**CodeCrunch – an online tutoring system**

Submitted by

Sng Ping Chiang

In partial fulfilment of the

requirements for the Degree of

Bachelor of Engineering

(Computer Engineering)

National University of Singapore

B.Eng. Dissertation

**CodeCrunch – an online tutoring system**

By

Sng Ping Chiang

National University of Singapore

2013/2014

Project No: H0651000

Project Supervisor: Assoc Prof Tan Sun Teck

Deliverables:

Report: 1

Contents

[Abstract 4](#_Toc416263807)

[Acknowledgement 5](#_Toc416263808)

[Introduction 6](#_Toc416263809)

[Background 6](#_Toc416263810)

[Motivation 6](#_Toc416263811)

[Objective and Scope 7](#_Toc416263812)

[Report Outline 7](#_Toc416263813)

[Literature Review 8](#_Toc416263814)

[Research of features and proof of novelty 8](#_Toc416263815)

[Research on sound analysis 10](#_Toc416263816)

[Definition of digital sound 10](#_Toc416263817)

[Definition of the frequency domain 11](#_Toc416263818)

[Design 12](#_Toc416263819)

[Development Tools 12](#_Toc416263820)

[Architecture 13](#_Toc416263821)

[GUI Design 14](#_Toc416263822)

[Implementation 18](#_Toc416263823)

[Multi-threading 18](#_Toc416263824)

[Loading of multimedia file 18](#_Toc416263825)

[Media Playback Functions 20](#_Toc416263826)

[Video Editing functions 21](#_Toc416263827)

[Trim / Crop video function 21](#_Toc416263828)

[Insert/Concatenation 23](#_Toc416263829)

[Sound analysis / editing functions 25](#_Toc416263830)

[Audio Searching 25](#_Toc416263831)

[Search Algorithm 27](#_Toc416263832)

[Hanning window function 27](#_Toc416263833)

[Comparison of signals 28](#_Toc416263834)

[Cross-Correlation 29](#_Toc416263835)

[Summary 29](#_Toc416263836)

[Audio Muting 30](#_Toc416263837)

[Reflections and Conclusion 31](#_Toc416263838)

# Abstract

This project is a continuation of a previous final year project that was conceived and created by a student of the previous batch. The concept of the previous project was a video creation tool. The tool which was created as a desktop application features screen capture recording, a whiteboard tool, webcam and PowerPoint support. The previous application was incomplete as video editing functions were designed to be included within it but were not finished.

This report describes an independent video editing tool that is created solely for NUS which can be used by lecturers or students for their education materials. The video editor consists of basic video playing and editing functions and a special sound analysis feature that is uniquely catered towards lecturer usage.

Keywords:

E-learning application, Video Editing

Implementation software:

C++, Qt, FFmpeg, Fftw, Libsndfile, Alglib

# Acknowledgement

This work would not have been possible without the help and guidance given by firstly, Prof. Tan Sun Teck, who allowed me a chance to do the project and endured my independent and quiet working character.

Secondly, I would like to extend my heartfelt gratitude to Dr. Steven Halim, who gave me critical and constructive feedback about the project, as well as a chance to proceed with it, despite my many mistakes.

And finally, I would like to thank Mr. David Kurt Grunberg, a PHD student and teaching assistant of CS4347, who gave me useful advice on sound analysis when all seemed lost.

# Introduction

## Background

In NUS, presentation of education content has rapidly advanced across the years. Students are becoming increasingly armed with laptops and tablets though pen and printed notes are still quite likely to be used. Projectors and computers along with PowerPoint slides have replaced the traditional blackboards and dusters. Recently, recorded lessons and interactive online education platforms have also been implemented.

Most of the educational technology services, that educators in NUS require, are provided by the Centre for Instructional Technology (CIT). These technologies include the Integrated Virtual Learning Environment (IVLE), lecture webcasts and various multimedia tools.

The multimedia tools provided by CIT are comprised of audio-visual services which allow lecturers to capture, edit and create educational content within lecture videos that would be uploaded online especially during E-Learning Week. These video production and production services are provided on a cost-recovery basis.

## Motivation

Some video editing tools that are used by educators in NUS include Final Cut Pro and Camtasia which are great video-editing tools created by Apple and TechSmith respectively. However, the users, who are mainly lecturers, may require some features that may not have been implemented. Since the tools are commercial in nature, numerous feedback would have to be given to the tools’ creators to get these features implemented, and there will still be a chance that they would not implement them.

Furthermore, these powerful editors sport complicated user interfaces and functions which are overkills for editing lesson videos which may consist of just slides. They may also prove unwieldy and difficult to use especially for the more senior generation of educators.

## Objective and Scope

This video editor was created with the objective of providing NUS with its own student made video editing tool. It is in the form of a desktop application which would be maintained and upgraded by students who can gather feedback from school educators about specific features and implement them accordingly.

The application consists of basic and easy to use video editing functions. It also comes along with a special feature which helps educators quickly locate and remove repetitive and unnecessary sounds from their educational videos.

This project was a continuation of an earlier final year project which is a video creation application. Though the video editor was designed to be within the application, it was not fully implemented. The media library used by the previous application, Microsoft Expression Encoder Professional, had its distribution and support halted by Microsoft. Though there is a free version of Microsoft Expression Encoder, it also does not support common codecs like H.264 and containers like mp4.

Taking into account the difficulty of obtaining Microsoft Expression Encoder and future problems with maintenance and enhancement, this video editor has been created independently from the previous application.

## Report Outline

This report is divided into five sections: Literature Review, Design, Implementation, Evaluation and Conclusion. The Literature Review section explains the novelty of the application’s special feature and also explains research done on sound analysis. The Design section describes the GUI and program architecture of the video editor. The Implementation section showcases all of the editor’s functions and how they were implemented. The Evaluation and Conclusion section reflects on skills learnt, difficulties faced and possible future implementations.

# Literature Review

## Research of features and proof of novelty

The concept of a video editor is not new. It has been heavily researched on and implemented by many companies which have entered the multimedia software race. While creating a video editor, in and of itself, is a huge learning curve, it can hardly be called novel. Thus a special feature had to be implemented within the application which has to be useful to NUS educators and cannot be found within existing technology.

The special feature that was implemented is basically a sound analysis function, which searches a video of any reoccurrence of an audio segment which is specified by the user. This would allow educators with undesired speaking habits to search an hour long clip for any reoccurrences of sounds, for example ‘Er’s and ‘Erm’s.

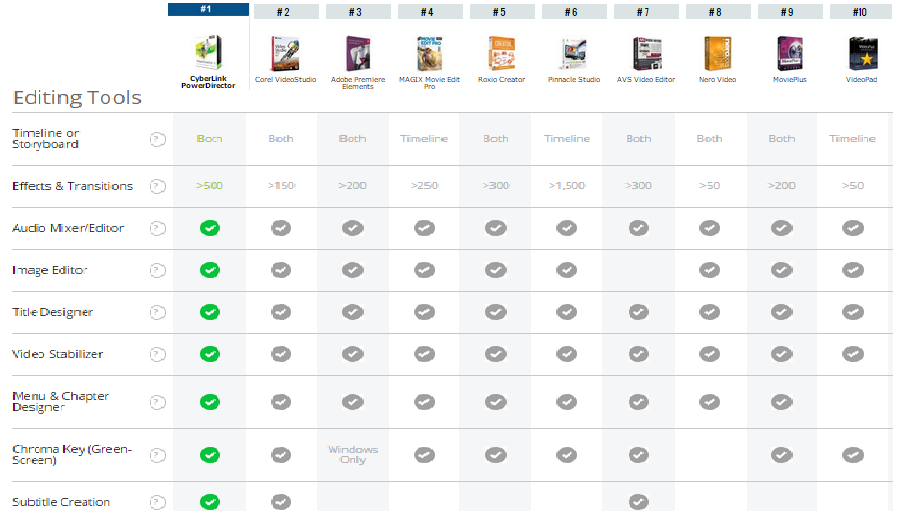


Figure 1: <http://video-editing-software-review.toptenreviews.com/>

Figure 1 above shows editing features present within top video editors of 2015 including:

* Timelines/storyboards
* Effects and transitions
* Audio mixers and editors
* Title designers
* Video stabilizers
* Chroma keys
* Subtitles
* Image editors
* Menu & chapter designers

The features specified that come closest to the concept of sound analysis is probably the audio mixers and editors. Upon closer research, these audio features which are included within modern video editors commonly consist of audio mixing tools which enable a user to add, remove or replace specified audio to their videos. Upon research of top video editors such as Cyberlink Power Director, Adobe Premiere and what the school currently uses which include Camtasia and Final Cut Pro, the special feature cannot be found.

Furthermore, a survey of 69 individuals has been conducted.

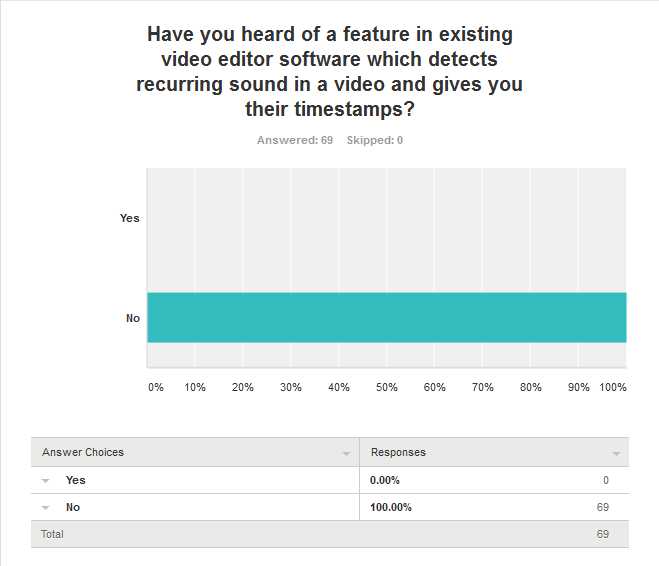


Figure 2 <https://www.surveymonkey.com/results/SM-2NSK6DK9/>

The results have shown that such a feature is unheard of within popular video editing applications. This goes to show that such a concept has most likely not been implemented within any existing multimedia software.

## Research on sound analysis

### Definition of digital sound

Currently, there are two ways to record sound. One way is analogue recording which records waveforms as they are, resulting in a continuous waveform. In digital recording, the sound that is recorded has a series of samples taken from it. The type of sound that will be focused in this report is digital sound.

In digital sound, samples are taken from an audio input at certain sample rates, for example, 4.41kHz which basically means 44100 samples per second. These samples, when combined together, will reproduce the sound, just like how static frames of a video will create a moving picture when played faster than a certain frame-rate.

This is how digital sound in the time domain will look like:

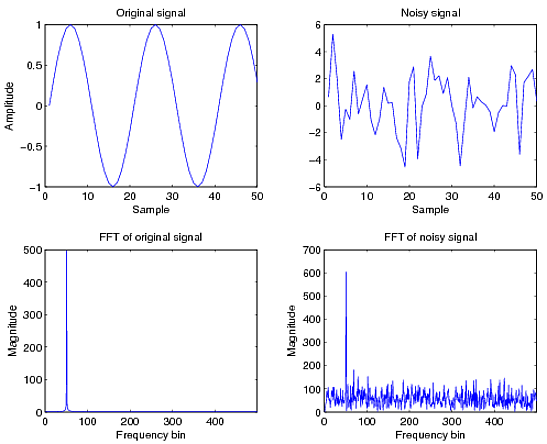
Time

Amplitude

The diagram above shows that digital sound in the time domain is represented by discrete values as opposed to the continuous wave forms represented by analogue sound.

### Definition of the frequency domain

To implement the audio feature, audio data have to be retrieved from the audio files and compared with each other. One can easily obtain the sample values from an audio file. However, digital sound cannot be directly compared within the time domain, especially for vocal sounds or noises. In fact, to compare two similar sounding audio clips, they would have to be compared in the frequency domain.



The diagram above shows two similar sound samples. One is the original sound sample. The other is the original sound sample with noise included. While the sounds look different in the time domain, their frequency spectra look very similar, thus identifying them as the same sound albeit with some noise.

The frequency domain of sound represented by spectrograms can be computed using Fast Fourier Transform (FFT). The FFT takes a signal in the time domain (i.e. the audio waveform, in which time is the x-axis) and transforms it into a signal in the frequency domain. Whereas the input signal consists only of real values, each element of the output series is a complex number, and can therefore be expressed as amplitude and a phase. If we discard the phase information, we are left with just the amplitudes of each of a discrete series of frequency values – which we can then plot as a frequency spectrum. The basic equation of FFT is:

http://www.phon.ucl.ac.uk/courses/spsci/dsp/fftbasis_files/image002.gif

# Design

## Development Tools

As mentioned in the abstract and introduction, the video editing tool that has been developed is a totally independent application from the E-Learning video creator inherited from the previous final year project. Though this project is guided by the research of the previous project, some consideration has to be done for the replacement and upgrading of development tools.

As mentioned earlier, the main media library of previous application has been put out of distribution of the company that had created it. As such, the very first thing that had to be replaced was the media library. There were a few contenders which consisted of Naudio, Gstreamer, FFmpeg and DirectShow were narrowed down to just two; FFmpeg or Gstreamer.

|  |  |
| --- | --- |
| Gstreamer | FFmpeg |
| Powerful with complex operations | Fast and good for simple operations |
| Requires plugins | Standalone |
| Need to write plugins for multimedia operations | Low level API for multimedia operations like encoding, decoding etc. |
| Slow patches and maintenance | Fast patches and well maintained |

One huge advantage of FFmpeg is that it is standalone and does not require writing or installation of plugins. As the basic operations required for video editing would be simple clipping and merging, FFmpeg would do the job fast and well. Looking at the project deadline, FFmpeg would be an easier library to master.

There was not much choice for the main programming language unless wrappers are brought into the picture. FFmpeg was written for C and has recently included support for C++. Because of familiarity and the fact that C++ is object-oriented, C++ was chosen over C.

The previous project used C# with windows presentation foundation for the GUI layer. Due to the change in main programming languages, the GUI layer also has to be changed. Qt was chosen due to its compatibility with C++ as well as its cross-platform compatibility.

## Architecture

The application was designed along the lines of Modal-View-Controller architecture structure. Basically there are 3 main classes:

**MainWindow**

This class represents the GUI layer. It contains buttons, text-boxes and other inputs which the user will use to interface with the application. Besides that, it is also responsible for the basic playing of multimedia content and displaying the results of the users input such as audio search results and timestamps.

**Engine**

This class would be interfaced by the MainWindow class. It will accept input from the MainWindow class and execute the logical functions of the video editor, such as video conversion and manipulation, Fast Fourier Transform and searching algorithms. It will then output the results back to the MainWindow class to be viewed by the user.

**DataStorage**

This class is the storage of the video editor. It stores the data that has been read in by MainWindow and processed by Engine, such as audio sample data, video manipulation timings and important information about the video being edited. . The data would be manipulated by Engine and retrieved and presented by the MainWindow.

.

MainWindow Object

Engine Object

DataStorage Object

Gives Instructions

Retrieves/manipulates data

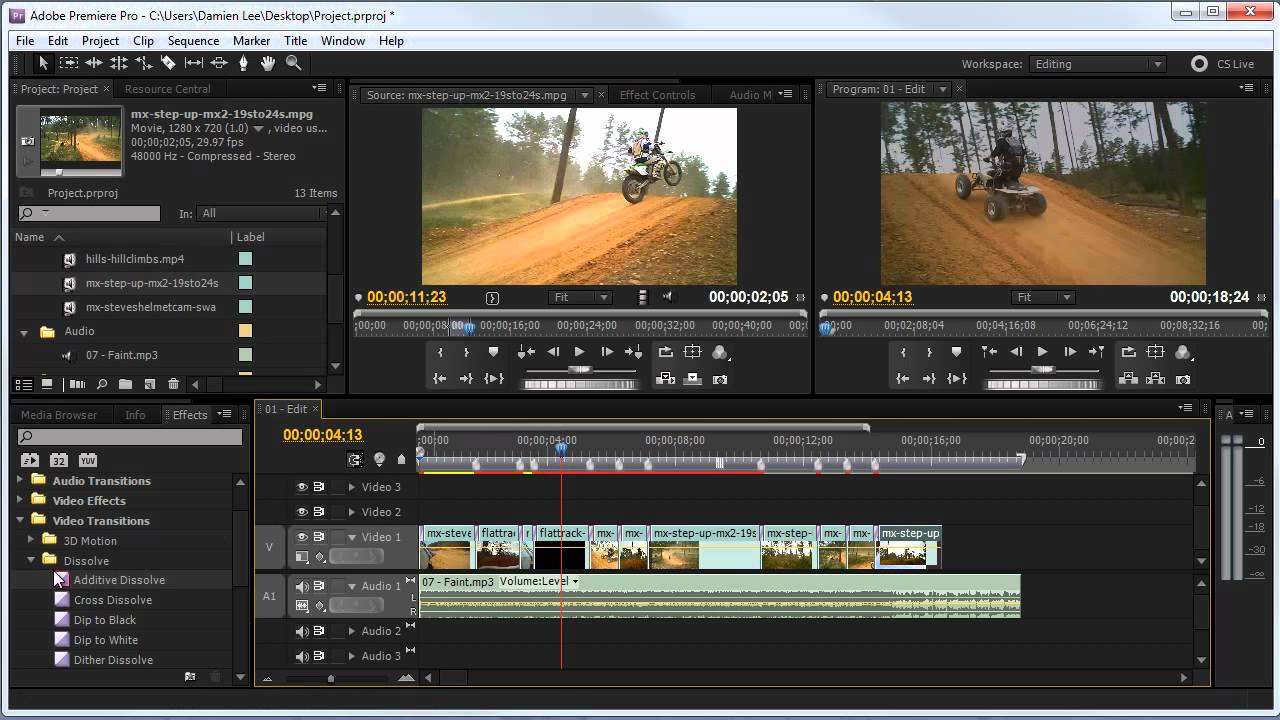
Interfaces

Results

Results

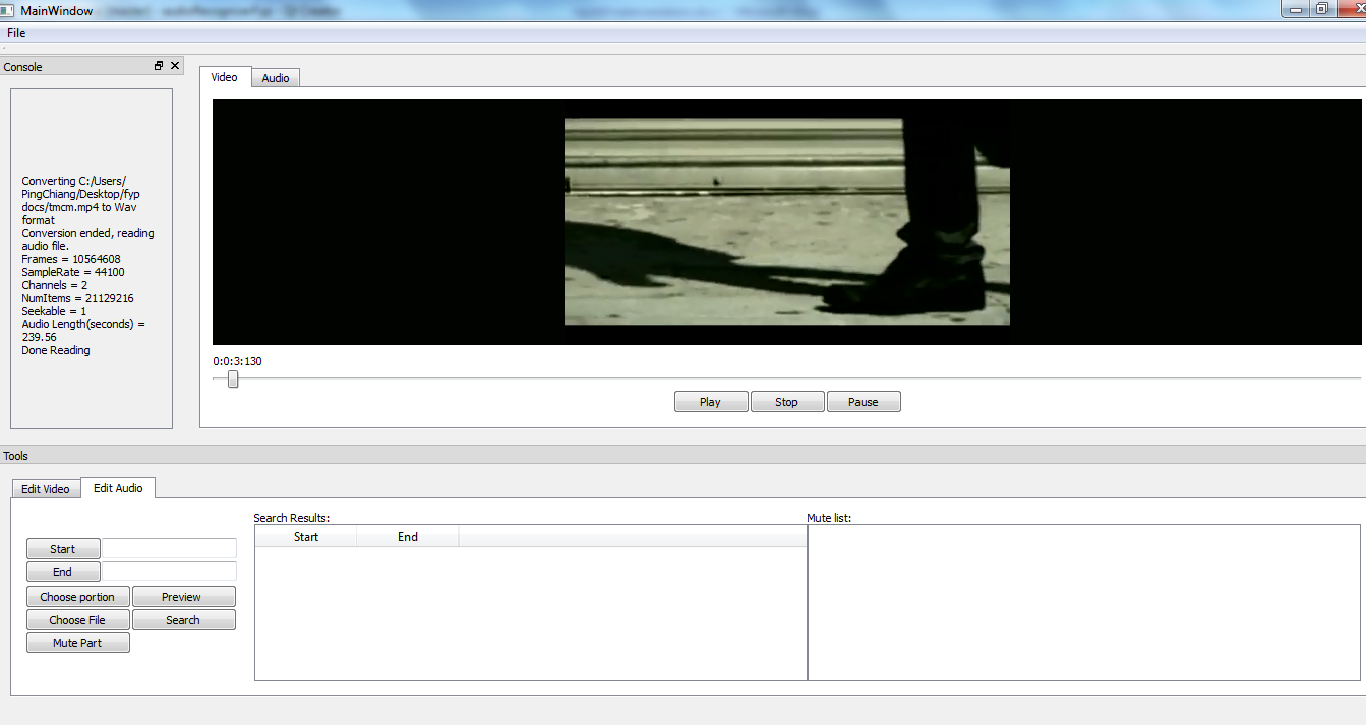
The diagram shows how the classes interact. A dataStorage object is contained within a mainWindow object. When the user presses a button, the mainWindow would create a new engine object to perform the logic that is required. The engine object would obtain data from the dataStorage object and send results back to the main window object which would then display the results to the user.

## GUI Design



The previous diagram shows how a typical commercial video editor (Adobe Premiere Pro) looks like. While considering the design of the GUI, the very first factor of importance is the usability of the interface. Modern video editors have very cluttered GUI which looks intimidating at first glance. They sport storyboards with multiple layers of objects to edit. They have many editing functions which are hidden within dropdown menus which make them hard to find or represented by icons which are difficult to interpret. Furthermore, while these functions, like the storyboard, which may make the GUI seem colourful and sleek, they are not tailored exactly to the needs of educators.

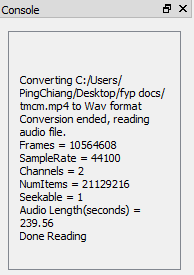
This project aims to give educators in NUS an easy-to-use video editing tool which is aimed towards the editing of educational content. While designing the graphic user interface, usability is taken into priority. The interface must not look intimidating and cluttered. That would prove to be counter-productive and users would have to spend a certain amount of time finding functions and mastering the tool. The interface must be easily intuitive to the user whom should expect to be able to tell what a button does just from one look. Also, the layout of the application should be rearrange-able by the user to their comfort.



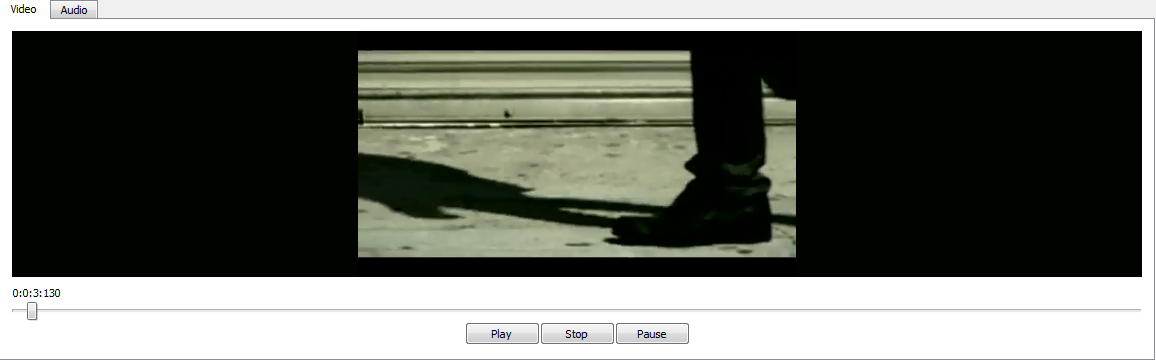
The previous diagram shows the interface of the video editor that has been created for the educators of NUS. The GUI is kept clean and uncluttered as compared to a commercial video editor. As shown, the number of buttons was kept at a minimum and they are accessible in easily found locations. Note that though it may add slightly to the clutter of the screen, the buttons are well labelled such that the user would be able to tell what they do in one glance.

The video editor has a simple and consistent layout. It also makes use of dock widgets which are provided by QT. Dock widgets allow different components of the GUI to be dragged around. Dock widgets can be rearranged, detached, closed or separated from the application in the form of separate windows. It is divided into 4 different parts: the menu-bar, the console, the multimedia presentation, and the tools portion.

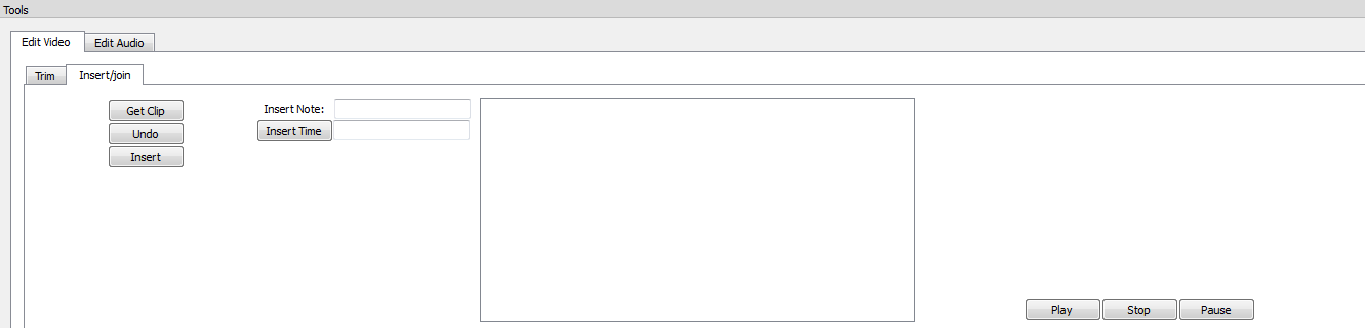
The menu-bar is a simple bar right at the top of the application. As dropdown menus would force users into finding functions that they want, the menu-bar only has one “File” button which is commonly used in applications to load and save files.



The console (above) is contained within a dock widget. It outputs important data normally released by the engine and GUI regarding the processes that are started by the user, such as completion percentage, errors, etc.



The multimedia presentation section (above) in the middle of the GUI is responsible for playback and representation of media. As it makes up the core of the editor, it is not encased in a dock widget. It has two tabs, both catering to the representation of video or audio playback.



The tools section, like the console section is contained in a dock widget and comprises of video and audio editing functions. Intuitively labelled tabs have been implemented in the tools and multimedia presentation to facilitate ease of navigation between different categories.

# Implementation

## Multi-threading

While implementing the video editor, it was noticed that any function or process such as audio extraction, audio conversion and video cropping would cause the GUI to freeze up. The reason for this is that even though these processes are normally from a different class, they operate on the same thread as the GUI. This would cause the GUI to wait and freeze while the processes are running and the users would be unable to interface with the application until the process is over. Thus there was a need for this application to be multi-threaded.

Multi-threading is implemented through the QThread class provided by Qt. The Engine class was sub-classed as a QObject. This allows its functions to be shifted by Qt into new threads using the moveToThread function provided by the QThread class. Threads communicate with other objects through signals and slots which are core mechanisms provided by Qt.

Signals are emitted by objects which wish to let another object know when a particular event occurs. Slots are functions which are connected to signals which would run when specific signals are emitted. Signals and slots are loosely connected: an object emitting a signal will have no knowledge of the slots that receive the signal. Signals and slots can take any number of arguments of any type and are completely type safe.

Many parts of this video editor, from basic buttons to complicated functions will make use of signals and slots for fast and effective communication between objects.

## Loading of multimedia file

This video editor supports editing of both video and audio recordings in common file formats such as mp4, mp3, wmv and wav. The loading and initialisation of the main multimedia file is done in the MainWindow class. The user basically just needs to choose the path of the multimedia file using the “open” button in the top menu bar. A simple diagram summarising the loading process is shown below.

Spawn separate Engine threads

Initialise video with QVideoPlater

Extract audio data using Libsndfile

Load File Object

Save sample data to dataStorage

Emit “Finished” signal to MainWindow

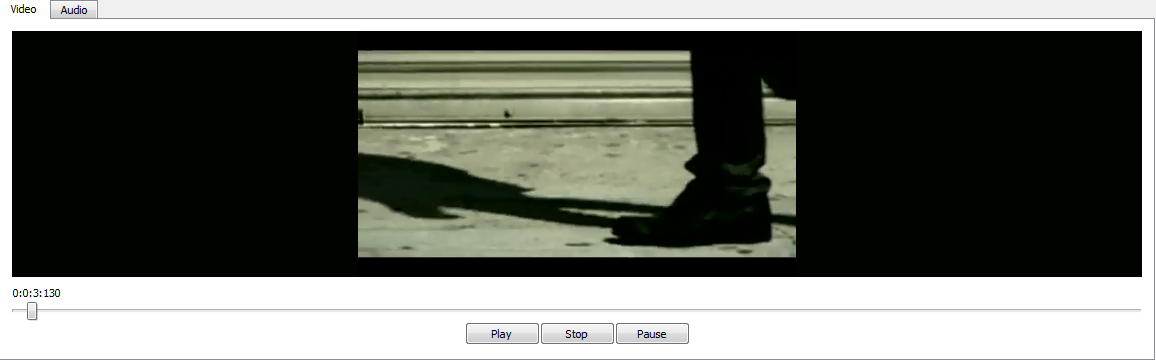
Multimedia support has been provided by Qt in the form of the Qt Multimedia module. It allows the GUI layer to perform playback of multimedia content. When the path of the multimedia file has been defined by the user, the MainWindow will initialise the video widgets and allow playback of the media.

Sound processing would definitely be needed for the audio search special feature. The sound processing is implemented by extracting out the audio stream of the multimedia file and converting it into wav format. The newly created wav file will have its sample data processed and stored into a dataStorage object.

Both video initialisation and sound processing are implemented in separate threads so that they can be done concurrently while also not freezing the GUI. Once these threads have finished their respective jobs, they would fire off a “Finished” signal which is connected to a slot in the MainWindow. The console would then let the user know when the media has been loaded and read into the application successfully.

Although the sound file can be read manually using C++ by opening the file, skipping header and other unnecessary values and reading just the sample data chunk into an array, it was decided that it would be best not to reinvent the wheel and experience unnecessary and potential problem causing bugs. Thus a library (Libsnfile) was chosen to extract sample data from the created wav file. Libsnfile is able to read a wave file and return an array of doubles which represent the sample data values.

## Media Playback Functions



The video editor supports the playback of both audio and video files. Media playback is done within the GUI layer, which is the MainWindow class. The picture above shows the playback section. It consists of a media container, a seek-bar and a play, stop and pause button.

The media playback was implemented using with the help of the QMultiMedia module provided by Qt. A normal Label is placed in the gui and promoted to a QVideoWidget. The QVideoWidget is automatically placed in a separate thread by Qt. The QVideoWidget can then be accessed by a QVideoPlayer object which is able to make it load, play, pause and stop media. The QVideoPlayer class also contains low level API which reveals important details of the media being played, such as its duration.

The seek-bar is implemented using a slider widget. It tells the user the timestamp and approximate position of the media that is being played. It also gives the user precise control of the position of the media playback. The seek-bar is also tied to other functions of the video editor which rely it to retrieve timing. To allow maximised performance and accuracy, the seek-bar has many slots connected and waiting for the emission of signals by the QVideoPlayer object and vice versa. Time is implemented down to the accuracy of milliseconds which explains the triple digits behind timestamps.

When a media is loaded, the seek-bar would be updated with the duration of the video. When a media is played, QVideoPlayer class would emit a signal containing an integer variable of the current position being played. This signal is connected to slot in the seek-bar which would move the seeker. Vice versa, when the seeker is moved, a signal is fired off to the QVideoPlayer object which lets it know what position the seeker is at. The QVideoPlayer object would then change the position of the playback accordingly.

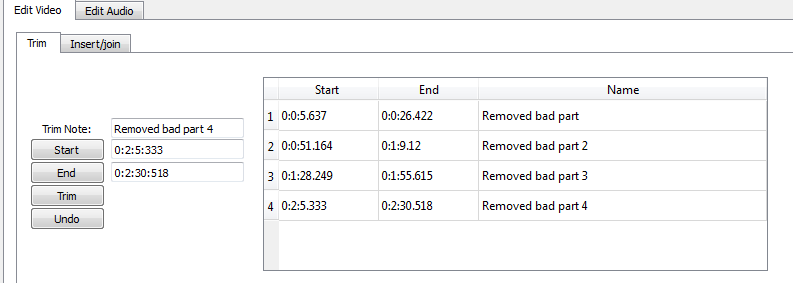
## Video Editing functions

During video recording, there would certainly be mistakes like unwanted parts or parts that are left out. The video editor must be able to manipulate and change video content. Namely the removal of unwanted segments and inclusion of needed segments so that the user does not have to re-record an entire video just because of a small mistake. As the audience are mainly lecturers, special features like transitions and video effects are not necessary. All editing functions in this video editor are powered by the FFmpeg media library.

### Trim / Crop video function

The trim function is essential in video editing. It allows the user to remove unwanted parts of the video. While designing the trim function, some important considerations have to be taken into account. How would the user determine which timestamp to cut the video and what is the easiest way to do it? How does the user preview the changes? Should a new video be encoded every time a trim is made?

Every trim made must have a start and end time defined. The start and end time can be manually typed in for higher accuracy. However, the video editor provides an easy way to input the timestamps. The user just has to adjust the position of the seek-bar and press the start or end buttons to get the position they want to trim.



The video editor also gives the user a choice to enter a note related to the trim, just in case they want to refer back to it and maybe undo it. After all the fields are filled in, the user just needs to press trim to execute the trim function. When trimming, the start and end values would be stored directly into the dataStorage object. All trims would be displayed in a table beside the buttons as shown in the previous diagram.

A simulation approach was taken towards implementing the trim function. Basically the video editor, with the help of pointers and objects should simulate the trim without really encoding a new video every time a cut is made.

Seek-bar

Cut Part



Skip to this position

If playback reaches here

If the video position is equals to or in between any of the start and end timing of values stored within the dataStorage, the video player would basically change position to the end timing, effectively skipping the cut part of the video. This would allow the user to preview quickly the results of the trim without having a new video encoded.

The video editor also supports an undo function. With the implementation of the simulated trim, the undo function basically allows the user to select any time segment from the table and remove it.

The simulated trim allows a quick preview of the end results of trimming a video. However, the video has yet to be really trimmed. To do that, an export function that would create a resultant video would be required. There are three trimming scenarios:

* Trim from start time to x time
* Trim from x time to y time
* Trim from y time to end time

The first and third scenarios are easy as they basically result in one resultant video each. The second scenario however is more complicated as there exists no function in FFmpeg which would cut and concatenate the videos automatically.

Given a list of trimmed segments, this is how the video editor implements the trimming:

D

C

B

A



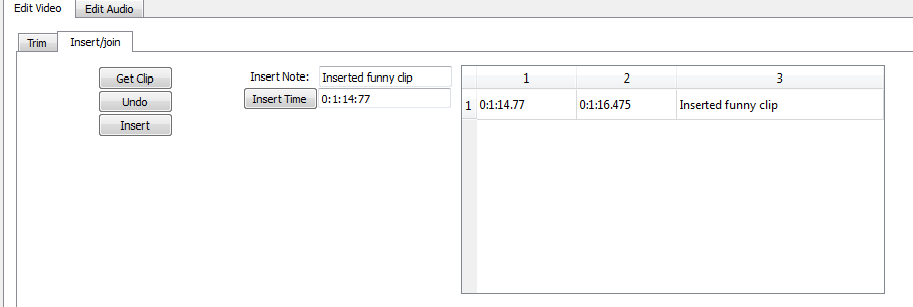
Assuming the squares in the timeline are the cut parts, the video editor would slice parts of the video from:

1. Start-Time to A
2. B to C
3. D to End-Time

These three parts would be encoded by FFmpeg as separate videos and concatenated together to form the resultant video.

### Insert/Concatenation

The insert or concatenation function basically allows the user to insert external clips into the video using the editor. Looking at the various considerations taken for the trim function, it can be seen that this function is more troublesome to implement.



The user can choose which clip to insert by pressing the get clip button and also type in a note related to the inserted clip. The position to insert the video can be input from the seek-bar or by manually typing in to the text box. Upon insert, the insert timing would be reflected in the table. Clicking any cell in the table and pressing undo would result in the inserted clip being removed from the table and video.

There are three scenarios where an insertion would happen:

* Insert at start time
* Insert at x time
* Insert at end time

While trimming can simulated easily by skipping parts that were trimmed, it would prove impossible to simulate the insertion of another clip within the duration of the edited video. Thus the video would have to be immediately encoded for the user to preview.

B



Inserting at the start time and end time consists of simple concatenation. However looking at the diagram above, a clip inserted within the duration of the video would be slightly problematic. One way to deal with it is to:

1. Cut from start time to B (clip A)
2. Cut from B to end time (clip B)
3. Concatenate clip A with inserted clip (clip C)
4. Concatenate clip C with clip B (resultant video)

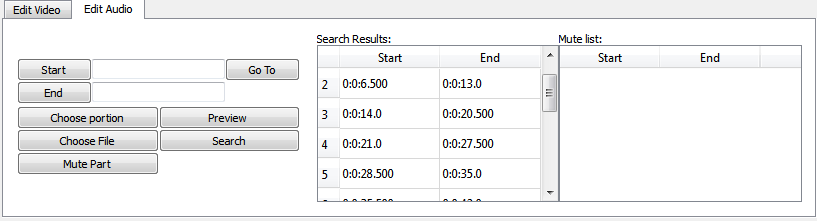
This presents a few problems. Now the entire editor has to work around the resultant video. That means that the video has to be reloaded into the editor and audio stream re-extracted and stored. Also, there would be a change in timing of the actual video. The shift in timing after part means simulated edits would have in-accurate timings. Thus all the simulated edits after point B would have to have their timings shifted forward by the duration of the inserted clip.

For the undo function, the opposite happens. A video would be created by trimming from the start of the video to the start of the inserted clip. Another video would be created by trimming from the end of the inserted clip to the end of the video. Both videos would be concatenated and encoded as a new resultant video. Similarly, all simulated edits that have timestamps after point B would have to have their values decreased by the duration of point B.

## Sound analysis / editing functions

### Audio Searching

This is the special feature of the video editor. Basically, this feature requires the user to either select a portion of the video clip or choose an external audio clip. The video editor would then search the entire audio portion for any occurrences of the selected audio portion/clip. This would prove useful for educators with undesired recurring speaking habits.



The user can specify a start and end, either by pressing the start/end buttons after moving the seeker in the seek-bar or manually typing in the timing in the textboxes, before pressing the choose portion button. The choose file button would open a dialogue which would allow the user to specify an external audio or video file. The user can then search for reoccurring sounds using the search button.

The main flow of the search algorithm goes like this:

a

Choose audio to search for (secondary clip)

Create a separate Engine Thread

Extract audio data from secondary clip Hanning Windowand FFT it

Retrieve audio data from main clip

Extract audio data from secondary clip

End of main clip

Slice portion from main clip = size of Audio Clip

Hanning window and FFT slice and compare

Update Search Results

No

Yes

### Search Algorithm

To enable more efficient searching and Fast Fourier Transform, the larger main clip would be broken up into slices the size of the selected smaller secondary clip. These slices may or may not overlap each other, depending on the time interval in between the slices.

Imagine a 1 second user defined sample and a 10 second main video with search interval of 0.5 seconds:



Compare

Looking at the diagram above, the 10 second main video would be sliced into overlapping smaller 1 second segments, in 0.5 second intervals. These smaller segments would be compared with the chosen secondary segment.

### Hanning window function

After slicing, the Hanning Window Function is applied to the both segments of the secondary and main audio to smoothen out any distortion near the edges of the segment. These distortions which may occur due to slicing may result in undesired results to the FFTed signal. The formula for Hanning Window is:

w(n) = \alpha - \beta\; \cos\left( \frac{2 \pi n}{N - 1} \right)\,



The previous diagram demonstrates the result of Hanning Window (green) on the original distorted waveform (blue).

**Fast Fourier Transform**

To avoid wasting development time & effort, the FFT of audio signals was implemented using a FFT library. This is a list of some libraries considered for the FFT:

* FFTW: “Fastest Fourier Transform in the West", available under GPL, commercial licenses available from MIT.
* Intel MKL: a commercial FFT library optimized for Intel chips.
* KissFFT: “Keep It Simple, Stupid", not as fast as FFTW, but BSD-licensed. http://kissfft.sourceforge.net/

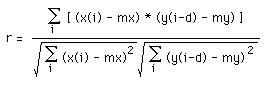
After much deliberation between KissFFT and FFTW, FFTW was chosen particularly because of its speed and documentation support.

### Comparison of signals

The signals which are in comparison would not be exactly 100% the same. Thus, a fuzzy comparison would have to be implemented. Upon research, Cross Correlation has been found to be the best algorithm for the job.

### Cross-Correlation

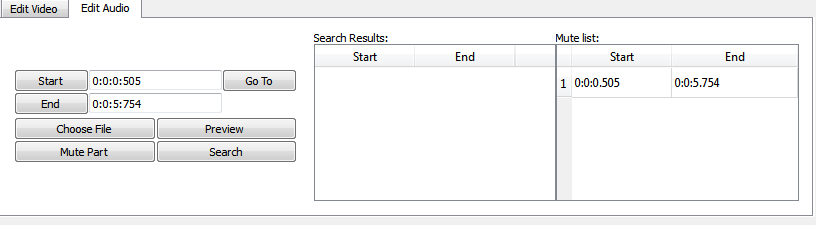
Cross correlation is one of the methods to estimate how similar two series are. Consider two series x(i) and y(i) where i=0,1,2...N-1. The cross correlation r at delay d is defined as



|  |  |
| --- | --- |
| As a simple example shows two rectangular pulses below in blue and green, the correlation series is shown in red.  http://paulbourke.net/miscellaneous/correlate/xcorrelate3.gif  The maximum correlation is achieved at a delay of 3. Considering the equations above, what is happening is the second series is being slid past the first, at each shift the sum of the product of the newly lined up terms in the series is computed. This sum will be large when the shift (delay) is such that similar structure lines up. This is essentially the same as the so called convolution except for the normalisation terms in the denominator. Summary The audio searching feature would return a list of cross-correlation values. Those which are closer to 1 are similar and those who are closer to -1 are less similar. This would allow the video editor to pick out the timestamps of the reoccurrences. |  |

## Audio Muting

Now that reoccurring sounds have been detected, there should be methods for the lecturers to eliminate these reoccurrences. One of the ways is to mute the sound. And thus, audio muting is included within the video editor.



The user can choose to:

* Click on particular timestamp within the search table and mute a particular reoccurrence
* Click on particular timestamp within the mute table and choose to undo the mute
* Preview specific timestamps

A point to take note: Muting is simulated by the video editor and is not really implemented until the user clicks the export button. This would allow faster previews as opposed to the trim function. During export, the video would have its parts muted before implementing the trims so as to prevent any interference in values.

Muting is implemented by first cutting the video into three segments.

* Segment A: Before mute
* Segment B: During mute
* Segment C: After mute

Segment B would then be muted before being concatenated with segments A and C.

# Reflections and Conclusion

Before the CA evaluation, research and implementation was done on the inherited FYP project. However, many flaws regarding the inherited FYP project were pointed out. After the CA evaluation, it was decided that it would be better to move into a different direction and to create an independent application from the previous FYP.

Because much development time was consumed into researching and creating a brand new application in the second half of the FYP, the video editing tool is still extremely unpolished. The functions can still be tweaked to be more user-friendly and the GUI can still be improved upon. Loading speed also has to be optimised further.

However, this project is well documented with comments and is created with up-to-date and non-proprietary libraries and can be easily maintained by future batches of students.

Some future implementations to consider:

1. Storyboard
2. Preview panel
3. Save project file
4. Share on social media