip 6

Segment number	Start time	End time
1	00:10	00:11
2	00:12	00:13
3	00:15	00:16

Clip 7

Segment number	Start time	End time
1	00:10	00:11
2	00:11	00:13
3	00:15	00:16

Clip 8

Segment number	Start time	End time
1	00:02	00:03
2	00:04	00:04
3	00:09	00:11
4	00:12	00:12
5	00:13	00:13
6	00:16	00:16
7	00:19	00:26
8	00:27	00:32

Clip 9

Segment number	Start time	End time
1	00:04	00:12
2	00:22	00:28
3	00:34	00:40
4	00:51	00:55

Clip 10

Segment number	Start time	End time
1	00:01	00:02
2	00:06	00:07

System recorded Segmentation

Clip 1

Segment number	Start time	End time
1	00:11	00:14
2	00:17	00:17
3	00:20	00:20
4	00:24	00:25
5	00:30	00:33

Clip 2

Segment number	Start time	End time
1	00:00	00:01
2	00:03	00:05
3	00:06	00:07
4	00:08	00:09
5	00:10	00:10

Clip 3

Segment number	Start time	End time
1	00:00	00:01
2	00:10	00:11
3	00:20	00:21
4	00:30	00:31
5	00:40	00:41
6	00:50	00:51
7	01:00	01:01

Clip 4

Segment number	Start time	End time
1	00:03	00:06
2	00:07	00:09

Clip 5

Segment number	Start time	End time
1	00:02	00:04
2	00:08	00:09

Clip 6

Segment number	Start time	End time
1	00:00	00:01
2	00:09	00:10
3	00:21	00:22
4	00:23	00:24

Clip 7

Segment number	Start time	End time
1	00:10	00:11
2	00:11	00:12
3	00:15	00:16

Clip 8

Segment number	Start time	End time
1	00:02	00:03
2	00:12	00:13
3	00:16	00:17
4	00:19	00:20
5	00:21	00:24
6	00:24	00:27
7	00:27	00:29
8	00:29	00:30
9	00:31	00:33

Clip 9

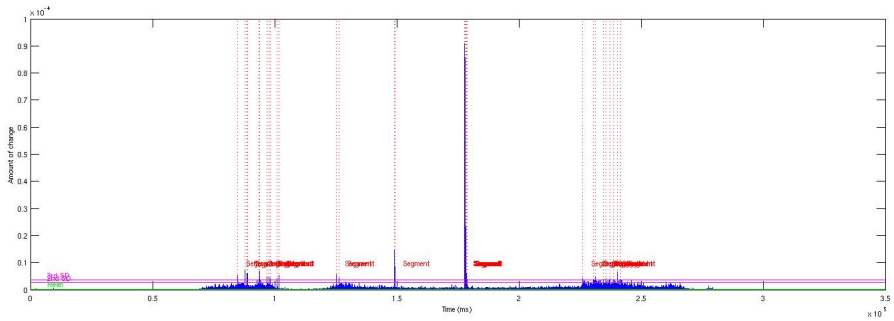
Segment number	Start time	End time
1	00:04	00:06
2	00:07	00:08
3	00:34	00:36
4	00:37	00:40
5	00:52	00:54

Clip 10

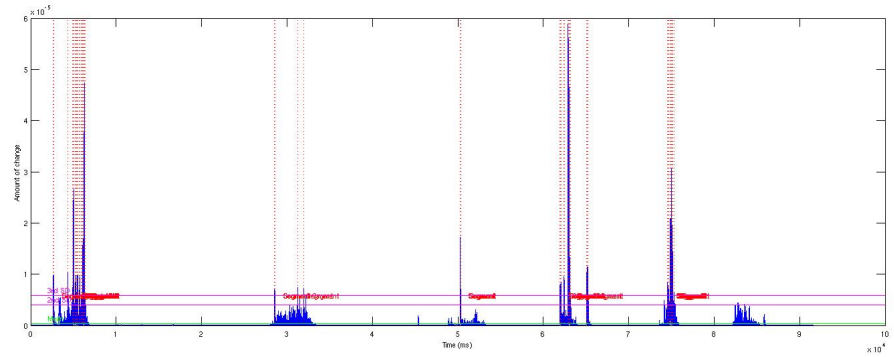
Segment number	Start time	End time
1	00:01	00:02
2	00:06	00:07

APPENDIX F – VALIDATION CHANGE & SEGMENTATION GRAPHS

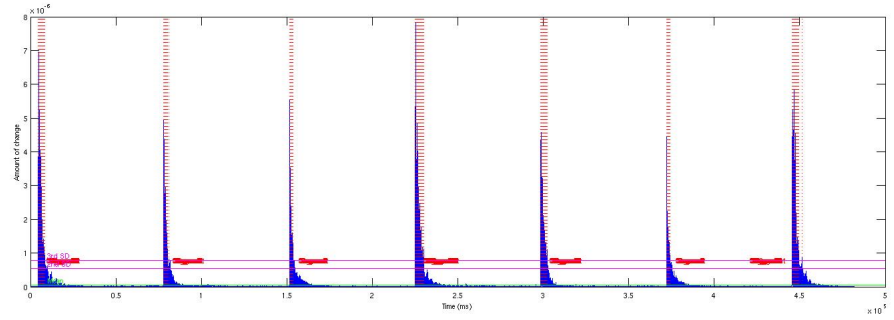
Clip 1



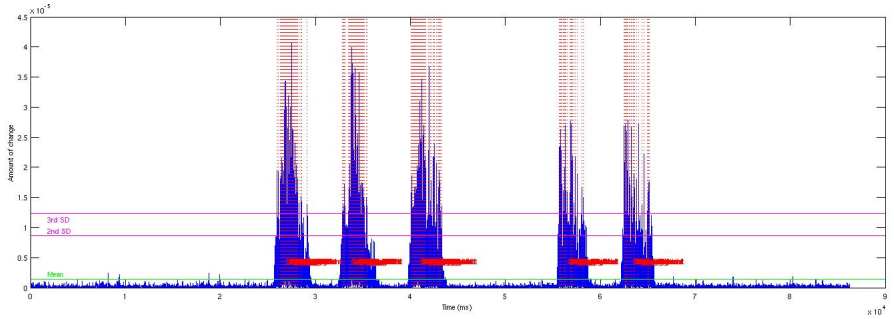
Clip 2



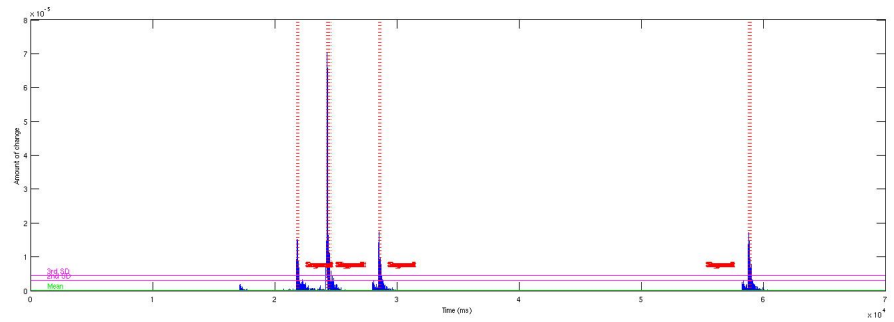
Clip 3



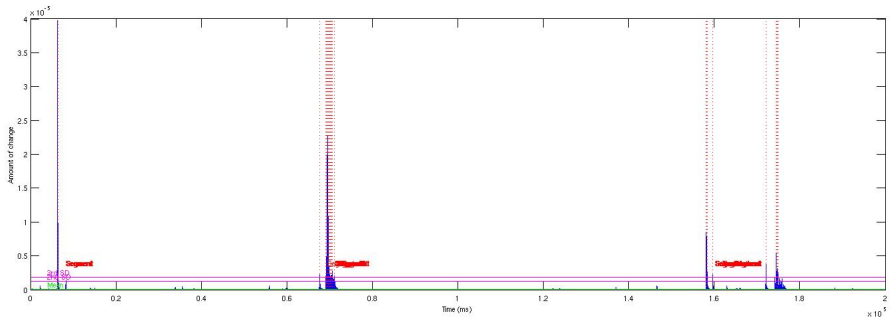
Clip 4



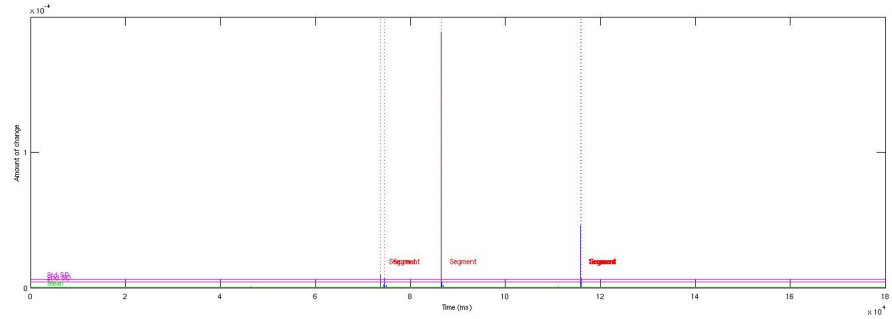
Clip 5



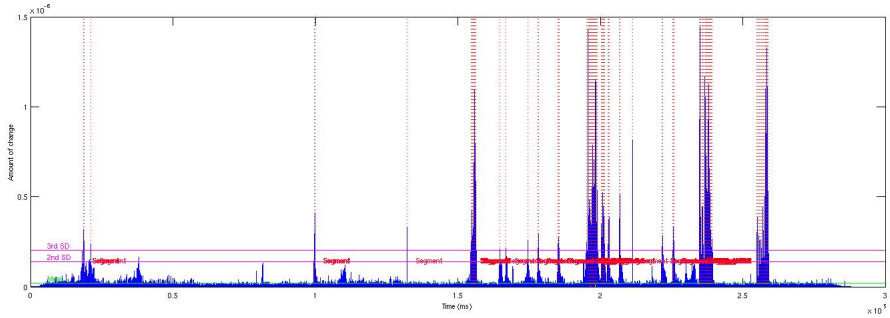
Clip 6



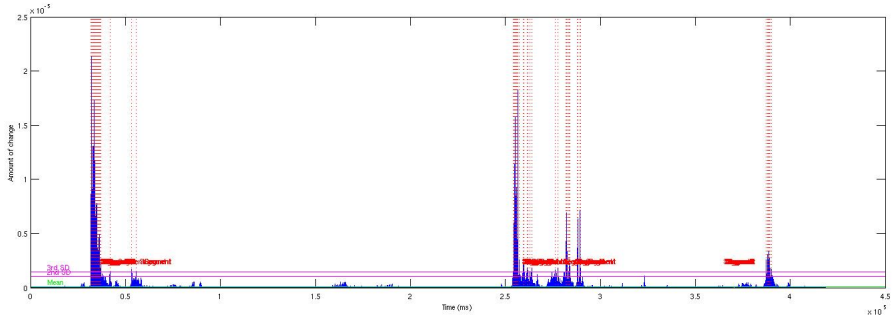
Clip 7



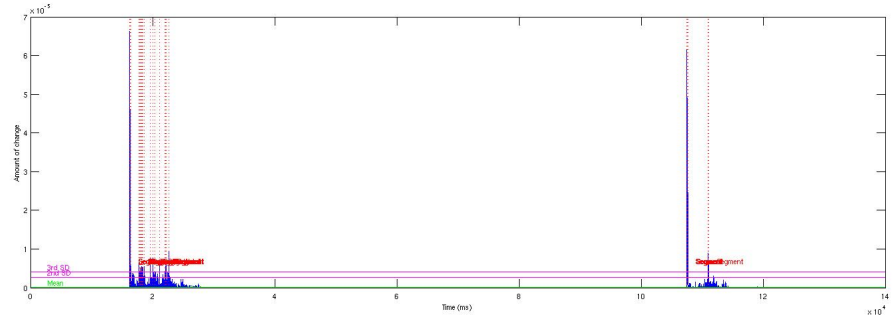
Clip 8



Clip 9

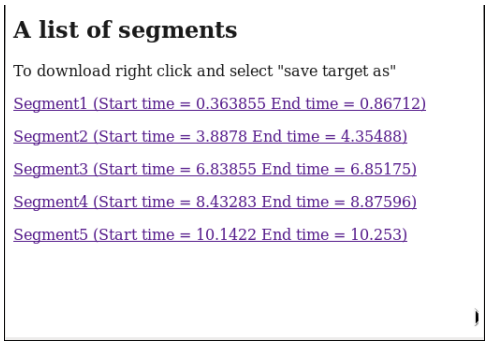


Clip 10

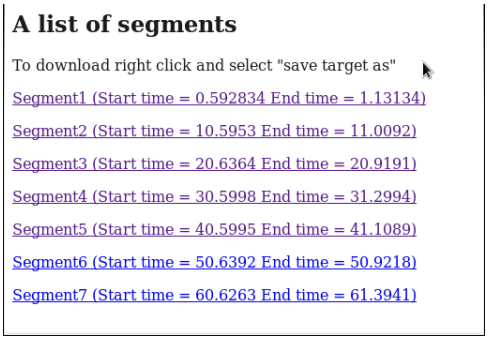


APPENDIX G – FINAL RESULTS + VALIDATION TABLE

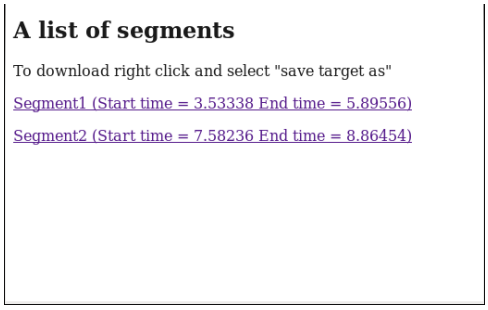
CLIP 1



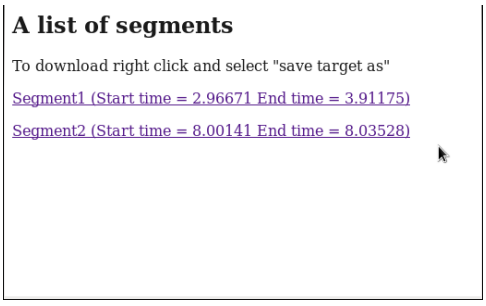
CLIP 2



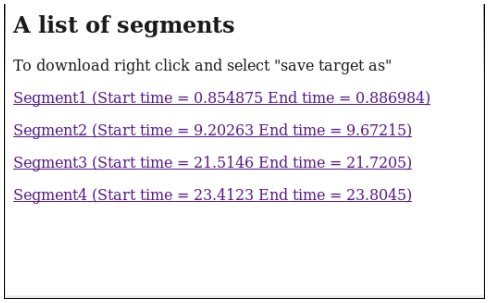
CLIP 3



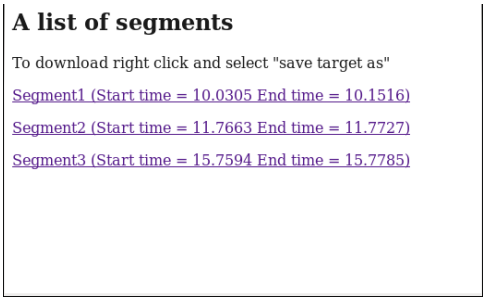
CLIP 4



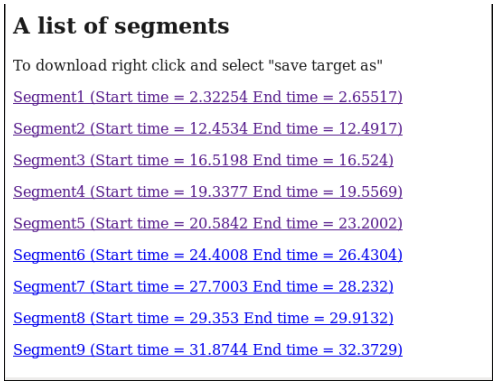
CLIP 5



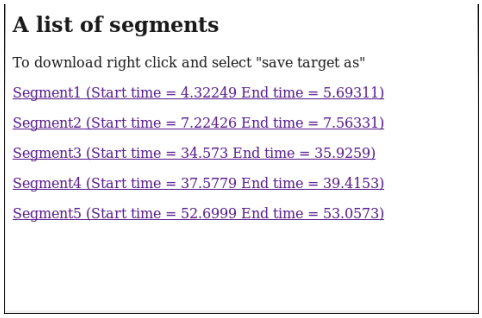
CLIP 6



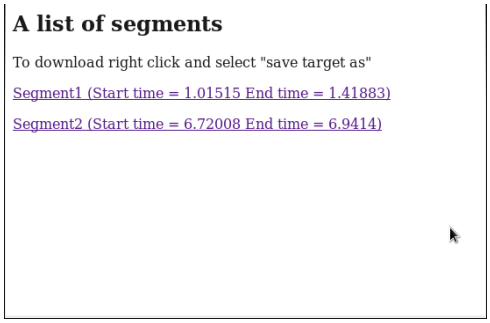
CLIP 7



CLIP 8



CLIP 9



CLIP 10

Validation table

Clip Number	Segment Number Tagged	Segment Number System	Start time missed/added	End time missed/added	Addional Missed time	Notes
1	1	1	1	0		
	2	2,3	1	0	3	Covered by multiple segments
	3	4	0	0		
	4	5	-2	-3		
2	1	1	0	0		
	2	2	0	1		
	3	3	0	0		
	4	4	0	-1		
	5	5	1	-2		
3	1	1	0	-3		
	2	2	0	-3		
	3	3	0	-3		
	4	4	0	-3		
	5	5	0	-3		
	6	6	0	-3		
	7	7	0	-3		
4	1	1	0	0		
	2	2	0	0		
5	1	1	0	0		
	2	2	1	1		
6	1	2	-1	-1		
	2				2	Failed to locate segment
	3				2	Failed to locate segment
	N/A	1	N/A	N/A		Found untaged segment
	N/A	3	N/A	N/A		Found untaged segment
	N/A	4	N/A	N/A		Found untaged segment
7	1	1	0	0		
	2	2	0	-1		
	3	3	0	0		
8	1	1	0	0		
	2				1	Failed to locate segment
	3				2	Failed to locate segment
	4	2	0	-1		
	5	2	0	0		
	6	3	0	1		
	7	4,5,6	0	1		Covered by multiple segments
	8	7,8,9	0	1		Covered by multiple segments
9	1	1,2	1	-4		Covered by multiple segments
	2				6	Failed to locate segment
	3	3,4	0	0		Covered by multiple segments
	4	5	1	-1		
10	1	1	0	0		
	2	2	0	0		