2019

An investigation into the use of video steganography in the distribution of subtitles

BSC (HONS) SOFTWARE ENGINEERING

PETER MCNEIL - 15848156

Table of Contents

Glossary of Terminology

- JPEG – A compression standard for images, with the file extension of .jpg/.jpeg.
- Homebrew – A package manager for MacOS, which downloads and configures software.
- Codec – Software that compresses data to enable lower storage costs and faster transmission and decompresses when the data is shown.
- H.264 – A popular video codec
- Colour space – A way of representing colours on a digital screen
- RGB - A colour space with Red Green Blue values
- YCbCr - A colour space used in digital images representing the luma component (Y), and the difference in the red and blue chroma components (Cb and Cr)
- Lossy – A codec is described as lossy if the original data cannot be retrieved when decoding.
- Lossless – A codec is described as lossless if the original data can be retrieved when decoding.
- Steganography – The practice of hiding information in plain sight, i.e. every capital in a paragraph spelling out a word.

1 - Introduction

In this project I will investigate whether it is feasible to combine video and subtitle files into a common file by using video steganography. Steganography is the art of hiding data in plain sight and has been in use for thousands of years to convey messages secretly, i.e. writing a message in invisible ink. Video steganography modifies the video itself to hide data inside the file, without effecting what a video consumer will see. This mechanism can be used to combine both files together. Video steganography is a complex process, which can affect the compression algorithms of the codec.

This idea occurred to me during my placement year as a Software Engineer at the BBC. I subsequently introduced the idea to my project manager, and we discussed how distributing subtitles with digital video was a problem that the BBC was struggling with. The subtitles used by the BBC are separate files to the video and are typically stored at a different location. This leads to the subtitles being difficult to locate and use alongside video. Despite having this conversation with the BBC, this project is not sponsored by them.

This project will deliver a toolkit, Lodge, with three components; a C++ library, a command line (CLI) tool and a graphical user interface (GUI) application. The GUI is intended for use by a video producer in a postproduction step to embed subtitles into a video, but can be used to extract the subtitle files if wished. The library and CLI are intended to be used by video player implementers to decode and display the subtitle files in a Lodge video.

All code for this project is hosted at https://www.github.com/petermcneil/lodge

2 – Background Research

2.1 - Subtitling

Subtitles, or captioning, is text displayed on-top of video that represents sounds made in visual media, such as videos or games. This sound can be speech, music, off-screen noise, ambient noise, foreign languages translated to the desired language, speech with an accent, and additional information to help the hard of hearing. There are two types of subtitling; open and closed (BBC, 2018). Subtitles are used by at least 10% of viewers and up to as many as 35% depending on the content being watched (Armstrong, 2016).

For the purposes of this project there will be a focus only on digital video making and therefore broadcast subtitling will be ignored. The reason being that there is no use for this project in broadcast or DVD manufacturing due to the fact the video file isn't wholly distributed to the end user.

2.1.1 - Open Subtitles

Open subtitles are subtitles that are burnt into the video itself. Editors can use subtitles to display extra information to the user (see fig. 1). More typically they are used to show translations of a foreign language or when the actor has a difficult to understand accent. They are able to match the films tone and improve the experience of viewing a film. Other times a video may be distributed with a full subtitle script burned into the screen, this method is becoming increasingly popular with online video on social media; allowing the end-user to watch the video and have full comprehension of the subject without turning on sound. A major disadvantage, however, is that the open subtitles aren't easily editable, requiring a full re-render of the video. Another disadvantage would be that the end user has no control over whether the subtitles are on screen or not.

Figure 1 - BBC, Sherlock, 2014

2.1.2 - Closed Subtitles

Closed subtitles on the other hand are separate files to the actual video. This has the advantage of being editable independently from the video, user toggle-able, and easily changed by the end user. This type of subtitling is the type most familiar to a layperson. Common use cases for closed subtitles are for use by the hard of hearing, to supplement audio in noisy environments, and for foreign language translations of the script.

Contrary to open subtitling, there is more regulation and standardisation surrounding closed subtitling. There is a wide variety of subtitling with different focuses, ranging from broadcast standard formats with tooling to support it, to simple formats that are easy to write with a text editor. For example, the BBC use two different encodings, EBU-STL in conjunction with EBU-TT for broadcast and EBU-TD for online video (BBC, 2018). Whereas YouTube supports a huge number of subtitle formats, SubRip being the most basic but preferring EIA-608 to be used on the site (Youtube, 2018).

An example of a basic SubRip subtitle format:

SubRip (.srt) example

```
1
00:00:00,599 --> 00:00:04,160
>> ALICE: Hi, my name is Alice Miller and this is John Brown

2
00:00:04,160 --> 00:00:06,770
>> JOHN: and we're the owners of Miller Bakery.

3
00:00:06,770 --> 00:00:10,880
>> ALICE: Today we'll be teaching you how to make
our famous chocolate chip cookies!

4
00:00:10,880 --> 00:00:16,700
[intro music]

5
00:00:16,700 --> 00:00:21,480
OK, so we have all the ingredients laid out here
```

Figure 2- SubRip file example (Youtube, 2018) - https://support.google.com/youtube/answer/2734698

As the above example shows, SubRip is an easy to read format that displays the subtitling information in a simple way.

On the other hand, EBU-TT-D is more complex, including metadata about the video and subtitling information (see fig 3)

```
<tt:body ttm:role="caption">
        <tt:div>
                <!-- Subtitle zero - not for display -->
                <tt:p xml:id="C0" region="R1" begin="00:00:00:00" end="00:00:00:02" style="S2">
                        <tt:span>Snow White</tt:span>
                        <tt:br />
                        <tt:span>ABC D123A/02</tt:span>
                        <tt:br />
                        <tt:span>XYZ12345</tt:span>
                </tt:p>
                <!-- Begin subtitles for display -->
                <tt:p xml:id="C1" region="R2" begin="10:00:32:05" end="10:00:36:08">
                        <tt:span style="S2">This programme contains some violent<tt:br/>
                        scenes and some strong language
                        </tt:span>
                </tt:p>
                <tt:p xml:id="C2" region="R2" begin="10:02:04:00" end="10:02:06:10">
                        <tt:span ttm:agent="sp2" style="S1">Snow White, wake up!<tt:br/></tt:span>
                        <tt:span ttm:agent="sp1" style="S2">But I'm so tired!</tt:span>
                </tt:p>
                <!--
                ...
                Additional subtitles omitted
                ...
                -->
                <tt:p xml:id="C809" region="R1" begin="01:03:29:20" end="01:03:29:24">
                        <tt:span ttm:agent="sp3" style="S1">..and they all lived happily ever after.</tt:span>
                </tt:p>
        </tt:div>
</tt:body>
```

Figure 3- EBU-TT-D example file (BBC, 2018) - https://bbc.github.io/subtitle-guidelines/sample-ebutt-prepared.html

Despite the differences in the subtitles files, they are all fundamentally a plain text file with a defined format. Plain text can easily be converted into a character stream and then onto a bit representation of that character.

2.2 - Video

A video file is made up of two main components; a container and data streams.

2.2.1 - Container

Video files (e.g. .avi or .mp4) are containers for video streams, which typically also hold audio streams, and can have subtitle streams. These streams are stored independently. For the purposes of this project the video stream will be considered exclusively.

Some container formats - such as .mp4 and .mkv – contain subtitle streams for a number of subtitle files. However, this is not true for every container format and the subtitle type allowed inside the containers can be limited – i.e. Timed Text is only allowed inside mp4 containers. An aim for this project would be to allow all containers and all subtitle formats to be used.

2.2.2 - Video Streams

A video stream – in the simplest form – is a series of images one after another displayed at a constant frame rate. Each image – or frame – is represented by a particular colour space; e.g. RBG or YCbCr. This can be said to be raw video. However, to ensure that file sizes are not ridiculously large, video streams are typically compressed to a smaller size. These compression techniques (codecs)

are usually lossy removing redundant data that doesn't affect the visual quality of the image, but dramatically reduce the size needed to store the video. To keep the scope of this project at a reasonable size the H.264 codec will be focused on.

2.2.3 - H.264

H.264 is the 4th codec in the H.26x line of video coding standards, developed by the Video Coding Experts Group of the ITU Telecommunication Standardisation Sector (ITU, 2019). It is a huge standard comprising of a massive range of profiles and configurations, supporting video resolutions up to 8192x4320. First released in May 2003, H.264 is currently the most widely used codec, with 82% of all video produced and consumed using this codec (encoding.com, 2019)

Many methods are used by H.264 to reduce the size of video files. These techniques are similar to techniques used by other codecs, including the previous versions of H.26x. H.264 is a block-oriented, motion-compensation based video codec and uses two main techniques to compress video; spatial compression, and temporal compression.

2.2.3.1 - Temporal Compression

Temporal compression, or inter-frame prediction, considers a series of frames and distinguishes the parts of the frame that move or stay the same. If a part of an image is static over the series, then there is no need for that part of the frame to be changed when displayed. Conversely, parts of the screen that move have their movement recorded in motion vectors. Motion vectors describe the transformation of a macroblock from one image to another. Macroblocks are blocks of pixels used in the temporal compression. The typical size is 16x16, but it can depend on the coding profile and codec

A temporal compression algorithm will generate a group of pictures (GOP) for a given set of frames. A set of GOPs combine to form a video stream. Each GOP is bookended by a reference frame, called an Intra-frame (I-Frame). An I-frame is a full image frame and does not require extra information for its reconstruction – although it is still passed through a discrete cosine transformation (DCT) algorithm. In between the two I-Frames are intra-frames.

Intra-frames are formed of the motion compensated difference - i.e. the motion vectors - with regard to the reference frame. There are two types of intra frame; P-Frames and B-Frames. P-frames – predicted frames – are generated by finding the motion vectors for each macro-block relative to the previously

decoded image. B-frames – bi-directionally predicted frames – use both the past and future frame to generate those motion vectors.

An example of the ordering of a group of pictures:

IBBPBBPBBPBBI

In summary, temporal compression dramatically reduces the size of a video file by removing the redundant motion information from a given series of frames.

2.2.3.2 - Spatial compression

Spatial compression, or intra-frame compression, is used on the I-Frames of a GOP. Using a similar method to most image compression formats, such as JPEG, to reduce the size of a single frame by transforming and quantising the data in the frame. A DCT algorithm divides a frame into macroblocks, in H.264 at a size of 4x4, and applies the transformation to the region. Resulting in a matrix of transformed coefficients, representing the image. Each block is subsequently passed through a quantisation process, in which the coefficients are divided by their corresponding element in a quantisation matrix. The resulting matrix is the data that is stored for the block. Decoding is the reverse process, with the stored matrix being multiplied by the quantisation matrix originally used. (Gent, 2019)

When storing the quantised matrix, it goes through a process of entropy coding. H.264 can be configured to use either Context Adaptive Variable-Length Coding (CAVLC) (Wikipedia, 2019) or Context Adaptive Binary Arithmetic Coding (CABAC) (Wikipedia, 2019). These algorithms losslessly reduce the amount of data used to represent the blocks of transformed coefficients, by rearranging the data into a predetermined format.

2.3 - Steganography

Steganography is the art of hiding data in plain sight. The technique has been in use for thousands of years to convey messages secretly, i.e. writing a message in invisible ink. In the digital age, steganography has huge potential as vast amounts of data can be hidden inside of digital files, such as images, audio, and video – among others. Typically, a person using a steganographic file will have no idea that there is hidden information unless they know where to look. Steg-analysis tools can be used to detect whether there is any information hidden inside of a file. Therefore, steganography is often combined with encryption techniques.

In this project however, the data does not need to be encrypted. Data secrecy is not important as steganography is being used as a channel to “smuggle” data in the video stream.

The method of steganography applied to video files is different depending on whether the file is compressed or not.

2.3.1 - Uncompressed Video Steganography

Uncompressed video files can be treated as a series of raw images. Least significant bit (LSB) steganography can be used to insert data into the frames. This technique is the simplest of video steganography techniques. In this process, a pixel is extracted from a frame.

A pixel is typically made up of three colour or chroma channels depending on the colour space, i.e. RGB or YCbCr. One or all of these channels are used to hide data. An LSB algorithm extracts the channel data and changes the least significant bit – the furthest to the right – of a pixel byte to our desired data. Once changed the channel is written back into the pixel, and subsequently to the frame. Although this changes the pixel data of the frame, the image of the frame is unaltered, and a human eye cannot tell that there is steganographic data stored inside. Reading the data from the frame is the reverse process.

2.3.2 - Compressed Video Steganography

Compressed video files are more complex to deal with. The coding algorithms used means that the method used for raw video would lead to data being lost. However, there are multiple vectors to approach this problem as stated in (Ramdhan J. Mstafa, 2017). They suggest five methods for steganography.

- Intra-Frame Prediction
- Inter-Frame Prediction
- Motion vector
- Transform Coefficients
- Entropy Coding via Cavlc and Cabac

These methods are all more complex than the LSB method used for raw steganography. Most of the listed methods require the underlying codec to be changed to insert data.

2.4 - Market research

2.4.1 - Users

There are two groups of users that Lodge is aimed at; producers, and consumers. Producers will be technically competent and will have used editing

software before. They will use the Lodge encoding software, to generate, edit, and embed subtitles inside video files.

Consumers will be the layperson that uses subtitling for enhancing their experience of the video content. They will use primarily the Lodge decoder software. The end user experience for them should have the least resistance and ideally be invisible to the user. This will mean that the software should be bundled inside a video player or provided as a plugin. As building a video player from the ground up is not in scope for this project, proving that a subtitle file can be decoded is sufficient.

Producers include:

- Stenographers
- Video Editors

Consumers include:

- Deaf or hard-of-hearing people
- Foreign language speakers
- Layperson

2.4.2 - Competitors

As mentioned before YouTube is one of the largest competitors for Lodge in the market, especially for digital video, as according to Alexa they are the second most trafficked site in the world (Alexa, 2018).

YouTube provides captioning software for their online videos in their Creators Studio (Youtube, 2018). This software provides a number of different ways to add captions to a video; one can either upload a file, transcribe with auto-timings, or transcribe and set timings. The Creators Studio has a very clear interface, which is easy to use and understand by the layperson.

Netflix, another huge online video platform, lists 8 pieces of software that it recommends people use when subtitling for Netflix videos (Netflix, 2018). Each of these pieces of software perform a similar function and output formatted subtitle files.

However, from my research I have not been able to find a software product that incorporates both subtitling and steganography. This separates Lodge from the competition, as it will provide a combined file for both subtitles and video.

3 - Methodology and Planning

3.1 - Plan

The project will have three deliverables; a C++ library, a CLI tool, and a GUI application. All of the software produced will provide both functions of the steganographic process, read and write. The C++ library is intended for use in integrating Lodge into another program and gives access to methods directly. The CLI tool is a simple wrapper over the library giving advanced users the ability to use Lodge in a script. The GUI application is intended for use by non-technical users, such as editors or a consumer of a video.

The steganographic process will "lodge" subtitle files inside the frame data of a video, which can later be extracted to a subtitle file. This will solve some of the issues with the current methods for distributing subtitle files, i.e. not burning subtitles on the image of video frame and having support across multiple video formats. Least significant bit steganography will be used on the frame data and depending on the coding performed on the video will determine what sort of data will be manipulated in the frame.

The project will be undertaken in two phases, the first one delivering an MVP product and the second phase building on-top of the MVP to build a more complete workflow. An MVP product ensures that a useable application will be delivered at the end. With the second phase giving a more fleshed out product with more features.

Drawing from my experience in my industrial year at the BBC, I plan to build this project with robust frameworks and tooling surrounding it. Using industry practices such as using a continuous integration (CI) server and logging frameworks.

3.2 - Methodology

This project will use an agile approach to software development, which places an emphasis on early and continuous delivery and developing features rapidly. With the agile approach, testing is completed during the same iteration as writing the code, catching bugs in the program early and often – building up reliability in the software. This is a crucial process for this project, especially as a solo developer, as the typical testing phase at the end of a waterfall style project could find fundamental flaws in the program, requiring a substantial amount of time to re-write.

An agile planning approach called Epics will be used in this project to structure feature development. An epic is a large body of work that is broken down into small tasks or stories. In this project an example of an epic would

be an MVP product. This style of project management allows for clear, independent deliverables to be set for each epic of a project. These siloed deliverables give the opportunity to work on different features at will.

3.3 - Requirements

Below is a list of requirements formulated throughout my research. They are categorised with the MoSCoW method (Must have, Should have, Could have, and Wish for).

PRIORITY	REQUIREMENT
MUST	A frame header must be inserted when writing to the data.
MUST	Lodge must be compiled to run on MacOS.
MUST	Lodge must combine a subtitle file with uncompressed video files.
MUST	Video output from Lodge must not have visibly changed.
MUST	The file extension of the input subtitle must not change through the encoding process.
MUST	The video must have no visible change when subtitles are merged in to it.
MUST	The decoder must not interfere with video playback.
MUST	Lodge must not overwrite the original file; either subtitle or video.
MUST	Subtitles must be toggle-able.
MUST	The encoder must have a GUI for ease of use.
MUST	The encoder must be able to specify the time where the subtitle is to be displayed from.
MUST	Lodge process must be easier to use than existing methods.
SHOULD	Lodge should support 2 or more subtitle formats.
SHOULD	Lodge should support one compressed video format, such as H.264.
COULD	Lodge could support different fonts for stylistic purpose.
COULD	Encoder GUI support real-time transcription.
COULD	Lodge could be compiled to run on multiple different operating systems (Windows, Linux, BSD.)
WISH	Lodge supporting multiple subtitle tracks.
WISH	Adding non-subtitle features, such as hyperlinks.
WISH	Web based decoder.
WISH	Lodge work with every popular video format.

3.4- Epics

Following from the requirements listed above, epics can now be defined. Each epic being a sizeable chunk of work that can be split down into tasks or specific user stories.

3.4.1 - Minimum Viable Product

> *As a video editor, the Lodge encoder should be able to merge a basic SubRip subtitle files with my video file.*
>
> *As a video watcher, the Lodge decoder should be able to recognise merged video files and display the subtitles on screen.*

Deliverables:

- Basic subtitle encoder
- Basic subtitle decoder
- Lodge header format

The goal of this epic is to quickly build a basic product that **is** usable by end users and fulfils most of the MUST requirements. However, requirements that aren't strictly necessary for the first iteration will be left to a later epic.

Initially only uncompressed video will be supported by Lodge.

1.1: Task - Lodge header format

Deliverables: A header format for steganographic data in video

Details:

To ensure data integrity during encoding/decoding of the subtitle file, a header should be written at the top of the frame. It will also be used to convey information about the subtitle file.

Enclosing the header information will be some identifiable data to ear mark that it was made by Lodge and not just some random data.

1.2: Story – Merging video and subtitles

As a video editor, I want to merge a subtitle file with raw uncompressed video.

Deliverable: Basic subtitle encoder

Details:

Encoding data in raw video will require LSB replacement in video frames.

This story requires building an LSB encoder/decoder, and the extraction of video frames.

Subtitle must be encoded with ASCII characters.

1.3: Story – Extracting subtitles

As a viewer I want to be able to extract a subtitle file out of a Lodge encoded video

Deliverable: Basic subtitle decoder
Details:
The decoding process is the opposite of the encoding, reading the LSB from the frame data.

Successful extraction of the subtitle file from a steganographic video file is the main criteria for this.

Subtitle must be encoded with ASCII characters.

3.4.2 - GUI

> *As a subtitle editor I would like to use the Lodge encoder to view and edit subtitles in a GUI.*

Deliverables: A GUI on top of the encoder
Details:
To make this product appealable to the non-technical end user, a user interface must be created on top of the encoder. This epic covers the introduction of a GUI and gives a breakdown of each component that will be built.

2.1 - Story – Basic Encoder GUI

> *As a user that is not familiar with a CLI, I should be able to use a basic GUI version of the Lodge encoder.*

Deliverables: A basic GUI that wraps the Lodge encoder.
Details:
A basic GUI wrapper around the Lodge library, taking in a subtitle file and video file and spitting out the combined one.

2.2 - Story – Basic Decoder GUI

> *As a user that is not familiar with a CLI, I should be able to use a basic GUI version of the Lodge decoder.*

Deliverables: A basic GUI that wraps the Lodge decoder.
Details:

A basic GUI wrapper around the Lodge library, taking in steganographic video file and spitting out the combined one.

2.3 - Story – Transcription GUI

When using the Lodge encoder I should be able to edit and view my subtitles

Deliverables: A text editor that supports Lodge style subtitles
Details:
This story involves just adding a basic text window that shows a subtitle file with the timestamps in order for a layperson to be able to edit the subtitles easily.

3.4.3 - Video format

Lodge should support more video formats

To be a truly universal tool Lodge should support a wide variety of video formats. Each will bring its own complexity and will require extension of the decoder. Due to the complex nature of compression algorithms, and the limited timeline of this project, only H.264 will be attempted.

3.1 - Story – H.264 support

As a producer of video content, I want Lodge to be able compatible with H.264 compressed videos.

Deliverables: Updated encoder/decoder that will be able to handle compressed H.264.
Details:
H.264 video is used universally across the video industry, with more than 80% of video produced using this codec (encoding.com, 2019). It is a lossy compression format and will require a different type of steganographic technique to the one used for the raw video.

As discussed during the research section, there are a few different techniques on how to achieve this. In this story, a decision will be made on the technique to use.

3.2 - Extension - AV1 support

As a consumer of video content, I want Lodge to be compatible with AV1 compressed videos.

Deliverables: Updated encoder/decoder that will handle AV1 compressed video.
Details:
AV1 is a new codec written by an industry group to provide highly compressed video for web distribution. Supported by major companies such as Google and Microsoft, it is set to be a new standard for the web video.

A similar technique to the one used in the H264 story, will be used here.

3.5 - Changes from initial plan

The initial planning document is included in Appendix A, below is the updated requirements and plans.

During the initial planning stage, I had assumed that in order to embed any format of subtitle file the encoder would first have to convert the file to a common format. Upon further consideration I found that this would add unnecessary complexity to the process. Instead the files would be embedded as they were. This removes the requirements around adding a transformation step to each type of subtitle file and leaves the subtitle handling to the end video player.

Another design change was how the subtitling would be decoded. The initial plan was to have the subtitles being decoded in real-time as a frame was displayed. However, this would have unnecessary overhead to video playback. Therefore, the decoder runs once before the video file is played and then the subtitle file is passed to the video player as normal.

Not specifically mentioned in the initial plan, but became necessary after a while, was a way of identifying that a file had steganographic information inside it. A header was designed to contain vital information regarding the subtitle file. This header is the first information steganographically embedded into the file and contains vital information about the subtitle file. Information such as bits written to the file, filename, total number of frames written to, current frame 0 indexed. In a way this is the specification that was originally envisaged.

A revision was made to the original MVP goal of having subtitles display. The new goal was to enable extraction of uncorrupted data from the video file. I made this revision as I believe it fit better to the idea of producing a minimum viable product. With Lodge's simplest function being to insert and extract subtitles from a given video file.

3.6 - Software

Researching the software that would be used in the project is a crucial step in the project development cycle. The primary decision was the language to choose. Under consideration for the language was ease of use, barrier to entry, library support for video encoding/decoding, cross-platform support, extensive and well supported GUI library, and tooling ecosystem.

3.6.1 - Video coding

The most important of those being the library support for video encoding/decoding. One standout project was FFmpeg, which is widely used in major software projects such as YouTube and iTunes. The website states that FFmpeg is "a complete, cross-platform solution to record, convert, and stream audio and video." (FFmpeg, 2019). Providing support for a wide variety of video containers and formats. It is the natural choice for the video work in this project.

FFmpeg is written in a mixture of C and Assembly. There are bindings to this software in other languages such as Java, Go, or Rust. However, to integrate most easily with the library either C or C++ must be used. This was an important factor as I did not want to be confused by an abstraction layer on top of FFmpeg.

3.6.2 - Language

I chose to use C++ for a number for reasons. Firstly, most of my development experience had been with Java, so the barrier to entry was lower than moving to C as there were familiar concepts such as OOP. Secondly, I had recently finished a module where I had written a significant amount of C++ and wished to continue with it. Lastly there is a huge community and ecosystem surrounding C++.

3.6.3 - GUI

The GUI framework chosen was Qt, a massive cross-platform framework. Written in C++ and used extensively by both hobbyists and in the software industry. Other frameworks such as Juce were considered, however the tooling and tutorial support from the Qt Framework were un-paralleled.

3.6.4 - Miscellaneous

Controlling dependencies and build processes was identified as another area to investigate. CMake is a build tool that covers the build and release process, and dependency management. Used widely in C++ libraries, making it easy to integrate with.

Testing is a vital part of software projects, ensuring reliability and consistency in tooling. I chose the Catch 2 testing framework, a single header testing library that provides support for multiple testing styles such as BDD and unit testing. In addition to this I decided that it was important to include a continuous integration (CI) service with this project. Not only would that assure the project could be built by someone else but would also automate the testing process. Travis CI was chosen as it was free for open source projects and had easy integration with GitHub.

3.7 – Gantt Chart

Commencing on the week of 26/09/2018 and ending on 25/04/2019

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Basic product idea																														
Preliminary research																														
Requirements gathering																														
Research report																														
Project planning																														
Epic planning																														
Investigation into language/frameworks																														
Architecture design																														
Further research																														
Project set up																														
MVP: Lodge header format																														
MVP: Merging video and subtitles																														
MVP: Extracting subtitles																														
GUI: Basic GUI																														
GUI: Subtitle GUI																														
Video: H.264 codec																														
GUI: Transcription GUI																														
Catch up/clean up session																														
Documentation and testing																														
Folio Page																														
Examiners report																														

4 - Implementation

The project progress was a tale of two halves. As stated in my plan an MVP product would be produced, with a second phase focussing on expanding features and covering more complex topics. The goal being that there would be at least be a product delivered at the end – albeit minimally featured. On reflection this was a good idea, giving me the assurance that I would have a final deliverable no matter what state the second phase of the project would be in.

4.1 - MVP Epic

The MVP stage covered a few product deliverables; an encoder, a decoder, and a lodge header format. The encoder and decoder were quickly set up in the same library – as to be reusable to the GUI. Raw video is the only consideration in this stage.

4.1.1 - Architecture

Prior to writing code for this stage, architectural designs were drawn up. Using a whiteboard I was quickly able to prototype the data flow and classes required for the MVP. It proved useful to do these diagrams as the logic of the system could be visualised and rearranged as problems arose. Considerations were made to the libraries used such as FFmpeg and the types of data being used in Lodge.

The only steganographic technique used in the MVP will be least significant bit (LSB) substitution on video frames. The LSB substitution process should not care where the data passed to it comes from, and be a pure function. The purity of the process guarantees the integrity of the data transformation, by not using the state of the system. Reflecting that in the design, LSB is performed by a static class that has a private constructor. The LSB class cannot be instantiated, therefore there is no state to modify.

LSB substitution sets the least significant bit of a byte – the last one- with a replacement bit. When writing to the byte, the LSB should be flattened – set to 0 – to give a clean slate for writing data, otherwise replacing a bit 1 with a 0 will not result in a change. Therefore when the data needs to be read back from a byte, you simply read the LSB from each byte written to.

In the case of Lodge, the LSB substitution process will occur for the amount of characters inside the subtitle file. The characters are converted to their byte representation, and each of those bits substitute the LSB of a colour channel of a pixel inside a frame – diagram below.

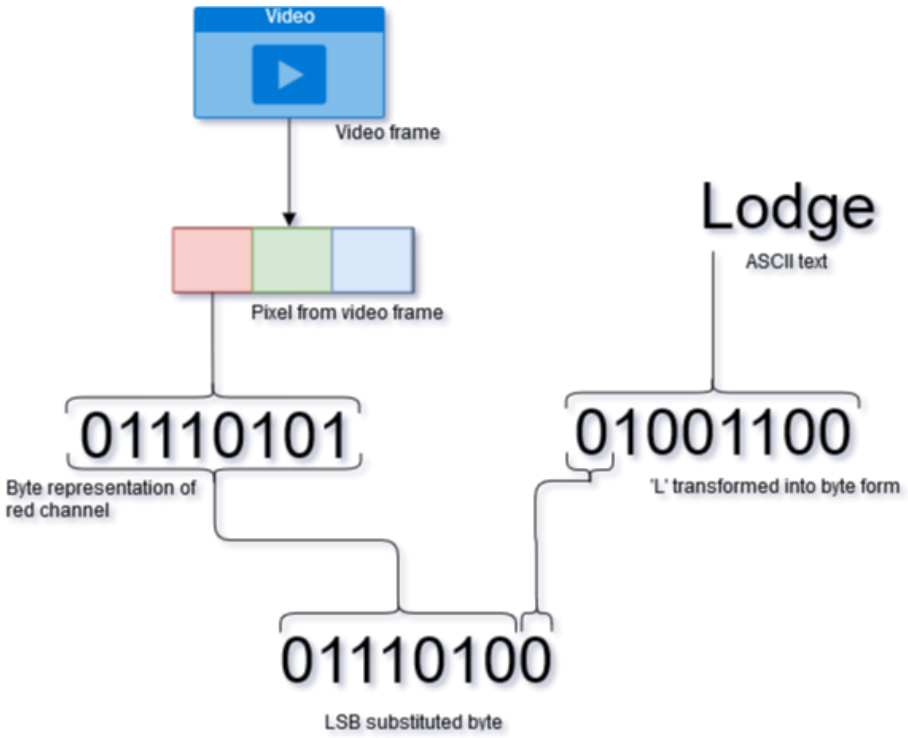


Figure 4 - Least Significant Bit Substitution

The main source of data in Lodge will be streams; a text stream representing a subtitle file, and video data streams. Subtitle files are text of an arbitrary length and by reading the subtitle as a stream, the file is not stored in memory, which reduces the resources used by Lodge.

Coordinating the streams is a crucial part of the design – if handled improperly, the data could become corrupted. Contributing to the complexity, a new video file has to be created with the steganographic data embedded which means another stream to handle.

There are three streams of data to handle when writing steganographic data to a video file (input video, output video, input subtitle), and two streams when reading subtitle files from a video (input video, output subtitle). The input video will be transcoded to output file. Therefore they will both have the exact same properties, except the frame data in the output stream will have the steganographic data hidden inside. This requirement for the input/output to have the same properties led the design to have a video class control both streams -reducing errors that otherwise could happen with multiple video objects. On instantiation the video class initialises both streams, depending on whether it will be reading or writing.

The basic data flow for writing a subtitle file to a video file is shown in Figure 5 and described here. The input and output video streams are initialised and the subtitle file is prepared to be read from. Once it has all been initialised, a frame is extracted from the input stream. The frame data is copied to the new output frame. Then the subtitle file is read from, retrieving a vector of characters for a line. This vector is transformed into its bit representation. With this bit data, least significant bit substitution is performed on the first colour channel of a pixel from the frame. This continues for the number of characters in the subtitle file. Once completed the output frame is written to file.

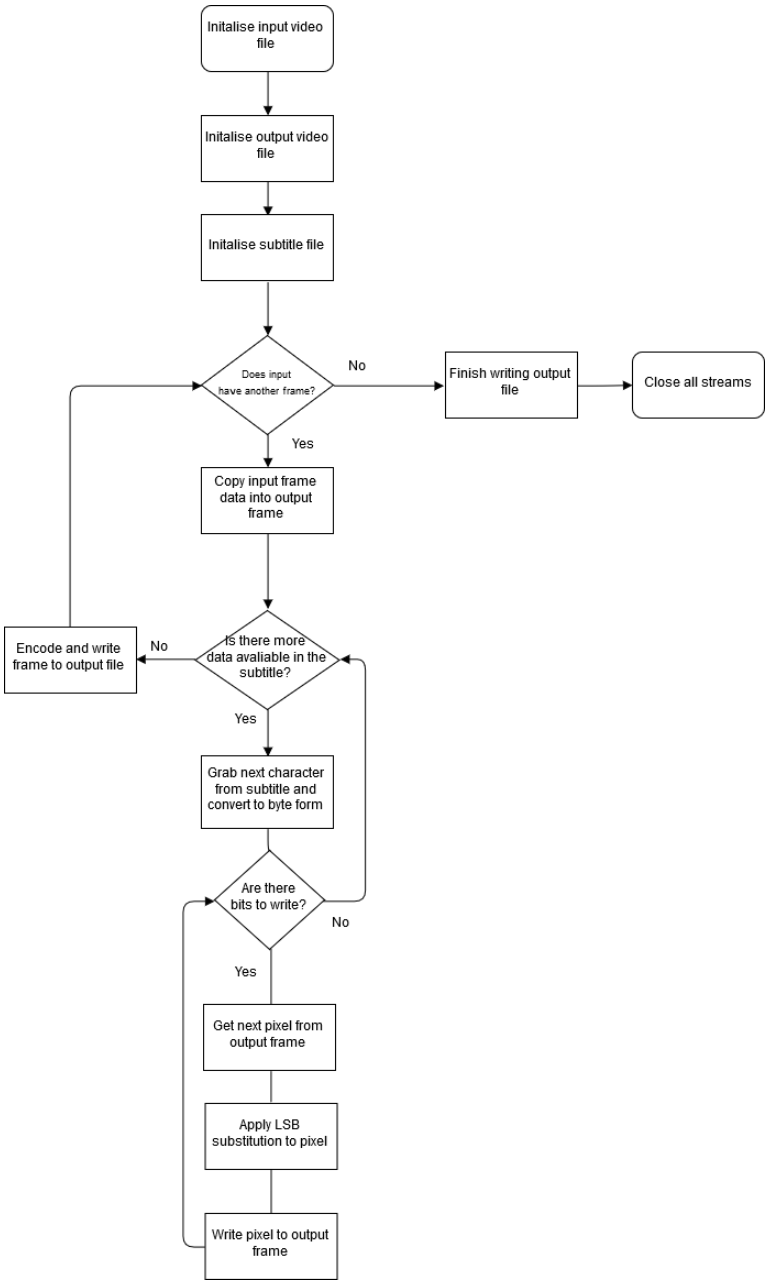


Figure 5 - Write data flow in MVP product

For reading a subtitle hidden in a video file, the process is much simpler. An input stream is created for the video file and an output subtitle file is setup. Frames are extracted from the video stream. Lodge then checks that the video frame contains Lodge compatible steganographic data, by reading three characters from the frame. If successful, Lodge then reads out the rest of the characters and writes them to the subtitle file. The process is described in the Figure 6.

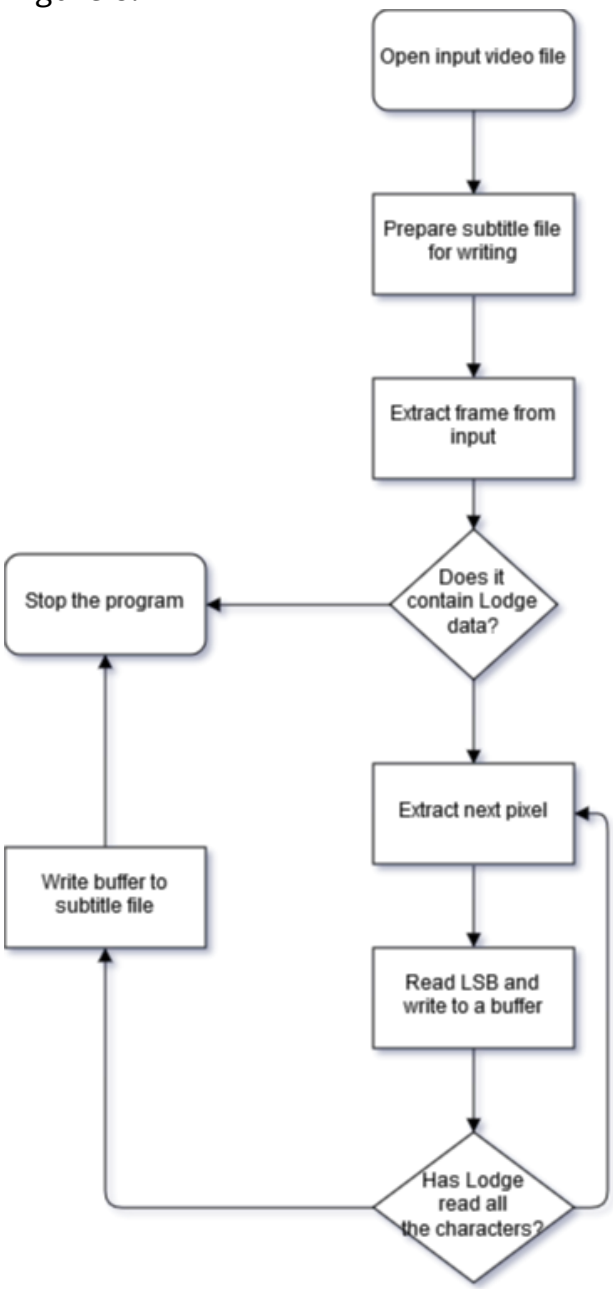


Figure 6 - Read data flow diagram in MVP product

From these flow diagrams I could model a high level class diagram, pictured below

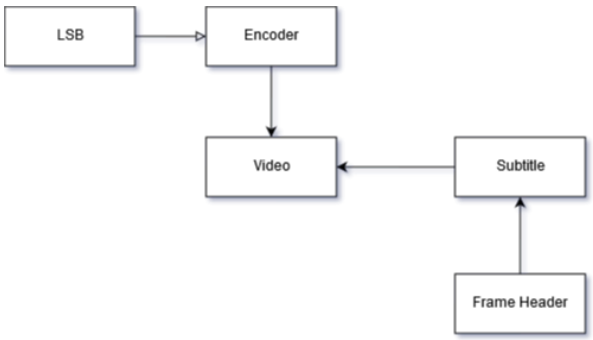


Figure 7 - High level class diagram of the MVP product

4.1.2 - Lodge header

As identified above, for raw video data a frame is used to lodge data inside for later extraction. This brings up a problem when extract data from a video file. There is no indication of where the data is embedded – if it is at all. Therefore an identifier, such as |LODGE|, will be written to the top of each frame that the Lodge writes to.

In addition, this header can be used to convey information back to the Lodge decoder. Identifying the length of the data inside the frame is crucial to the decoding step, otherwise the decoder will run until the end of the frame – spewing out data that is irrelevant and corrupting the subtitle file. This is solved by adding a count of the data written to the header, which can be parsed out by the decoder. In a similar vein, the file extension of the subtitle is crucial – marking the extension allows subtitle software to recognise the file type.

This results in the design for the frame header:

|LODGE|(bits written to frame)|(file extension)|LODGE|

4.1.3 - Implementation

The two main stories in the MPV epic regard the encoder and decoder. Although distinct user stories, they are tightly coupled as the decoder proves that the encoder works. Therefore, both of these user stories were worked on in tandem, to ensure correctness.

Before the implementation of the design, however, I had to familiarise myself with the FFmpeg library and build a basic C++ project. FFmpeg is a huge library and approaching the project is quite intimidating. The library assumes a good foundation of knowledge of digital video and audio, with documentation being obtuse to users unfamiliar with those concepts –including myself. However,

there are numerous tutorials online to work through that introduce basic concepts in video and audio along, with basic FFmpeg code exercises[1]. For Lodge the main focus is on the libavcodec and libavformat portions of the library. These two parts work in conjunction to perform the coding and handle the formatting of the video files.

Understanding FFmpeg and how video files work was a difficult undertaking, exacerbated by the blurred lines between FFmpeg concepts and general video concepts. Initially I went in "blind" and tried to implement a simple program to save off video frames as PPM images. After struggling with understanding how the process worked, I took a step back and decided to gain a deeper understanding of video files. This was a good decision. I gained a clearer picture of video containers, video codecs, and how to interact with them; without being bogged down in implementation specifics.

On completion of the tutorials, I realised that I was approaching the implementation the wrong way around. By tackling the "top" of the project first it was difficult to handle the size of the problem and understand where to start. However, using a bottom-up approach ensures that the fundamental "transaction" of Lodge was valid and could be guaranteed to work via testing. Shifting focus to the LSB substitution implementation gave a simpler foundation that could be built on top of.

The next layer to build on top after the LSB substitution was implemented and tested, was the subtitle files. These subtitle files are represented in an fstream object and have been implemented to provide line-based access to the subtitle file – converting between the ASCII character and the binary representation. This transformation was test and verified to work. A frame header was generated by the subtitle file class on initialisation, containing the information discussed in 4.1.2.

The penultimate layer was building the video class and the associated transcoding function. The video class holds reference to the location of both video files and a subtitle object. Both video files are lazily loaded and are not initialised during object construction. This measure ensures that the video data only be loaded when needed. Following an example file provided by FFmpeg, the transcode function was implemented in the write method of the video file (FFmpeg, 2019).

[1] The tutorials used were https://github.com/leandromoreira/ffmpeg-libav-tutorial and http://dranger.com/ffmpeg/tutorial01.html

Finally, the Lodge classes were wired up with the read and write algorithms for LSB substitution. When wiring up I found that H.264 encoded files could be coded in a lossless mode, this allows for video steganography in H.264 encoded files. However this dramatically increases the size of the file, in my testing a 2MB video was transcoded into a 34.8MB file. Once completed test cases were added and used to verify the data that Lodge was writing steganographic data properly.

4.1.4 - Constructing a build system

A crucial part of this epic was setting up the software stack and build system/process. As stated in planning, CMake would be the build system for Lodge. CMake is different to the build systems that I have used previously in the Java ecosystem – Maven and Gradle. Both tools bundle building and dependency management together, using centrally managed repositories, such as Maven Central, to control and install dependencies/libraries However the C++ ecosystem is more fragmented. There is no equivalent to Maven Central, and dependencies are installed in different ways depending on the package, type of package supplied, and the operating system. CMake doesn't handle dependency installation Therefore scripts need to be written to install dependencies on each OS. MacOS is the supported OS in the MVP as this was the one on my development machine.

Homebrew is an add-on community supported package manager for macOS. If possible this would be used to install packages to the system. However, not all packages are able to be installed via brew and need to be fetched and built from source. CMake handles the build process and links libraries to the Lodge source code. Using CMake was a continuous learning experience, requiring restructuring of the CMake files and the project itself.

I used the continuous integration (CI) service, Travis CI, to build and test Lodge. Travis CI provides a platform – host machines - for building, running, testing, and installing programs from source code. A CI environment is a crucial step in modern software engineering, improving the testability and reliability of the project. Through Travis CI some bugs surfaced; including missing dependencies from the build scripts and failing tests.

4.1.5 - Conclusion

In summary, all the user stories identified in the planning stage were completed during this epic. Delivering a basic CLI tool that could read and write subtitle files to video files – converting compressed video to uncompressed video.

4.2 - GUI Epic

4.2.1 - Qt Integration and Basic GUI

Qt comes in two different flavours; Qt Quick (a declarative framework using JavaScript and QML –which can be supported by a C++ backend) and Qt Widget (a traditional desktop UI framework using C++ and .ui files). I chose to use Qt Quick with a C++ backend to bridge to the Lodge library. The declarative QML files of Qt Quick allow for more control over the style and layout of the program compared with Widget – which provides a more rigid set of UI elements.

Although Qt uses qmake as its build tool, there is also support for CMake. This made the integration of the Qt simple, adding a sub-project with the GUI source and resources – linking the lodge library to the GUI executable.

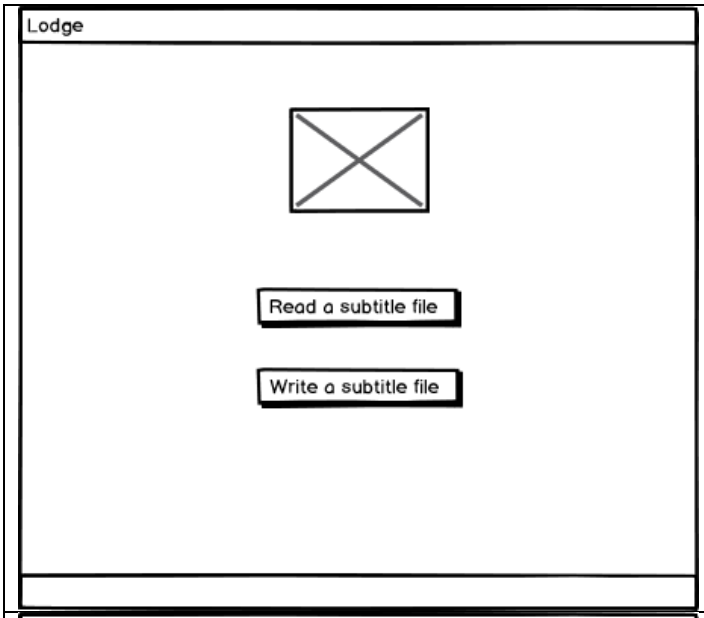
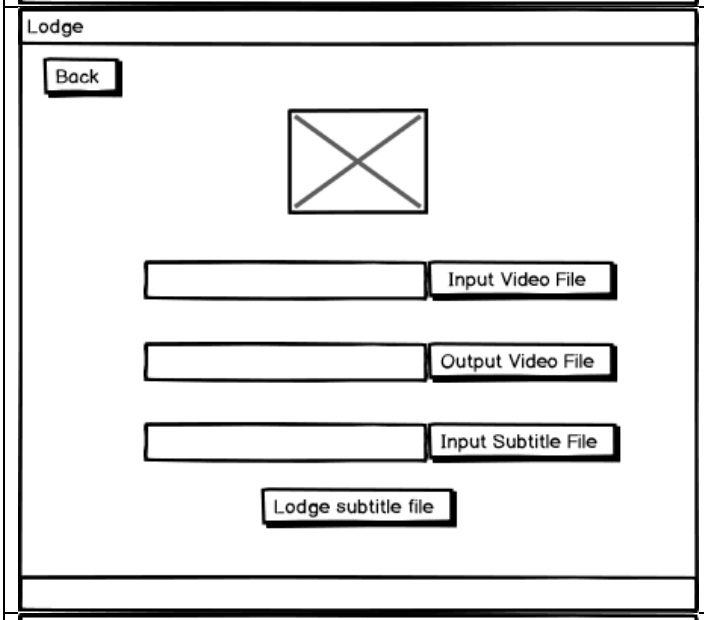
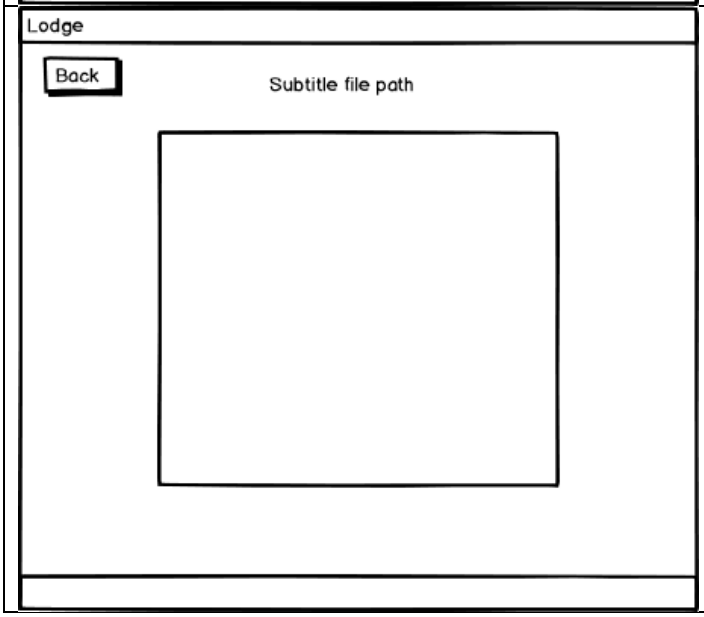
4.2.2 - UI/UX design philosophy

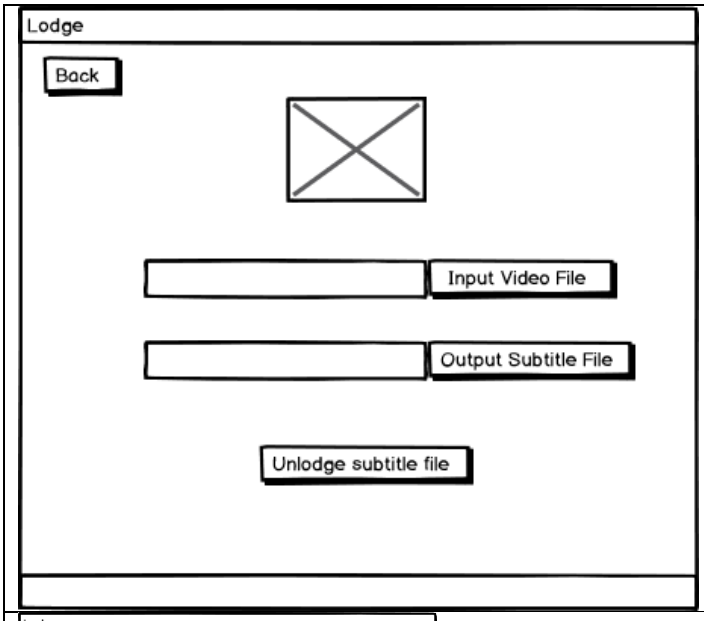
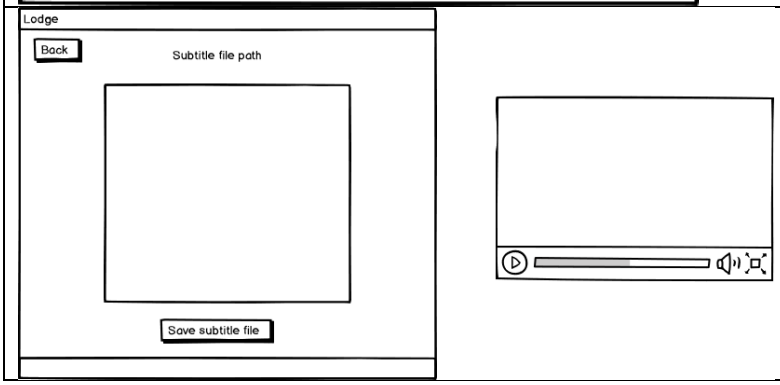
Although the GUI would not be that large or complex, I felt it was prudent to design the user workflow through the Lodge GUI. Through this design I developed a philosophy for the UI, which was to make the user interaction simple. The design mattered less than the actual user interaction with the software – although these are not mutually exclusive. This led to a few main points to consider while designing the UI:

- A user should be able to open the application and go through the process of extracting or lodging a subtitle file to/from video files, as quickly and easily as possible without confusion.
- There should be easily accessible signposts to push the user in the correct direction.
- The user interface should be clean, and free of noise.
- Errors should be fed back to the user with simple explanations, understandable to the lay-person and not just a developer.

4.2.3 - Designs

Lodge supports two workflows; lodging a subtitle file into a video file and un-lodging a subtitle file from a video to be played. Below are the designs that were drawn up during the

		Home screen
		Write subtitle view
		View of the extracted subtitle

	Read subtitle view
	Transcription view for writing subtitles, with a video player that launches.

4.2.3 - Implementation

Translating the design to code was a straightforward process. The Qt Creator allows for WYSIWYG editing of the QML files, and the design could be one to one mapped with this tool. With all the UI elements configured in QML, there is only a need for a minimal C++ backend. This backend bridged the gap between the Lodge library and the GUI.As the app as a whole was designed at the same time, all the views were implemented at the same time.

4.2.4 - Bundling the app

To complete the GUI program, the binary produced needed to be bundled into a macOS app. A macOS app is runnable by itself and, for this project, doesn't require adding an installer. These apps can be signed by an Apple Developer Certificate, certifying with Apple who owns the app and adding accountability. I chose against doing this as the app can be run without signing, and the overhead of managing a developer profile with Apple was not worth it.

4.2.5 - Conclusion

All of the user stories for this epic were completed. A GUI for use by both the video producer and video consumer, with the two main workflows implemented, has been produced.

4.3 - Compression Epic

There were many approaches suggested by (Ramdhan J. Mstafa, 2017), each with their own intricacies.

4.3.1 – Intra-Frame/Macroblock manipulation

The approach that interested me the most initially was the Intra-frame prediction. In his paper, he described the process of manipulating macroblocks to insert data into the LSB of a frame. The advantage of this approach is twofold; it requires no modification of the video coding process and it should theoretically work across all codecs that use macroblocks.

Although I found no paper directly researching this method, I found one with a similar technique (Riff, 2018). In the paper, Riff embedded data into all available frames by inserting one piece of data across a whole macroblock. I decided to investigate and implement an approach utilising the macroblocks of an I-Frame.

As part of the H.264 standard, macroblocks can be as small as 4x4 and up to 16x16 in size. One bit of the subtitle file is written to each sub-part of the macroblock, meaning that the increased amount of data lodged in could be up to 16^2. A typical 1920x1080 (1080p) frame with a 16x16 macroblock would only hold 8100 bits of data per channel, compared with RAW video steganography which could hold 2,073,600 bits. Using all three Chroma/colour channels would give 24,300 bits of data per frame, which is smaller than most subtitle files. A sample SubRip subtitle file for a Game of Thrones episode is 38,700 bits, for example. A total of 5 frames would need to be used to lodge the full file into the video.

4.3.1.1 - New algorithm

Lodge would need to be redesigned to spread the subtitle file across multiple frames. The read/write algorithms need to know the macroblock size and accommodate these into the functions. When writing the bit, the algorithm must write to every bit in a block sized square. When reading the bit, it must read every LSB in the block and work out an average. Below is the code for the updated write portion of the algorithm.

```
for (int iter = 0; iter < 8; ++iter) {
    for (int block_y = 0; block_y < block_size; ++block_y) {
        for (int block_x = 0; block_x < block_size; ++block_x) {
            unsigned char b;
            auto bit = bs[iter];
            if (bit) {
                b = static_cast<unsigned char>(1);
            } else {
                b = static_cast<unsigned char>(0);
            }

            auto _write_x = this->write_x + block_x;

            if (_write_x > fr->width - 1) {
                auto _write_y = this->write_y + block_size;

                this->write_y = _write_y;
                this->write_x = 0;
            }

            auto pos = fr->linesize[0] * (this->write_y + block_y) + (this->write_x + block_x);

            auto rChannel = fr->data[0][pos];
            lsb<unsigned char>::write_lsb(rChannel, b);
            lsb<unsigned char>::set_bit(rChannel, b, 1);
            fr->data[0][pos] = rChannel;
        }
    }
    this->write_x = this->write_x + block_size;
}
```

Figure 8 - Updated write function

4.3.1.2 - Frame Header

The frame header set out in the MVP would now need to be expanded. Frame total and current frame number were added, to help Lodge understand where in the video file it was. As each bit of data was now potentially taking up 16^2 more space in the frame, any reduction in the data written to the frame was needed.

|L|(number of characters in file)|(filename of subtitle file)|(number of frames)|(frame number - zero indexed)|L|

4.3.1.3 - Corrupted data

A bug occurred after implementing the new algorithm, where the data written to the subtitle file was corrupted. The start of the file had a lot of unexpected characters written to, and at regular intervals the characters extracted out to the file would be corrupted. After investigating both issues, I found that the encoding method was attempting to write past the end of the colour channel array on the height axis, causing undefined behaviour. In this case, the encoder would loop back round to the start of the array and continue to write data to the frame.

A similar effect was happening when the data was being read from the frame. The decoder would run past the bottom of the frame causing the undefined behaviour and return random data. This is what caused the corrupted data at regular intervals in the extracted subtitle file.

This bug was caused by the encoder and decoder using the number of bits used by the subtitle file field in the frame headers. This number was

generated by finding the pixel count of a frame and dividing that by the block size. However this did not take into account, the number of bits used by the frame header itself. Fixing this required making the frame header "self-aware" and replacing the field with a revised total taking in account the length of the frame header as a string.

4.3.1.4 - Turning on compression

Up until this point, Lodge was still only working with uncompressed video as I wanted to prove that the new algorithm works. When turning on compression for the sample files, I found that the data read out by the decoder was corrupted – more specifically the data inserted was not there. I investigated following the path of the frame and data up until being written to the output video file by FFmpeg and found that Lodge was correctly inserting the data. The data was being destroyed by the DCT transformation in the H.264 codec.

I tried to find solutions to this problem for a while, by playing around with the block sizing and scouring the internet for ideas. However I wasn't able to find a solution for this method of steganography. I decided that I would have to try a different method.

4.3.2 - Motion vector LSB

Initially I had ruled out motion vector-based steganography as it required modification of the coding algorithm of the video codec. This reliance on modifying specific implementations of the coding algorithms would make the project "heavy" and not easily portable. However the failure of the macroblock manipulation algorithm meant that other steganographic methods had to be explored.

I found a project by Edgar Liberis who had implemented a video steganographic tool using FFmpeg (Liberis, 2016). This source code was an invaluable source of information and formed the basis of my implementation of motion vector LSB steganography.

4.3.2.1 - Restructuring for FFmpeg

In order to get access to the motion vector data of any given frame, Lodge required access into the coding algorithm. A call-back to Lodge would be inserted at some stage along the H.264 coding pipeline, which returns the motion vector data to be edited and manipulated.

In the MVP product, Lodge relied on compiled binaries being installed via Homebrew for FFmpeg, which aren't distributed with the source files.

Therefore, FFmpeg must be downloaded and compiled locally with the call-back injected into the source code. This required a restructure of the project by creating a new bridge library to connect from FFmpeg to Lodge.

As FFmpeg is a C project the bridge must be compatible with C. By using a proxy, ffmpeg_connector.h, the C++ libraries of Lodge and FFmpeg could communicate and exchange data. From this proxy, the motion vector algorithm could perform the

4.3.2.2 – Searching for an injection point

The next step was finding a place in the H.264 coding pipeline to insert the call-back that would return the motion vector data. Liberis had identified a location for a call back for H.264 videos and I implemented them into FFmpeg. However with the video files I tested the call back in this location was never called. This may be due to a number of factors, including FFmpeg changing the coding process from 2016. Therefore I investigated other points in the pipeline that could be used. This was a significant undertaking as FFmpeg has over one million lines of code (Open Hub, 2019). While searching for a location, the project ran out of time and drew to a close – halting any further progress with this method.

The code can be found on the compression/h.264 branch of the project.

5 - Testing

Drawing from the knowledge gained in my industrial year at the BBC and the Software verification and validation module I took, I saw the benefit in having a robust testing regime. Both automated and manual testing methods were used to ensure the integrity of the Lodge project.

5.1 - Automated

Automated tests were written throughout the development process and were part of the process when finalising a new feature or bug fix. Unit tests were written using the Catch 2 framework and run via CMake. After each git commit to the project, Travis CI runs the build scripts and tests on a separate machine, adding an extra layer of defensibility to Lodge.

Some of the test cases run in automated testing are:

- Does Lodge perform LSB substitution correctly?
- Can a subtitle file be read from and written to?
- Does Lodge correctly convert a character to its bit representation?
- Can a subtitle file be lodged into a video file?

5.2 – Manual

Manual tests were performed as part of the feature development process and was used for parts of testing that aren't easily automatable. The testing included:

- GUI workflow testing
- RAW video testing – due to the size of raw files
- Running Lodge on another computer

5.3 – Notable bugs

Through manual testing, a major bug was found. When running either flavour of Lodge on a computer that was not my own, caused the program to crash on start-up. A linking error was occurring, where Lodge was trying to find the Boost, Qt, or FFmpeg libraries Lodge was expecting to find the dynamic libraries of these projects to be installed on the computer. This error was caused by the false assumption that all dependencies are packaged up into the binary file by default, as is with the Java ecosystem.

Adding the static version of Boost to the Lodge executable was simple, adding a line to the CMakeList file. For Qt it was more difficult as it does not install with the static binaries by default, requiring a recompile from source. The Qt

docs provided a guide on how to compile Qt statically, from which I implemented a script for use by anyone who needs to build the project.

When building with the static installation of Qt, CMake was unable to configure Qt for the project. CMake for Qt is a second-class citizen to the main build system for Qt, qmake. CMake required a more fragile build configuration, whereas the qmake works out of the box. The benefit of having the project build in one system had now been outweighed by the incompatibility of the static version of Qt with CMake. Transitioning from CMake to qmake was seamless and simplified the process of building the GUI.

Unfortunately linking FFmpeg statically was less trivial, as FFmpeg relies on platform specific libraries to be available and a full re-compilation as a static library. An attempt was made to link it to the project statically, but was unsuccessful, even after linking most of the dynamic Apple Framework libraries. This led to the decision to include the dynamic version of FFmpeg instead and require all users of the software to have FFmpeg installed on a system.

6 - Results

6.1 - Deliverables

Delivered from this project

- A C++ library that handles raw LSB steganography on video files
- A CLI tool that wraps the C++ library
- A GUI application that uses the C++ library

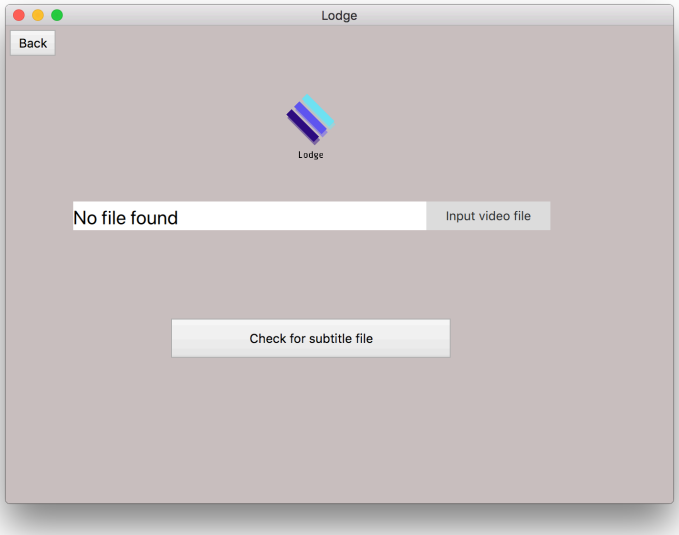
6.2 – Uncompressed video

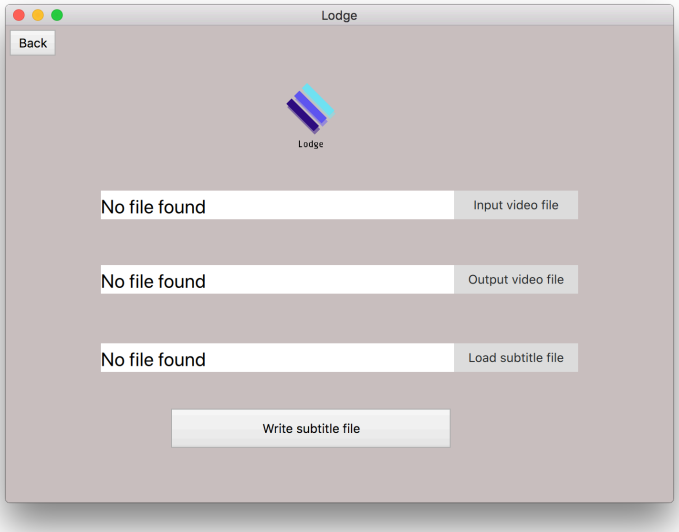
An uncompressed video can be successfully used as a distribution method for subtitle files. There is a negligible different in size between a Lodge encoded file and the original video file.

6.3 – Compressed video

H.264 encoded video can be still be used by Lodge, with a penalty on the size of the output file. From testing, a 2MB video H.264 transcoded with Lodge will expand into a 34.8MB video file. This is not a desired result for this project. Therefore, Lodge has not provided an implementation of compressed video steganography as a vector for distributing subtitle files..

6.4 – GUI

	Read subtitle screen, with a check button

	Write subtitle screen with all the inputs.

See Appendix 12.2 for more screenshots of the GUI app.

7 - Evaluation

The success of this project is mixed. The goal of distributing a video file with a subtitle lodged inside it has been completed, but only with uncompressed video. This opens up the ability to distribute subtitle files with formats that don't support subtitling. However, the lack of support for using steganography on compressed video without editing the codec demonstrates the limitations of this method as a distribution method for subtitle files. Coupled with the size of the Lodge library at 10 MB without other dependencies installed, makes this method not feasible for use in the distribution of online video as bandwidth reduction is a concern.

Two out of the three epics had all of their user stories completed and tested, however the compression epic is left in an uncompleted state. All main deliverables were achieved and the support for uncompressed video steganography fills the gap in the market for a distribution method for subtitle files.

Looking back on H.264 compression story I find problems with my approach to it. When starting the research, at the beginning of the story, I had a superficial understanding of the codec and how a compressed video steganography technique could be developed. This led me to rush into an implementation of the process without fully understanding the concepts of the underlying technology. It was only as the project was coming to an end that I started to understand the H.264 codec and how a solution to the problem could

be implemented. I believe that if I took the time at the start of the story to fully understand the H.264 codec I would have been able to complete the story.

Changing the direction of the H.264 story after failing to implement the macroblock-based algorithm was a necessary decision. However, this led me to invest a disproportionate time amount of time into the story. This had an impact on the project as a whole, not allowing myself time to further improve other features of Lodge.

The agile methodology taken in this project, particularly using epics, worked out well. Allowing the freedom to implement the features when the time was right for the project but guaranteeing the delivery of a final unit. The MVP phase gave the project a kick start with the major pieces of the project being developed and this ensured delivery of a final product. After completing the MVP epic, the agile approach and epics gave the flexibility to complete features independently of each other and work on them in parallel. Testing early and often was a good practice, giving the project a consistent source of truth. I believe that if I had used a waterfall method, the product would have been bound by deadlines that it could not keep and would have fallen apart when testing occurred at the end of the project

As a whole I am proud of the project and the progress I was able to make with it. At the start of the project I had a basic knowledge of C++ language and ecosystem, and little knowledge of how video technology works. By the end I developed a substantial understanding of both topics and I am confident in my skills in these areas. I believe that the code and structure of the project is of high quality and the integration with a CI service demonstrates good software engineering principles.

9 - Conclusion

From this research I found a limited solution for distributing subtitles with digital video. This solution uses least significant bit substitution on uncompressed video frames to embed subtitle files into a video file. A toolkit was developed to facilitate this process, containing a C++ library, a CLI tool, and a GUI application. However, the practical use of this method is limited as most video digitally distributed is compressed and this method of steganography will not work with it.

Additional research into using a steganographic technique for compressed video is incomplete and a conclusion on its practicality cannot be drawn.

Future work for this project would include:

- Additional research into the Intra-frame video compression technique
- Completion of motion vector-based steganography
- Web solution for Lodge

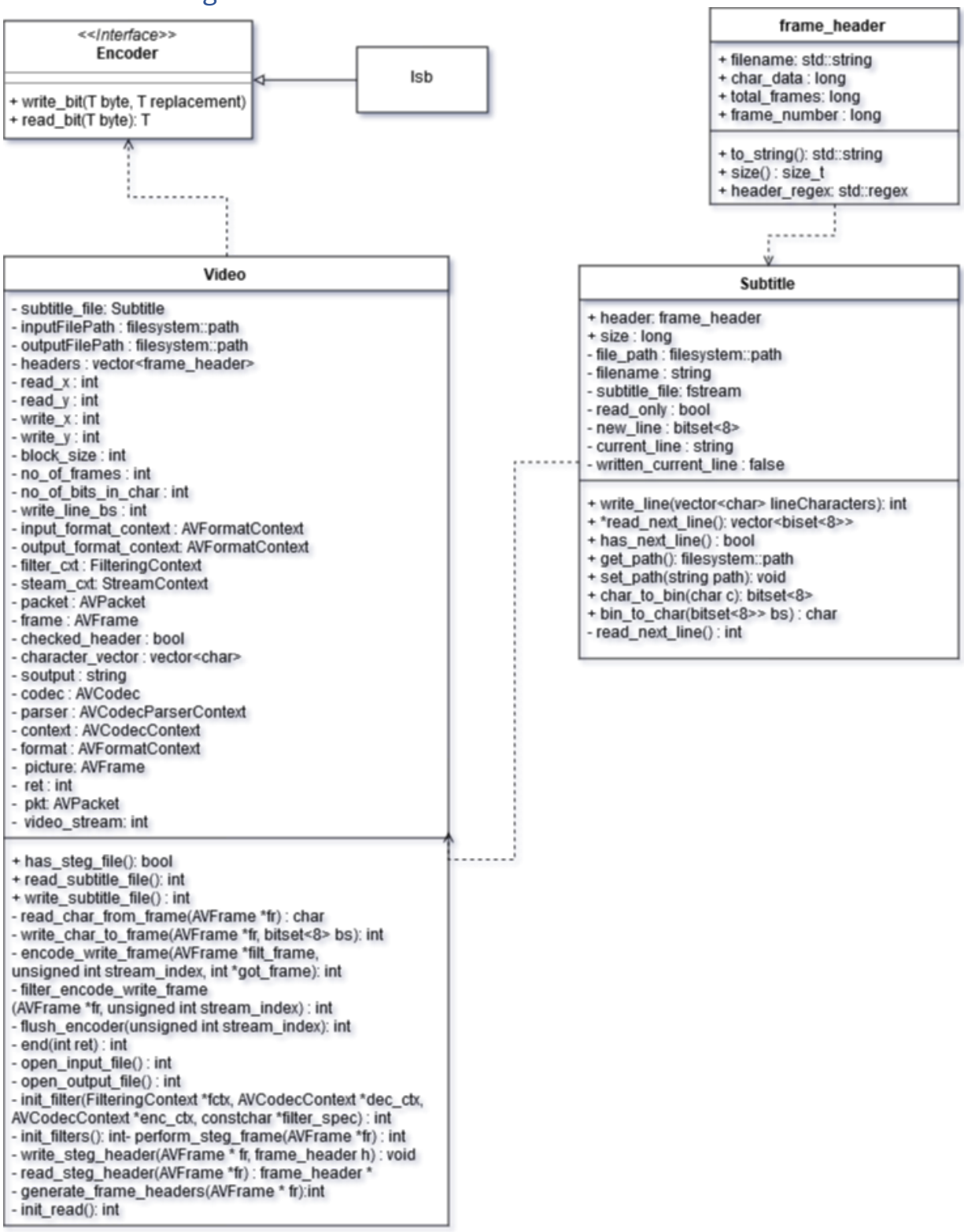
10- Bibliography

Alexa, 2018. *Youtube.* [Online]
Available at: https://www.alexa.com/siteinfo/youtube.com
[Accessed 5 Novemeber 2018].
Armstrong, M., 2016. *Automatic recovery and verification of subtitles for large collections of video clips,* London: BBC.
Bala, S., 2016. *H.264 is magic: a techinical walkthrough.* [Online]
Available at: https://sidbala.com/h-264-is-magic/
BBC, 2018. *How do I create subtitles?.* [Online]
Available at: https://www.bbc.co.uk/guides/zmgnng8
[Accessed 10 October 2018].
BBC, 2018. *Subtitle Guidelines.* [Online]
Available at: https://bbc.github.io/subtitle-guidelines/
[Accessed 25 October 2018].
Changyong Xu, X. P. T. Z., 2006. *Steganography in Compressed Video Stream.* s.l., IEEE.
encoding.com, 2019. *Global Media Format Report 2019.* [Online]
Available at: https://1yy04i3k9fyt3vqjsf2mv610yvm-wpengine.netdna-ssl.com/files/2019-Global-Media-Formats-Report.pdf
[Accessed 03 2019].
FFmpeg, 2019. *FFmpeg.* [Online]
Available at: https://ffmpeg.org
[Accessed 10 04 2019].
FFmpeg, 2019. *FFmpeg: doc/examples/transcoding.c Source File,* s.l.: s.n.
Gent, K. C. a. P., n.d. *dct.* [Online]
Available at: https://www.math.cuhk.edu.hk/~lmlui/dct.pdf
ITU, 2019. *H.264 : Advanced video coding for generic audiovisual services.* [Online]
Available at: https://www.itu.int/rec/T-REC-H.264
Liberis, E., 2016. *"GitHub - eliberis/movest: Video Steganography using motion vectors. Part II project, part of the University of Cambridge Computer Science course..* [Online]
Available at: https://github.com/eliberis/movest
Moreira, L., 2019. *FFmpeg Libav Tutorial.* [Online]
Available at: https://github.com/leandromoreira/ffmpeg-libav-tutorial
Netflix, 2018. *Technology Resources.* [Online]
Available at: https://partnerhelp.netflixstudios.com/hc/en-us/categories/202282017-TECHNOLOGY-RESOURCES
[Accessed 16 November 2018].
Open Hub, 2019. *The FFmpeg Open Source Project on Open Hub.* [Online]
Available at: https://www.openhub.net/p/ffmpeg

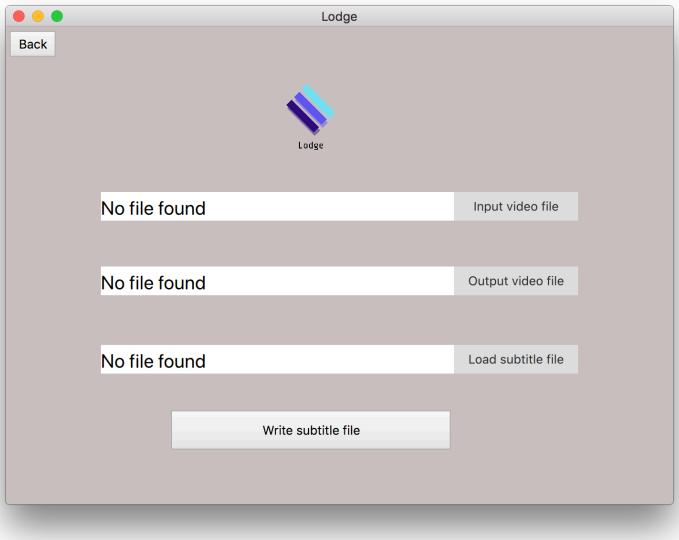
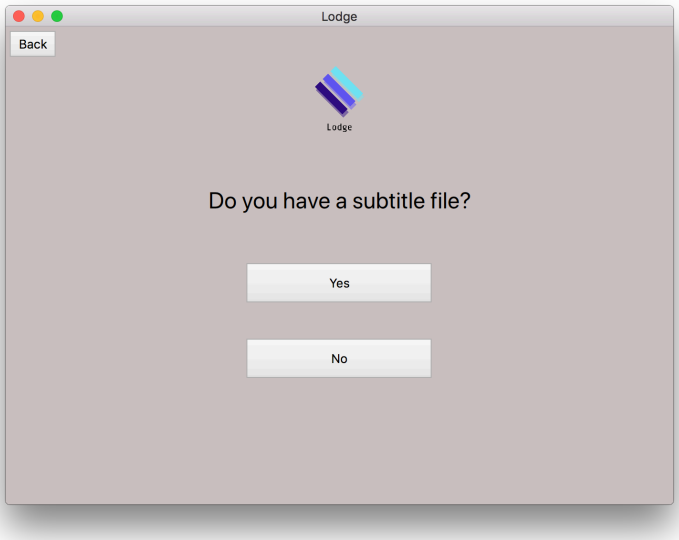
OpenStego, 2018. *Open Stego.* [Online]
Available at: https://github.com/syvaidya/openstego
[Accessed 25 10 2018].
Ramadhan J. Mstafa, K. M. E. E. A., 2017. *Video steganography techniques: Taxonomy, challenges, and future directions.* Farmingdale, NY, USA , IEEE.
Ramdhan J. Mstafa, K. M. E. E. A., 2017. *Video steganography techniques; Taxonomy, challengs, and future directions.* Farmingdale, NY, USA, s.n.
Riff, D., 2018. *Re-encoding persistent video steganography.* s.l.:s.n.
Wikipedia, 2019. *Context-adaptive binary arithmetic coding - Wikipedia.* [Online]
Available at: https://en.wikipedia.org/wiki/Context-adaptive_binary_arithmetic_coding
Wikipedia, 2019. *Context-adaptive variable-length coding - Wikipedia.* [Online]
Available at: https://en.wikipedia.org/wiki/Context-adaptive_variable-length_coding
Youtube, 2018. *Captioning.* [Online]
Available at: https://studio.youtube.com
YouTube, 2018. *Captioning.* [Online]
Available at: https://studio.youtube.com/
[Accessed 10 November 2018].
Youtube, 2018. *Subtitle Support.* [Online]
Available at: https://support.google.com/youtube/answer/2734698?hl=en-GB

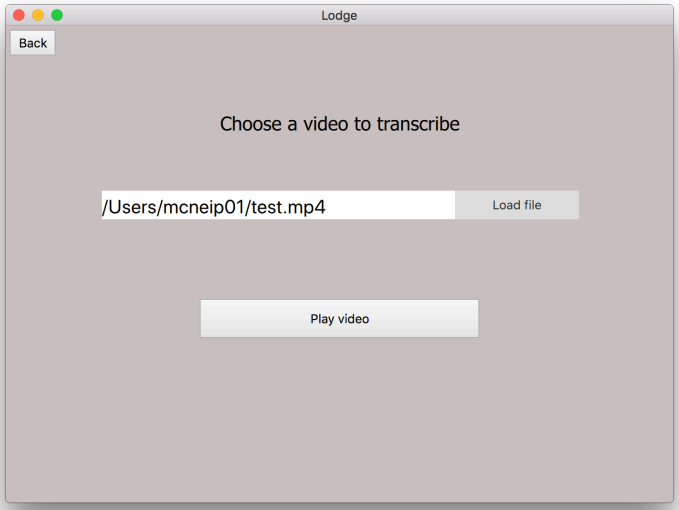
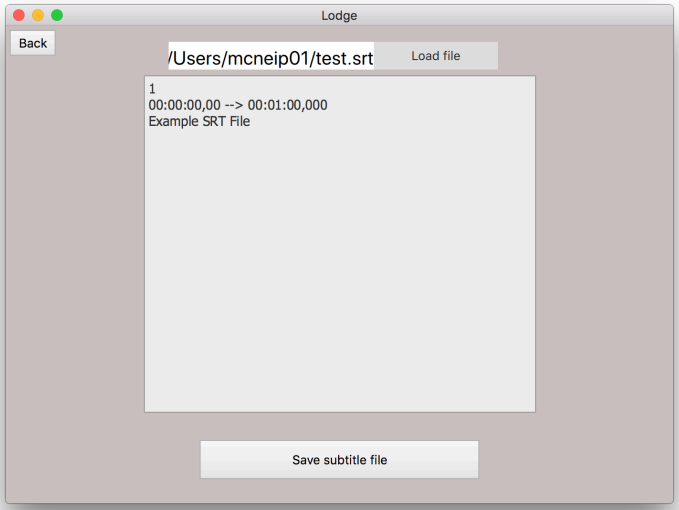
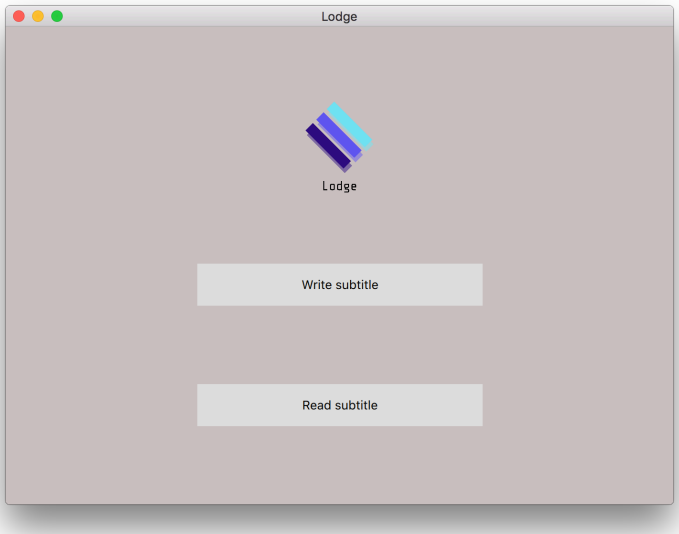
11– Appendices

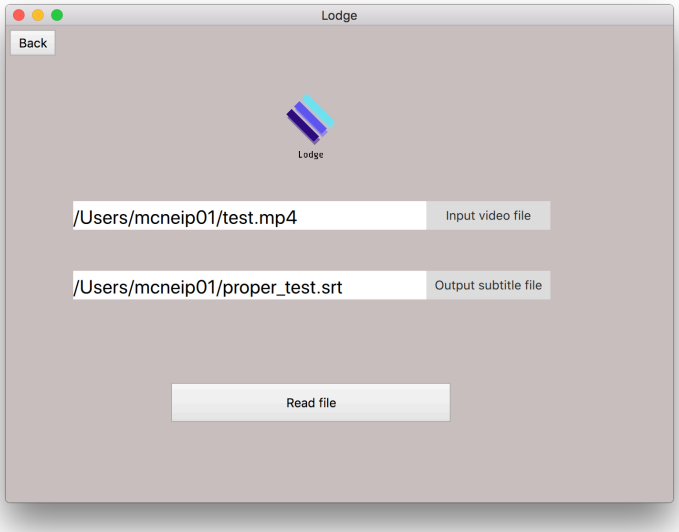
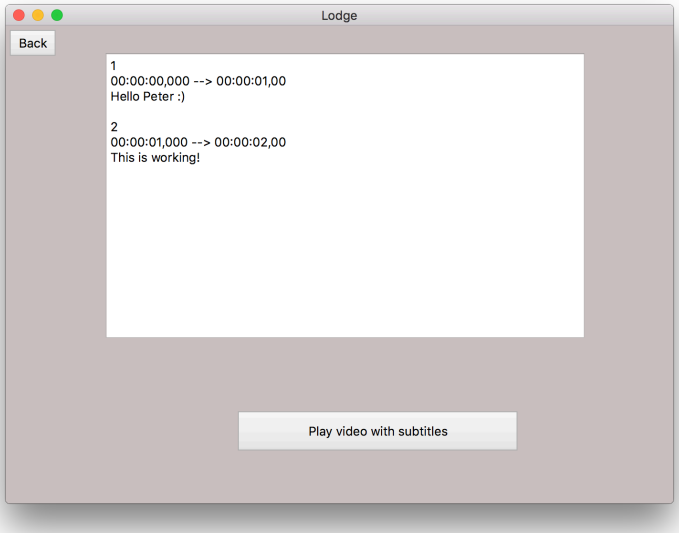
11.1 – Class diagram



11.2 – GUI Screenshots

	Write screen for subtitles
	Subtitle decision screen

	Transcribe choice screen
	Transcribing screen
	Main screen

	Found a subtitle view, with option to read the file.
	Extracted subtitle view, with option to play video with subtitle file.

11.3 – Git log

(Changyong Xu, 2006)

```
* ec2087b (HEAD -> master, origin/master) Clean up commit
* 7ecba26 Updated UI
* fc26bc3 Fields auto fill after transcription
* 0b15f22 Fixed bug when transcribing subtitles
* bf029b3 Added more tests
*   eeb11ba Fixed merge conflicts
|\
| * 29709b5 (origin/bug/dynamic-libraries, bug/dynamic-libraries) Updated failing tests
* | 4c63187 Fixed README image
* |   367fcb1 Merge pull request #11 from petermcneil/bug/dynamic-libraries
|\ \
| |/
| * 8073a17 Fixed some bugs and restructed the resources folder
* |   d1b7e9f (tag: v0.2) Merge pull request #10 from petermcneil/bug/dynamic-libraries
|\ \
| |/
| | * f5ce519 (origin/compression/h.264, compression/h.264) Proper ffmpeg installation
| | * 1a60d69 (origin/compression/h.264-own) FFmpeg connect works, but there is no indication where to find the mv tables
| | * 163ac4c Trying main branch
| | * 7345557 Fixed something
| | * c364263 Trying it on my own
| | * 1bbc126 Trying it on my own
| |/
|/|
| | *   4a75045 (refs/stash) WIP on bug/dynamic-libraries: 15f3a6f Qt is now statically linked to the project.
| | |\
| |/ /
| | * 2b09ca5 index on bug/dynamic-libraries: 15f3a6f Qt is now statically linked to the project.
| |/
| * 15f3a6f Qt is now statically linked to the project.
| * fb91430 Fix for overriding output names of subtitle files
| * 47a8cf4 Adding boost static linking
|/
| * 0886277 (compression/h.264-movest) Trying my best
|/
* bcb94ab Added release badge to README
*   75ba745 (tag: v0.1) Merge pull request #9 from petermcneil/bug/failing-ci
|\
| * f843225 C str for spdlog
| * e91690c Update dependencies.sh
|/
* c36b117 Fixed tests and cleaned up code
* 5f2dbeb Fixed build logo link
* 5e6920d Cleaned up some errors
* 71d8bff Re-structure lodge source code
*   f0db45b Merge pull request #8 from petermcneil/feature/compression-h264
|\
| * a5f6c22 (origin/feature/compression-h264, feature/compression-h264) Updated README and dependencies
| * 94a036d Fixed failing tests and updated logging levels.
| * c7ae108 Updated video file testing
| *   c39cd0f Merge pull request #7 from petermcneil/feature/more-advanced-gui
```

```
| |/ /
| | * 2b09ca5 index on bug/dynamic-libraries: 15f3a6f Qt is now statically linked to the project.
| |/
| * 15f3a6f Qt is now statically linked to the project.
| * fb91430 Fix for overriding output names of subtitle files
| * 47a8cf4 Adding boost static linking
|/
| * 0886277 (compression/h.264-movest) Trying my best
|/
* bcb94ab Added release badge to README
*   75ba745 (tag: v0.1) Merge pull request #9 from petermcneil/bug/failing-ci
|\
| * f843225 C str for spdlog
| * e91690c Update dependencies.sh
|/
* c36b117 Fixed tests and cleaned up code
* 5f2dbeb Fixed build logo link
* 5e6920d Cleaned up some errors
* 71d8bff Re-structure lodge source code
*   f0db45b Merge pull request #8 from petermcneil/feature/compression-h264
|\
| * a5f6c22 (origin/feature/compression-h264, feature/compression-h264) Updated README and dependencies
| * 94a036d Fixed failing tests and updated logging levels.
| * c7ae108 Updated video file testing
| *   c39cd0f Merge pull request #7 from petermcneil/feature/more-advanced-gui
| |\
| | * bcb25eb (origin/feature/more-advanced-gui, feature/more-advanced-gui) Fixed an error when running the decoder
| | * c9f52d6 Added a better icon
| | * 7b7a7f6 Added a transcription GUI
| | * 48d07bc Added some basic error handling
| | * 0d54cba Rearranged QML files to make the dir clean
| | * 41e3e78 VLC now opens the video if avaliable
| | * 28f1b49 Read subtitle file can now be played!
| | | * 73a25c7 (origin/feature/compression-h264-est, feature/compression-h264-est) New logos and FFmpeg submodule
| | |/
| |/|
| | | * 49866e5 Read subtitle file can now be played!
| | |/
| | *   fc52f11 Merged with current branch
| | |\
| | |/
| |/|
| | | *   8cafcd7 Merged with current branch
| | | |\
| | |/ /
| | | /
| | |/
| |/|
| * | 674a615 Attempts to fix data being written over frames
| * | 53f897b Frame header now is set on each frame
| | * adc8f87 Subtitle view
```

```
| | * bcb25eb (origin/feature/more-advanced-gui, feature/more-advanced-gui) Fixed an error when running the decoder
| | * c9f52d6 Added a better icon
| | * 7b7a7f6 Added a transcription GUI
| | * 48d07bc Added some basic error handling
| | * 0d54cba Rearranged QML files to make the dir clean
| | * 41e3e78 VLC now opens the video if avaliable
| | * 28f1b49 Read subtitle file can now be played!
| | | * 73a25c7 (origin/feature/compression-h264-est, feature/compression-h264-est) New logos and FFmpeg submodule
| | |/
| |/|
| | | * 49866e5 Read subtitle file can now be played!
| | |/
| | *   fc52f11 Merged with current branch
| | |\
| | |/
| |/|
| | | *   8cafcd7 Merged with current branch
| | | |\
| | |/ /
| | | /
| | |/
| |/|
| * | 674a615 Attempts to fix data being written over frames
| * | 53f897b Frame header now is set on each frame
| | * adc8f87 Subtitle view
| |/
|/|
| | * 1186071 Subtitle view
| |/
|/|
| | * 511f984 Subtitle view
| |/
|/|
| | * 0e97fff Subtitle view
| |/
|/|
* | 862cec3 Fixed tests to actually work
| | * 6eac3c6 Frame header now is set on each frame
| |/
| * df959a4 Work in progress towards multiple frames with data
| * b6f43a1 Changes to writing/reading algorithm
|/
*   6f0bbed Merge pull request #6 from petermcneil/feature/gui
|\
| * 3ae9b53 (origin/feature/gui, feature/gui) A lot of cleanup
| * 291047e Revamped the options when dealing with it via CLI
| * 09fe6cd Added a release process
| * fee6ffa Encode/Decode view completed MVP
| * e655543 Basic one screen GUI
| * f153cac Qt 5 dependency added
| * 057d443 Qt GUI project added
```

```
|/|
| | * 1186071 Subtitle view
| |/
|/|
| | * 511f984 Subtitle view
| |/
|/|
| | * 0e97fff Subtitle view
| |/
|/|
* | 862cec3 Fixed tests to actually work
| | * 6eac3c6 Frame header now is set on each frame
| |/
| * df959a4 Work in progress towards multiple frames with data
| * b6f43a1 Changes to writing/reading algorithm
|/
*   6f0bbed Merge pull request #6 from petermcneil/feature/gui
|\
| * 3ae9b53 (origin/feature/gui, feature/gui) A lot of cleanup
| * 291047e Revamped the options when dealing with it via CLI
| * 09fe6cd Added a release process
| * fee6ffa Encode/Decode view completed MVP
| * e655543 Basic one screen GUI
| * f153cac Qt 5 dependency added
| * 057d443 Qt GUI project added
|/
*   4144b36 Merge pull request #5 from petermcneil/feature/steganography
|\
| * 0b13fa8 (origin/feature/steganography, feature/steganography) Full subtitle file saved in a video
| * ec5dbc7 Steganography working on uncompressed video.
| * 23d054c Video Steg starting
|/
*   e5afc95 Merge pull request #4 from petermcneil/feature/subtitle-file
|\
| * 7c609a9 (origin/feature/subtitle-file) Subtitle File:
| * 8c9193f Subtitle File:
| * b798955 Subtitle File:
| * 2b8158c SUBTITLE FILE:
| * 59f95f0 Subtitle File:
|/
*   74aa329 Merge pull request #3 from petermcneil/feature/least-bit
|\
| * c498e8d (origin/feature/least-bit, feature/least-bit) Encoder least bit completed
| * b1d5135 Read LSB from an array
| * c055064 Replacement of LSB in arrays works.
| * 7beea0c Generic Bit Manipulation
| * f792054 Read/Write LSB
| * 69aee63 Fix centered logo
| * b8a7b93 Flattened final bit of a byte
| * 32dbef1 Encoder implementation:
|/
```

```
*   4144b36 Merge pull request #5 from petermcneil/feature/steganography
|\
| * 0b13fa8 (origin/feature/steganography, feature/steganography) Full subtitle file saved in a video
| * ec5dbc7 Steganography working on uncompressed video.
| * 23d054c Video Steg starting
|/
*   e5afc95 Merge pull request #4 from petermcneil/feature/subtitle-file
|\
| * 7c609a9 (origin/feature/subtitle-file) Subtitle File:
| * 8c9193f Subtitle File:
| * b798955 Subtitle File:
| * 2b8158c SUBTITLE FILE:
| * 59f95f0 Subtitle File:
|/
*   74aa329 Merge pull request #3 from petermcneil/feature/least-bit
|\
| * c498e8d (origin/feature/least-bit, feature/least-bit) Encoder least bit completed
| * b1d5135 Read LSB from an array
| * c055064 Replacement of LSB in arrays works.
| * 7beea0c Generic Bit Manipulation
| * f792054 Read/Write LSB
| * 69aee63 Fix centered logo
| * b8a7b93 Flattened final bit of a byte
| * 32dbef1 Encoder implementation:
|/
* 2f54bed No command line parsing in tests
* 9f2e9e2 Logging and Command line parsing
*   2923d5f Merge pull request #2 from petermcneil/feature/ffmpeg_hello_world
|\
| * 642361a (origin/feature/ffmpeg_hello_world) New tutorials followed:
| | * a28d08c New tutorials followed:
| |/
| * 4441513 Tests no longer rely on hardcoded path
| * 24a1ee9 Print out what Travis thinks it's doing
| * 03ebffb Upgrade not install boost
| * 65bf033 Bash doesn't like commas in arrays
| * e5fe1d0 FFmpeg tutorial complete:
| * 8c91b3f Fix typoe
| * 0fe2965 OS specific configure script
| * 634e28a A more sophisticated depenedencies script
| * d0f37e9 Install ffmpeg with brew on Travis
| * b203ace Add Main.cpp back in
| *   8d9dbdb Merged with master
| |\
| |/
|/|
* |   b5d63a5 Merge pull request #1 from petermcneil/feature/travis
|\ \
| | *   1efbc45 Fixing merge to master
| | |\
| | | * 68358c0 Create CONTRIBUTING.md
```

```
| |/
|/|
* |   b5d63a5 Merge pull request #1 from petermcneil/feature/travis
|\ \
| | *   1efbc45 Fixing merge to master
| | |\
| | | * 68358c0 Create CONTRIBUTING.md
| | | * 529e4e4 Initial commit
| | * 9848bbf Following first tutorial
| | * 9faf7aa Initial commit
| | * fa70fc3 (origin/feature/travis) export CMAKE_ROOT
| | * d9e506e Clear cache before running cmake
| | * df6a0c0 Remove old version of cmake
| | * ce0b391 Change bash conditional
| | * 928d3b4 Don't use brew to download
| |/
| * e569eb9 - File not directory
| * 2ddbc86 - Fix curl
| * d837ced - Fix curl
| * a225f9b - Ignore docs
| * f922c49 - No brew update
| * 881bf97 - Actually follow docs
| * 99f9f3c - Remove brew update
| * e9b1b4e - Rejigged CMakeLists - Updated travis.yml
| * d966c6b - Added build badge - Made configure and dependencies executable
| * 35b89f8 - Added a configure script
| * b833a34 Travis CI Integration:
|/
| *   2bf29a8 (refs/original/refs/heads/feature/ffmpeg_hello_world) Fixing merge to master
| |\
| |/
|/|
* | 8acf4be Create CONTRIBUTING.md
| * 80fefd5 Following first tutorial
| | * 4cd2699 Create CONTRIBUTING.md
| |/
|/|
| | * d262606 Following first tutorial
| |/
| | * e291a39 Following first tutorial
| |/
| | * 52ed380 Create CONTRIBUTING.md
| |/
|/|
* | 74bb745 Initial commit
 /
| * 8dbfae7 Create CONTRIBUTING.md
|/
| * c116531 Update issue templates
|/
* 55353a5 Initial commit
```

11.3 – Original Planning document

Introduction

Traditionally video and subtitle files have been separated; requiring distributing many files to play a video with subtitles or for the video editor to burn the subtitles into the video frame. This project, Lodge, aims to fix this by using steganography in a novel way to hide the subtitles inside individual video frames.

Lodge will consist of two software products for dealing with subtitling in videos. One program (Lodge Encoder) will be used by video editors or stenographers and will merge the subtitles into the video file. The other program (Lodge Viewer) will be used by the layperson while watching videos and will extract the subtitles from the video file and display them to the user. The two products together will provide a complete end-to-end workflow for subtitling.

In addition to the Encoder and Viewer, Lodge will deliver a data format specification. This specification will detail how the subtitles are merged with the video files, with the hope that other implementations could be made from the open source specification.

Simply put the goal of this project is to combine subtitles and video into the same file, without being burned into the video itself.

Research

Subtitling

Subtitles, or captioning, is text displayed that represents sounds made in visual media, such as videos or games. This sound can be speech, music, off-screen noise, ambient noise, foreign languages translated to the desired language, speech with an accent, and additional information to help the hard of hearing. There are two types of subtitling; open and closed (BBC, 2018). Subtitles are used by at least 10% of viewers and up to as many as 35% depending on the content being watched (Armstrong, 2016).

For the purposes of this project there will be only a focus on **digital** video making and therefore broadcast subtitling will be ignored. The reason being that there is no use for this project in broadcast or DVD manufacturing due to the fact the video file isn't wholly distributed to the end user.

Open Subtitles

Open subtitles are subtitles that are burnt into the video itself. Editors can use subtitles to display extra information to the user (see fig. 1). More typically they are used to show translations of a foreign language or when the actor has a difficult to understand accent. They are able to match the films tone and improve the experience of viewing a film. Other times a video may be distributed with a full subtitle script burned into the screen, this method is becoming increasingly popular with online video on social media; allowing the end-user to watch the video and have full comprehension of the subject without turning on sound. A major disadvantage, however, is that the open subtitles aren't easily editable, requiring a full re-render of the video. Another disadvantage would be that the end user has no control over whether the subtitles are on screen or not.

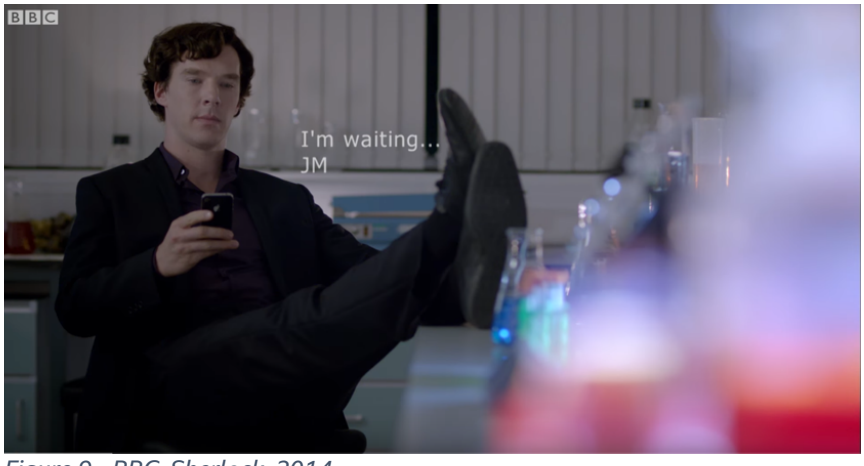


Figure 9 - BBC, Sherlock, 2014

Closed Subtitles

Closed subtitles on the other hand are separate files to the actual video. This has the advantage of being editable independently from the video, user toggle-able, and easily changed by the end user. This type of subtitling is the type most familiar to a layperson. Common use cases for closed subtitles are for use by the hard of hearing, to supplement audio in noisy environments, and for foreign language translations of the script.

Contrarily to open subtitling, there is more regulation and standardisation surrounding closed subtitling. There is a wide variety of subtitling with different focuses, ranging from broadcast standard formats with tooling to support it, to simple formats that are easy to write with a text editor. For example, the BBC use two different encodings, EBU-STL in conjunction with EBU-TT for broadcast and EBU-TD for online video (BBC, 2018). Whereas YouTube supports a huge number of subtitle formats, SubRip being the most basic but preferring EIA-608 to be used on the site (Youtube, 2018).

An example of a basic SubRip subtitle format:

SubRip (.srt) example

```
1
00:00:00,599 --> 00:00:04,160
>> ALICE: Hi, my name is Alice Miller and this is John Brown

2
00:00:04,160 --> 00:00:06,770
>> JOHN: and we're the owners of Miller Bakery.

3
00:00:06,770 --> 00:00:10,880
>> ALICE: Today we'll be teaching you how to make
our famous chocolate chip cookies!

4
00:00:10,880 --> 00:00:16,700
[intro music]

5
00:00:16,700 --> 00:00:21,480
OK, so we have all the ingredients laid out here
```

Figure 10- SubRip file example (Youtube, 2018) - https://support.google.com/youtube/answer/2734698

As the above example shows, SubRip is an easy to read format that displays the subtitling information in a simple way.

On the other hand, EBU-TT-D is more complex, including metadata about the video and subtitling information.

```
<tt:body ttm:role="caption">
        <tt:div>
                <!-- Subtitle zero - not for display -->
                <tt:p xml:id="C0" region="R1" begin="00:00:00:00" end="00:00:00:02" style="S2">
                        <tt:span>Snow White</tt:span>
                        <tt:br />
                        <tt:span>ABC D123A/02</tt:span>
                        <tt:br />
                        <tt:span>XYZ12345</tt:span>
                </tt:p>
                <!-- Begin subtitles for display -->
                <tt:p xml:id="C1" region="R2" begin="10:00:32:05" end="10:00:36:08">
                        <tt:span style="S2">This programme contains some violent<tt:br/>
                        scenes and some strong language
                        </tt:span>
                </tt:p>
                <tt:p xml:id="C2" region="R2" begin="10:02:04:00" end="10:02:06:10">
                        <tt:span ttm:agent="sp2" style="S1">Snow White, wake up!<tt:br/></tt:span>
                        <tt:span ttm:agent="sp1" style="S2">But I'm so tired!</tt:span>
                </tt:p>
                <!--
                ...
                Additional subtitles omitted
                ...
                -->
                <tt:p xml:id="C809" region="R1" begin="01:03:29:20" end="01:03:29:24">
                        <tt:span ttm:agent="sp3" style="S1">..and they all lived happily ever after.</tt:span>
                </tt:p>
        </tt:div>
</tt:body>
```

Figure 11- EBU-TT-D example file (BBC, 2018) - https://bbc.github.io/subtitle-guidelines/sample-ebutt-prepared.html

Therefore, I will start by building support for the more simple SubRip format and develop a parser for the more complex EBU-TT-D and EIA-608 format in the future.

Steganography

Steganography is the practice of concealing a file, message, image, or video within another file, message, image, or video. The technique has been used for thousands of years, for example writing in invisible ink. However, in the digital age steganography can be applied to images, binaries, and more, with the ability to hide vast swathes of information inside them without changing the human-visible appearance. Steganography is typically paired with encryption techniques.

There are many different techniques that use steganography, however for this project we will be interested in only the ones relating to images and videos.

Least significant bit (LSB) steganography is a simple steganography method for embedding information into images. LSB changes 2-4 of the least significant bits in the pixels of an image, to the human eye nothing has changed. Extraction then becomes trivial; to read the information you just read the LSB of an image.

In their conference paper *Video steganography techniques: Taxonomy, challenges, and future directions*, Ramadhan et. al., detail how steganography methods can be applied to raw video formats. Further they state that

> *"raw video steganographic techniques deal with the video as a sequence of frames with the same format."*

therefore, Lodge will be able to use a simple LSB steganography approach to embed the subtitles inside the frames of the video. The paper also details techniques that are used for compressed video formats such as MPEG-4, listing 5 methods that have successfully been used to add steganographic data. (Ramadhan J. Mstafa, 2017). These methods involve extensively work changing the encoding algorithm of the lossy compression.

Other techniques are used to hide information more securely, and defends against steg-analysis. However for this project, security of this kind is not needed as we need the information to be as easily extractable as possible.

Market research

Users

There are two groups of users that Lodge is aimed at; producers, and consumers. Producers will be technically competent and will have used editing software before. They will use the Lodge encoding software, to generate, edit, and embed subtitles inside video files.

Consumers will be the layperson that uses subtitling for enhancing their experience of the video content, they will use primarily the Lodge decoder software. The end user experience for them should have the least resistance and be invisible to the user. This will mean that the software should be bundled inside a video player. As building a video player from the ground up is not in scope for this project, alternatives methods to bundle this encoder will have to be found.

Producers include:

- Stenographers
- Video Editors

Consumers include:

- Deaf or hard-of-hearing people
- Foreign language speakers
- Layperson

Competitors

As mentioned before YouTube is one of the largest competitors for Lodge in the market, especially for digital video, as according to Alexa they are the second most trafficked site in the world (Alexa, 2018).

YouTube provides captioning software for their online videos in their Creators Studio, shown in **Appendix A** . This software provides a number of

different ways to add captions to a video; one can either upload a file, transcribe with auto-timings, or transcribe and set timings. The Creators Studio has a very clear interface, which is easy to use and understand by the layperson.

Netflix, another huge online video platform, lists 8 pieces of software that it recommends people subtitling for Netflix videos (Netflix, 2018). Each of these pieces of software do a similar

However, from my research I have not been able to find a software product that incorporates both subtitling and steganography. This separates Lodge from the competition, as it will provide a combined file for both subtitles and video.

Project Planning

Methodology

During this project I will use a number of project planning techniques, focused around Agile software development. I have decided to forego strict agile methods and adapt some from different bits.

For planning I will use feature epics, which allow me to categorise features that will be delivered by my project. With epics I can define a wide story for the product and whittle down to individual user stories. From these stories I get a concise piece of work, that is actionable and completable. Completed stories lead to a completed epic, and with that a completed feature of the product.

Timeboxing will be my time keeping strategy. This gives a hard deadline for a task that must be kept to, whether it is fully completed or not. For this reason, I have allotted myself a clean-up/catch-up session towards the end of the project. I believe this will be the most effective strategy to complete the most features possible for this project.

Requirements

Below is a list of requirements formulated throughout my research. They are categorised with an adapted MoSCoW method (Must have, Should have, Could have, Wish for).

PRIORITY	REQUIREMENT
MUST	Both programs will support the same project specific subtitling format.
MUST	Lodge must work with one uncompressed video format.
MUST	The encoder must not change the file extension and the original video must be playable, or make it corrupt in any other way.
MUST	The video must have no visible change when subtitles are merged in to the video file.
MUST	The decoder must work in “real-time” and must not interfere with video playback.
MUST	The decoder must display subtitles in an easy to read format. The decoder must be able to be turned off.
MUST	The encoder must have a GUI for ease of use.
MUST	The encoder must be able to specify the time where the subtitle is to be displayed from.
SHOULD	Lodge should support 2 or more subtitle formats
SHOULD	Lodge should support one compressed video format, such as MP4.

SHOULD	The subtitles should be readable by most people
COULD	Lodge could support different fonts for stylistic purpose
COULD	Encoder GUI support real-time transcription
COULD	Lodge could be compiled to run on multiple different operating systems (Windows, MacOS, Linux)
WISH	Lodge supporting multiple languages
WISH	Adding non-subtitle features, such as hyperlinks
WISH	Web based decoder
WISH	Lodge work with every popular video format

Epics
Following from the requirements listed above, epics can now be defined. Each epic being a sizeable chunk of work that can be split down into tasks or specific user stories.

As with all projects these epics are subject to change as new requirements/limitations pop up. Some epics are not fully fleshed out and will be reviewed before starting the tasks in them.

Minimum Viable Product

As a video editor, the Lodge encoder should be able to merge a basic SubRip subtitle files with my video file.

As a video watcher, the Lodge decoder should be able to recognise merge video files and display the subtitles on screen.

Deliverables:

- Basic subtitle encoder
- Basic subtitle decoder
- Lodge subtitle format

The goal of this epic is to quickly build a basic product that **is** usable by end users and fulfils most of the MUST requirements. However, requirements that aren't strictly necessary for the first iteration will be left to a later epic.

The initial video format support by Lodge will be an uncompressed video.

Task - Lodge subtitle format
Deliverables: A data format for subtitles in Lodge
Details:
To keep the decoder as simple as possible and not be muddy up with multiple parsers there must be a Lodge specific subtitle format.

As this epic is about delivering an MVP the format will be at its most simple. Only detailing the content of the subtitle and timing information.

Story – Merging video and subtitles

As a video editor, I want to merge a basic SubRip subtitle file with raw video.

Deliverable: Basic subtitle encoder
Details:
This subtitle file (SubRip) is very simple and will provide the simplest implementation for subtitle transformation for the encoder.

By using a raw video format, such as AVI, Lodge does not have to support any weirdness with compression (yet).
Sub-tasks:

- File transformation, SubRip to Lodge format
- Subtitle embedding, using steganography

Story – Displaying subtitles

As a viewer I want the subtitles displayed over the video in real time.

Deliverable: Basic subtitle decoder
Details:
Exploration in to the implementation will be needed. It may be possible to write a VLC plugin.

The subtitles should be easily visible when displayed, and non-configurable at this stage of the product as that adds unnecessary complexity. Black background with a white text will be fine for the first iteration.

GUI

As a subtitle editor I would like to use the Lodge encoder to view and edit subtitles in a GUI.

Deliverables: A GUI on top of the encoder

To make this product appealable to the non-technical end user, a user interface must be created on top of the encoder. This epic covers the introduction of a GUI and gives a breakdown of each component that will be built.

Story – Basic GUI

As a user that is not familiar with a CLI, I should be able to use a basic GUI version of the Lodge encoder.

Deliverables: A basic GUI that wraps the Lodge encoder.
Details: A basic GUI wrapper around the Encoder CLI application, taking in a subtitle file and video file and spitting out the combined one.

Story – Transcription GUI

When using the Lodge encoder I should be able to edit and view my subtitles

Deliverables: A text editor that supports Lodge style subtitles
Details:
This story involves just adding a basic text window that shows a subtitle file with the timestamps in order for a layperson to be able to edit the subtitles easily.

It should handle formatting to the Lodge format for the user of the software.

Story – Video GUI

As a subtitle editor I would like to directly create subtitles while watching the video.

Deliverables: A video player interface with subtitling capabilities
Details:
Taking inspiration from the various subtitle editing software already available, the lodge encoder should make the video playable alongside the transcription interface.

Subtitles

Lodge should support more subtitle formats

This epic will be focused on improving the subtitle experience for both producers and consumers of them.

Story – EIA-608

Lodge should support EIA-608

EIA-608 is the standard for subtitling online video in the United States. YouTube, the largest online video platform, uses this as their preferred format.

Deliverables: Updated the encoder to deal with EIA-608 files
Detail: Requires adding a data transformation from EIA-608 format to Lodge subtitle format

Story - EBU-TT-D

Lodge should support EBU-TD files

EBU-TT-D is a standard of subtitling online video in the EU. The BBC use this for all their online videos.

Deliverables: Updated encoder that parses EBU-TT-D files
Detail: Requires adding a data transformation from EBU-TT-D format to Lodge subtitle format.

Story – Accessible subtitles

As a video-watcher I would like the subtitles to be accessible to all

Deliverables: Accessible subtitles
Detail: Accessibility surrounding subtitles has been detailed at length in the subtitling guidelines from the BBC (BBC, 2018).

Video format

Lodge should support more video formats

To be a truly universal tool Lodge should support a wide variety of video formats. Each will bring its own complexity and will require extension of the decoder.

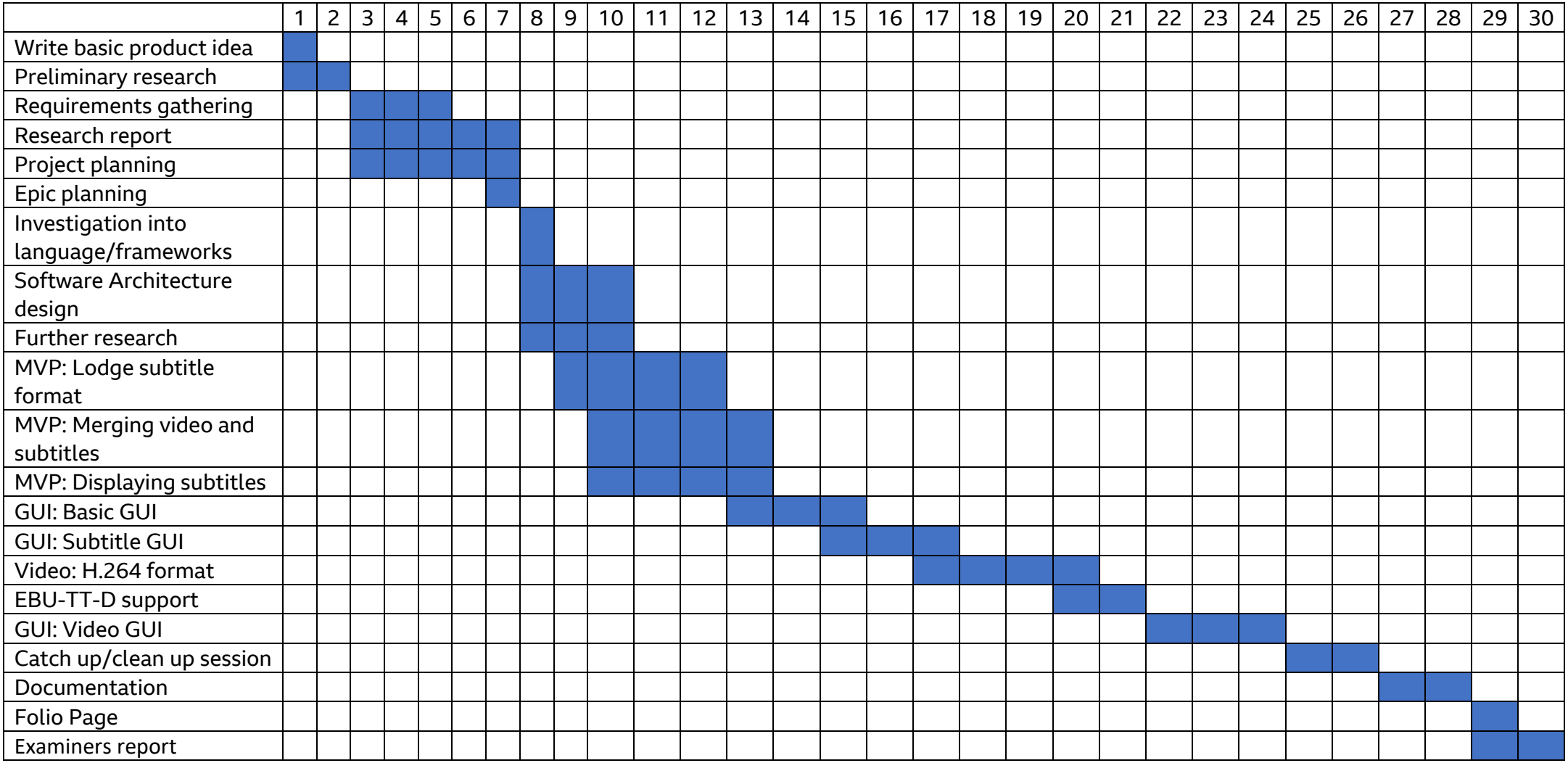
Story – H.264 support

As a consumer of video content, I want Lodge to be able compatible with H.264 compressed videos.

Deliverables: Updated encoder/decoder that will understand MP4 videos
Details:
H.264 video is used universally across the video industry. Lodge should develop support for this codec.

It is a lossy compression format and therefore will be difficult to implement. This story will take a substantial amount of time to

Gantt Chart



In the above Gantt chart, the columns are individual weeks starting from 26/09/2018 and ending on 25/04/2019. The timing for this chart is modest giving flexibility to over running features of the product. Each story/task block (MVP, GUI, etc.) includes a design stage.

Product

Implementation details

This project will be open-source as I see no advantage to being proprietary. An open-source license hasn't been chosen just as yet, but it will most likely be an MIT or a GPL v3 compatible license. All code and documentation will be made publicly available on GitHub or similar website.

As for the programming language, I have yet to make a choice. I have most experience with JVM based languages, so this may seem the most natural choice. JavaScript/TypeScript also has massive advantages in that it runs cross platform (thanks to browsers) and can be built to run natively on machines with frameworks like Electron. However, the project should be written in the same language across both products to mitigate the potential confusion of multiple languages.

Issues

An issue faced by the Lodge product is that the decoder needs to be ubiquitous, every major video player will need an implementation of the decoder. Otherwise there will be a poor end-user experience, with users expecting a subtitle to be shown and the player not recognising or showing the subtitles.

To combat this issue Lodge will provide an open source reference decoder implementation. In addition, there will a formal format specification for the subtitling with Lodge. The specification will allow people outside of this project to write their own decoder implementation.

A similar issue is that the project, to gain any wide-spread use, it will need to support every major video format. Unfortunately, each video format will require its own internal decoder. This grows the workload for this project significantly, therefore it will be limited to a select few formats. The same happens with subtitle formatting.

Another issue that will need to be tackled by the project is using compressed video formats. Compressed video formats use lossy codecs to remove information from the images displayed. For this reason, the Lodge encoder will need to be used after a video has been compressed and any further compression may corrupt the subtitles.

Annotated Bibliography

BBC, 2018. *Subtitle Guidelines.* [Online]
Available at: https://bbc.github.io/subtitle-guidelines/
[Accessed 25 October 2018]

The BBC shares its knowledge about subtitling in this all-encompassing guide. Giving a detailed overview of formats used for broadcasting and online video, and an expansive list of guidelines of how subtitles should be displayed on the screen. As this is the strategy used by one of the world's largest broadcasters it will be an invaluable source of information for the project and will be constantly used to reference, especially when dealing with displaying subtitles.

OpenStego, 2018. *Open Stego?* [Online]
Available at: https://github.com/syvaidya/openstego
[Accessed 27 October 2018].

This web-page holds the source code for an open source implementation for image steganography, written in Java. As I have a good amount of experience with Java, I was able to read the source code as if it were documentation. OpenStego uses various techniques to hide information, such as Least Significant Bit and DCT-LSB. I found this source to be valuable in allowing me to examine how an actual steganography program would work. Giving me insight in how a robust program may be structured. However, OpenStego only deals with images and not video steganography so its usefulness is limited.

Ramadhan J. Mstafa, K. M. E. E. A., 2017.
Video steganography techniques: Taxonomy, challenges, and future directions.
Farmingdale, NY, USA , IEEE

Presented at the IEEE LISAT conference in 2017, this paper covers the techniques used for video steganography. Methods for both compressed and uncompressed videos are shown, with itself referencing the papers detailing the implementation. This is a vital source for my project, giving a wealth of knowledge on the video steg. From this paper I was able to prioritise my strategy for implementing steganography, choosing to do tackle uncompressed video first. It will be invaluable resource as I move forward with this project. As this paper was presented at an IEEE

sponsored conference we can fairly sure of its accuracy and reliability.

Youtube, 2018. *Subtitle Support.* [Online]
Available at: https://support.google.com/youtube/answer/2734698?hl=en-GB
[Accessed 7 November 2018].

In this web-page YouTube details the formatting supported by their video platform. From the simple SubRip format to full broadcast-grade subtitling formats such as EBU-TT. As the world's largest video platform, their support of a product/format is a good indication of whether it is useful/used by a wide audience. I was able to gain a good knowledge base of the subtitle formatting standards and take decisions on what to use for this project.

YouTube, 2018. *Captioning.* [Online, Software]
Available at: https://www.youtube.com/timedtext_video
[Accessed 10 November 2018].

Another source from YouTube, this time some in-browser subtitling software. The features this software boasts are the full support of 15+ subtitling formats, a live transcription service, and an auto-syncing transcription service. This was my first interaction with any subtitling software and will be used as my reference while developing my own project. With software, as well as OpenStego, I was able to start to piece together how this project might evolve. This transcription software will be used as the golden standard to shoot for in terms of subtitling utility.

Works Cited

Alexa, 2018. *Youtube.* [Online]
Available at: https://www.alexa.com/siteinfo/youtube.com
[Accessed 5 Novemeber 2018].
Armstrong, M., 2016. *Automatic recovery and verification of subtitles for large collections of video clips,* London: BBC.
Bala, S., 2016. *H.264 is magic: a techinical walkthrough.* [Online]
Available at: https://sidbala.com/h-264-is-magic/
BBC, 2018. *How do I create subtitles?.* [Online]
Available at: https://www.bbc.co.uk/guides/zmgnng8
[Accessed 10 October 2018].
BBC, 2018. *Subtitle Guidelines.* [Online]
Available at: https://bbc.github.io/subtitle-guidelines/
[Accessed 25 October 2018].
Changyong Xu, X. P. T. Z., 2006. *Steganography in Compressed Video Stream.* s.l., IEEE.
encoding.com, 2019. *Global Media Format Report 2019.* [Online]
Available at: https://1yy04i3k9fyt3vqjsf2mv610yvm-wpengine.netdna-ssl.com/files/2019-Global-Media-Formats-Report.pdf
[Accessed 03 2019].
FFmpeg, 2019. *FFmpeg.* [Online]
Available at: https://ffmpeg.org
[Accessed 10 04 2019].
FFmpeg, 2019. *FFmpeg: doc/examples/transcoding.c Source File,* s.l.: s.n.
Gent, K. C. a. P., n.d. *dct.* [Online]
Available at: https://www.math.cuhk.edu.hk/~lmlui/dct.pdf
ITU, 2019. *H.264 : Advanced video coding for generic audiovisual services.* [Online]
Available at: https://www.itu.int/rec/T-REC-H.264
Liberis, E., 2016. *"GitHub - eliberis/movest: Video Steganography using motion vectors. Part II project, part of the University of Cambridge Computer Science course..* [Online]
Available at: https://github.com/eliberis/movest
Moreira, L., 2019. *FFmpeg Libav Tutorial.* [Online]
Available at: https://github.com/leandromoreira/ffmpeg-libav-tutorial
Netflix, 2018. *Technology Resources.* [Online]
Available at: https://partnerhelp.netflixstudios.com/hc/en-us/categories/202282017-TECHNOLOGY-RESOURCES
[Accessed 16 November 2018].
Open Hub, 2019. *The FFmpeg Open Source Project on Open Hub.* [Online]
Available at: https://www.openhub.net/p/ffmpeg

OpenStego, 2018. *Open Stego.* [Online]
Available at: https://github.com/syvaidya/openstego
[Accessed 25 10 2018].
Ramadhan J. Mstafa, K. M. E. E. A., 2017. *Video steganography techniques: Taxonomy, challenges, and future directions.* Farmingdale, NY, USA , IEEE.
Ramdhan J. Mstafa, K. M. E. E. A., 2017. *Video steganography techniques; Taxonomy, challengs, and future directions.* Farmingdale, NY, USA, s.n.
Riff, D., 2018. *Re-encoding persistent video steganography.* s.l.:s.n.
Wikipedia, 2019. *Context-adaptive binary arithmetic coding - Wikipedia.* [Online]
Available at: https://en.wikipedia.org/wiki/Context-adaptive_binary_arithmetic_coding
Wikipedia, 2019. *Context-adaptive variable-length coding - Wikipedia.* [Online]
Available at: https://en.wikipedia.org/wiki/Context-adaptive_variable-length_coding
Youtube, 2018. *Captioning.* [Online]
Available at: https://studio.youtube.com
YouTube, 2018. *Captioning.* [Online]
Available at: https://studio.youtube.com/
[Accessed 10 November 2018].
Youtube, 2018. *Subtitle Support.* [Online]
Available at: https://support.google.com/youtube/answer/2734698?hl=en-GB

Appendix A

All images listed below were accessed from YouTube Creators Studio. (YouTube, 2018)

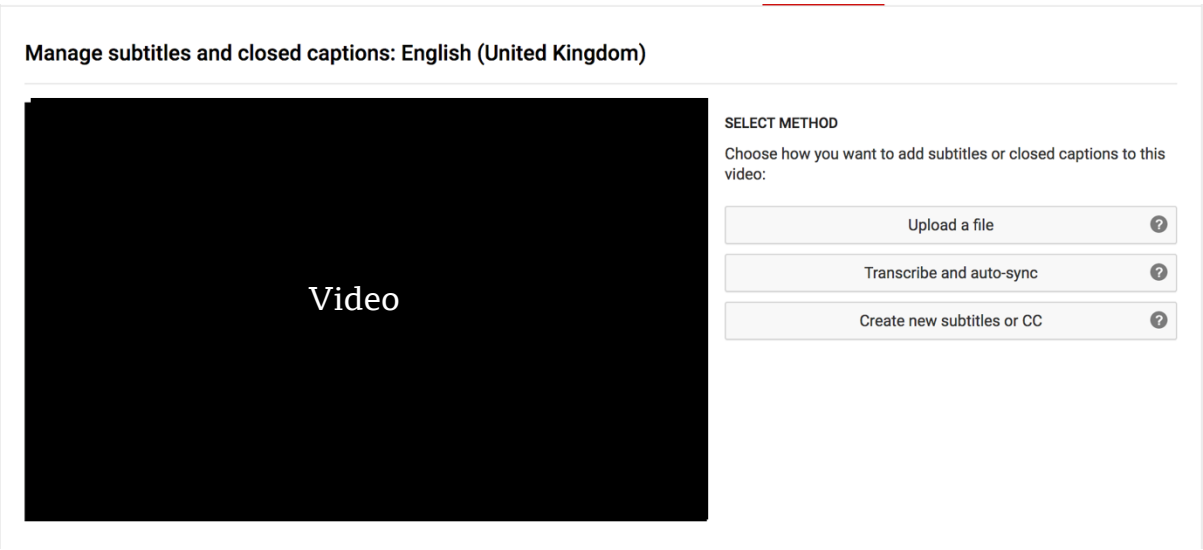


Figure 12-Subtitling welcome screen

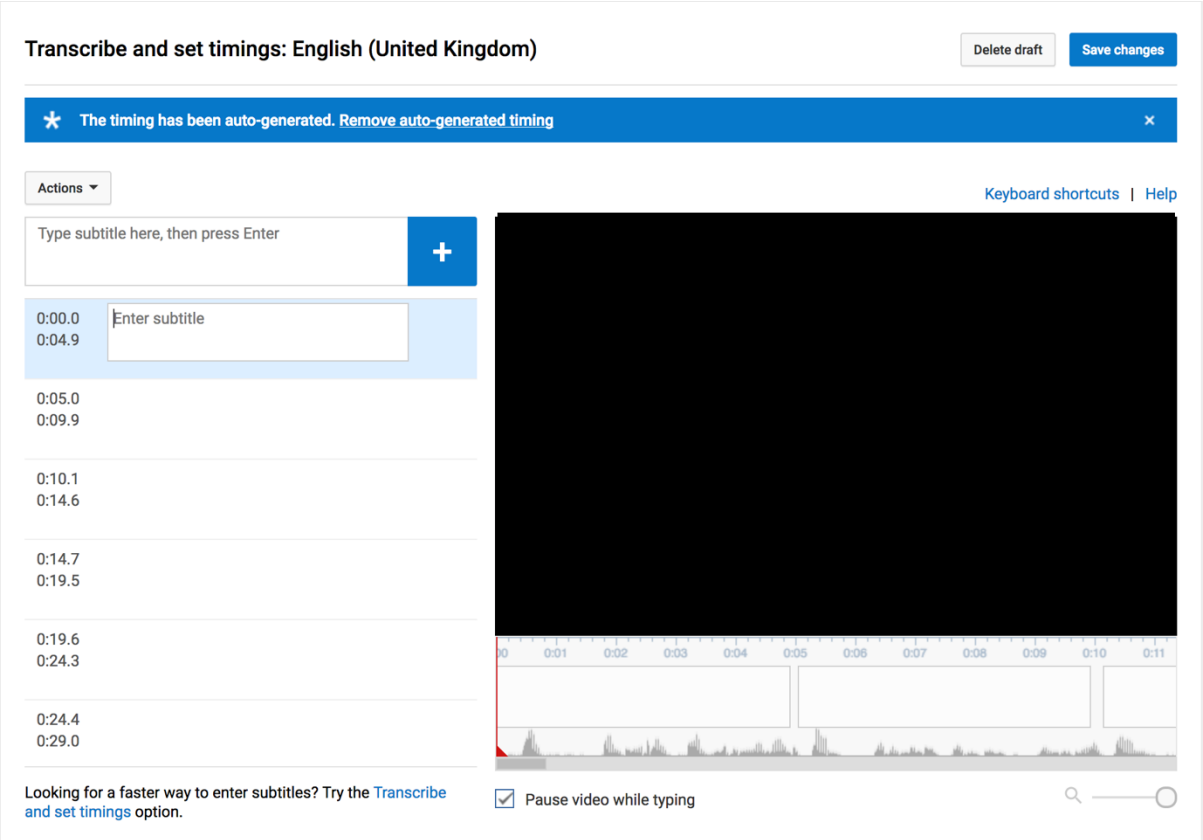
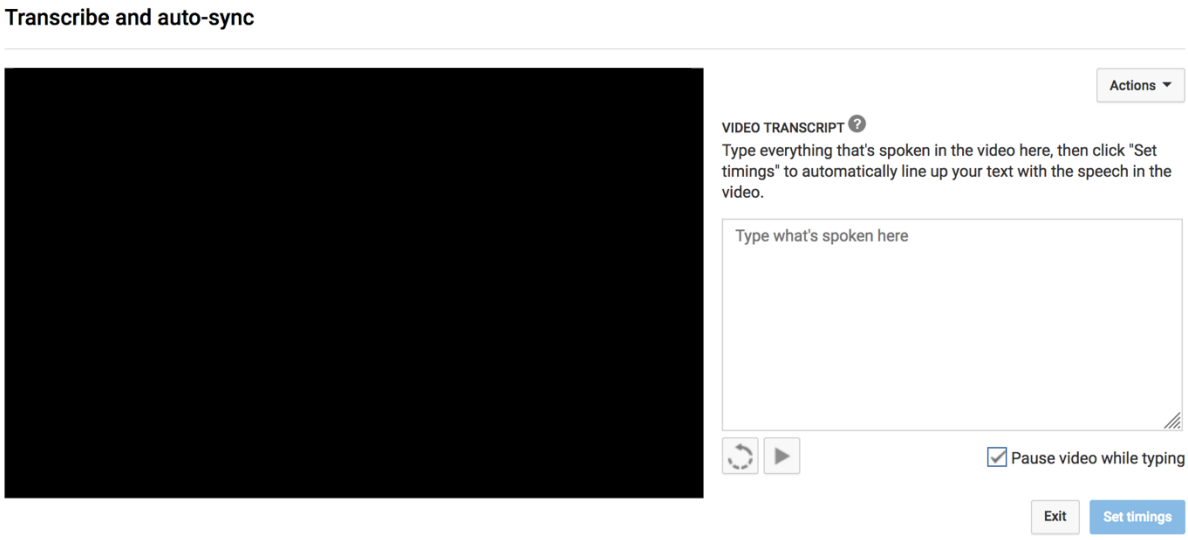


Figure 13 - Transcription interface


Transcribe and auto-sync
Actions
VIDEO TRANSCRIPT
Type everything that's spoken in the video here, then click "Set timings" to automatically line up your text with the speech in the video.
Type what's spoken here
Pause video while typing
Exit
Set timings