CS2108 Project

By: Louiz Kim-Chan (A0168113L), Anu Kriti Wadhwa (A0170594Y), Tiffany Chong (A0171997J)

Contents

Background

Steganography is the art of hiding secret messages within another file. Our task was to implement a system to encode a secret message (a snippet of text) inside a 36-second music clip, `edelweiss_36seconds.mp4`. The goal is to play this clip such that another device can record and recover the secret message.

Instructions

Mode

On line 10 of `Encoder.m` and line 8 of `Decoder.m`, choose to set `highFreqMode` to 1 or 0. Set to 1 if your equipment is high-end (e.g. Macbook Pro 2017 and above, external microphone etc.) and you are in a quiet room, or if you intend to send the `.mp4` file directly (no recording). This is very stealthy and generally undetectable, but not very well supported by generic hardware. Set to 0 otherwise, which has a slightly higher chance of detection by hearing. The only difference is the exact frequencies being encoded. The readme is written w.r.t. **highFreqMode** frequencies (low freq mode in brackets as LFM).

Encoding

1. Place `edelweiss_36seconds.mp4` and the `.txt` file containing the encoding message into the directory of the `Encoder.m`.
2. Run `Encoder.m` on Matlab.
3. Encoder will prompt the user for the filename of the text file. Input the filename (including `.txt`) and press enter.
4. An `.mp4` file with the message encoded, called `encoded_audio.mp4`, will be generated in the directory.

Decoding

1. Under the input mode (line 5, `readMp4`) of `Decoder.m`, specify whether you want to decode a `.mp4` file directly, or record audio using the computer's built in microphone.
2. Mp4 decoding instructions:
    a. Ensure `encoded_audio.mp4` is in the directory of `Decoder.m`.
    b. Run `Decoder.m` on Matlab.
    c. The secret message will be printed.
3. Recording decoding instructions:
    a. Run `Decoder.m` on Matlab.
    b. Start playing the encoded `.mp4` file (around 4 seconds after the decoder has started running, to prevent any truncation of the message).
        i. Note: You can start recording while the `.mp4` file is playing as well, but you will receive a truncated message.
    c. The spectrogram will appear in a figure, and the secret message will be printed.

Testing/Tuning

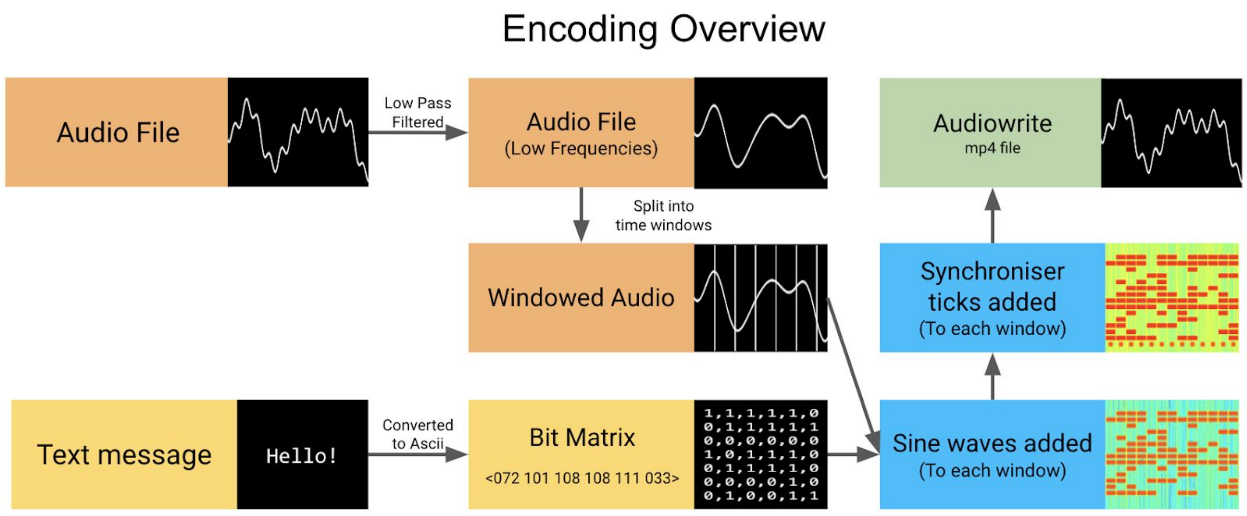
- To verify accuracy of the program regardless of hardware, you can read audio directly from `encoded_audio.mp4`. In `Decoder.m`, assign `readMp4` (line 5) a value of 1.
- If messages cannot be detected, you can try to adjust the variables in `Encoder.m`: `signalScale` (line 12) upward until it can be detected, and/or `syncTickScale` (line 14) downward. Observe the spectrogram produced by `Decoder.m` to guide your tuning.

Implementation

Two main MATLAB files are used in this project.

- **`Encoder.m`** - encodes the secret message in binary in the form of on-off signals in the higher frequency bands, and saves encoded audio into file
- **`Decoder.m`** - records audio, reads data from spectrogram, recovers the binary message, and converts it back to text.

Encoder



Low-Pass Filter

The encoder filters off frequencies higher than 17000 Hz (LFM: 14000 Hz) in order to leave high-frequency bands empty for transmission of the secret message. This is already in the upper part of the average audible range, and audio quality changes are hardly noticeable.

Conversion of Text to Binary Representation

Conversion of the first 128 ascii characters is supported. The encoder uses a 7-bit binary format to encode the secret message. This is done by using Matlab’s built-in 8-bit ascii-to-binary converter and then discarding the 1st bit.

Time Windows

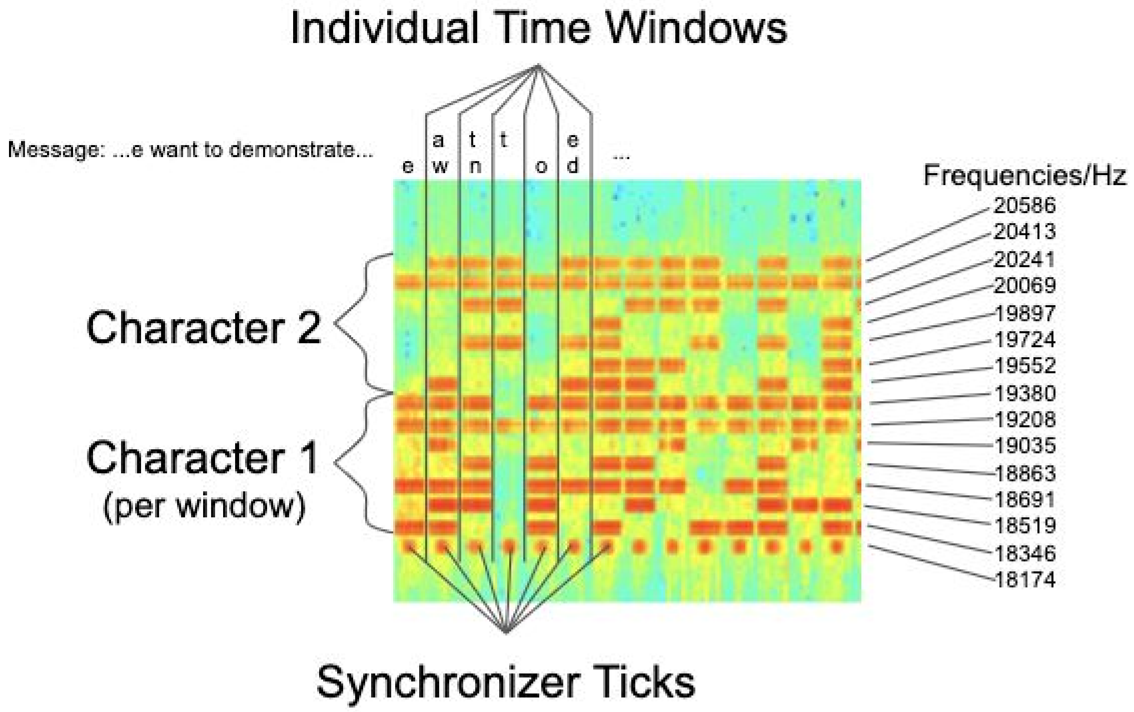


Figure - Sine waves and synchroniser ticks per time window.

Every 0.4s time window has up to 15 extra sin waves encoded in the time domain. Up to 14 sin waves are added for signal information, and 1 sin wave is added for the synchronizing. All sin waves encoded have a window applied, to prevent popping sounds due to truncated waveforms. Synchronizer ticks use a Hamming window, while the signal waves use a Tukey window.

Addition of sine waves
Two characters (14 bits) are transmitted every 0.4s, to enable enough bandwidth to transmit 180 ascii characters in 36 seconds. Each bit in the 14-bit binary signal is transmitted using one frequency channel. For each of the 14 frequency channels we are using, if a bit is **on** in the character being encoded, the corresponding sin wave will be added.

E.g. if the letters being encoded in this window are HI, where H: ascii-72 = 1001000b, I: ascii-73 = 1001001b, we will include the following sin waves based on:

H			**I**		
Frequency/Hz	Bits	Inclusion	Frequency/Hz	Bits	Inclusion
18346	1	**Included**	19552	1	**Included**

18519	0	Excluded	19724	0	Excluded
18691	0	Excluded	19897	0	Excluded
18863	1	**Included**	20069	1	**Included**
19035	0	Excluded	20241	0	Excluded
19208	0	Excluded	20413	0	Excluded
19380	0	Excluded	20586	1	**Included**

Synchroniser Ticks
A constant ticking signal is transmitted using the 18174 Hz (LFM: 16150 Hz) channel. For every 0.4s time window, a 18174 Hz sine wave is added to the middle 0.2s (see figure above). Using these ticks, the decoder can reliably locate time windows and check whether or not the other sine waves are present in that time window.

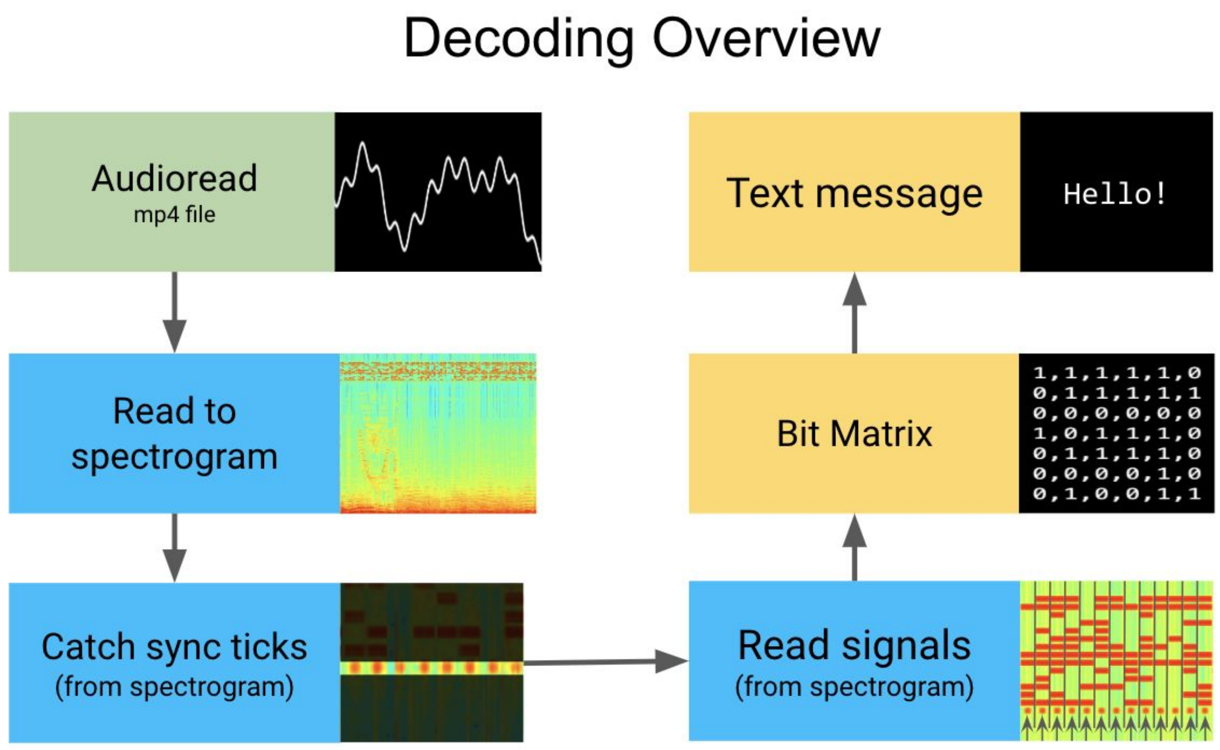
Placing ticks in the middle improves the detection of the signal compared to detection at the edges, as false positives can occur due to bleeding from other time windows (since spectrogram is taken with overlapping windows) or false negatives from dimming due to usage of windows to filter the added sine waves.

Audiowrite
The resultant 36-second signal is encoded into an mp4 file with 44.1 kHz sampling frequency.

Explanation of Variables

1. **Damping**: As high-amplitude sine waves are added into the signal, automatic audio-clipping may occur, introducing noise to the music file. A damping factor is used to scale down all added frequencies. This is based on the maximum amplitude, A_{max},of the input audio file, and the maximum number of frequencies played, n_{freqs}. We conservatively damp all bit signals by $(1 - A_{max}) / n_{freqs}$ to prevent the amplitude from exceeding 1. Note that the synchronizer ticks are scaled with a different factor from the signal waves, due to their shorter duration, to calculate an effective threshold (see Decoder section).
2. **Window functions**: For encoding all signal frequencies, windows were used. For the synchronizing ticks, we used Hamming windows, which are relatively gentle. For the bit signals, we used Tukey windows with cosine fraction 0.3, which is “steeper” than the Hamming window. This gives us relatively smaller gaps between each frequency, while still leaving a gap, minimizing bleeding.

Decoder



Audioread
Decoder listens to audio for 40s (at least 36s) with 44.1 kHz sampling frequency.

Spectrogram
Matlab's built-in STFT function is used to read data into spectrogram.

Catching Synchroniser Ticks
The program reads the row in the spectrogram which contains the synchronizer ticks (row 423 or LFM: 376), and finds the mean of absolute values of the STFT readings, taken as the *threshold*. It then loops through from the leftmost column, rightward, until a value greater than the mean is found. From this first index, the sine waves will then be read at all indices at 0.4s intervals after the detected ticker.

Note that this is not entirely foolproof, as it can be falsely triggered by loud noises prior to the time the actual signal starts playing, hence the music should start playing as close to the audio signal's start as possible.

Sine wave detection
From the spectrogram, we gather data from 14 different frequencies, which are specified under the section "Frequency Channels" below. From these rows, we gather data at the spectrogram indices specified by the previous step's synchronizer tick detection. If the value at the bit's row is

greater than *threshold*, the bit is read as 1, otherwise it is 0. This produces a matrix of bits, producing two 7-bit characters per index.

Conversion to ascii
The decoder converts 7-bit binary matrix into string format by appending 0 to the start of the 7-bit binary and then using Matlab's built-in 8-bit binary-to-ascii converter.

Frequency Channels

The following frequency channels and indices are used to encode and decode.

Every odd-indexed character:

Frequency (Hz)	18346 (16322)	18519 (16494)	18691 (16667)	18863 (16839)	19035 (17011)	19208 (17183)	19380 (17356)
Spectrogram Index	427 (380)	431 (384)	435 (388)	439 (392)	443 (396)	447 (400)	451 (404)

Every even-indexed character:

Frequency (Hz)	19552 (17528)	19724 (17700)	19897 (17873)	20069 (18045)	20241 (18217)	20413 (18389)	20586 (18562)
Spectrogram Index	455 (408)	459 (412)	463 (416)	467 (420)	471 (424)	475 (428)	479 (432)

Test results

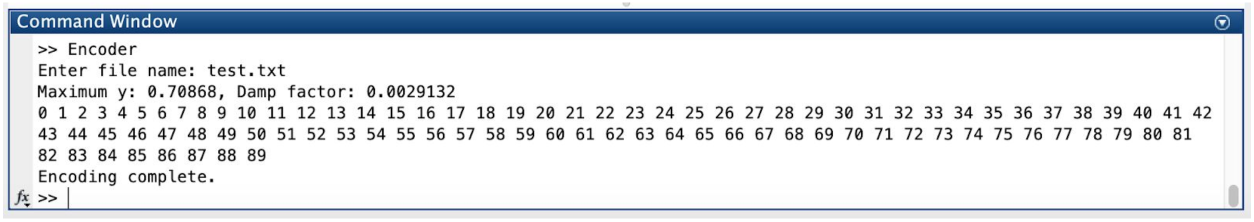
Transmission/Low Frequency Mode Tests

Note that these tests were done in low frequency mode, for transmission testing across “non-high-end” laptops.

Encoding

The following message was encoded:

Today is a bright sunny day. It is the D day. Everyone is excited. We want to demonstrate our course project in our CS2108 Introduction to Media Computing.



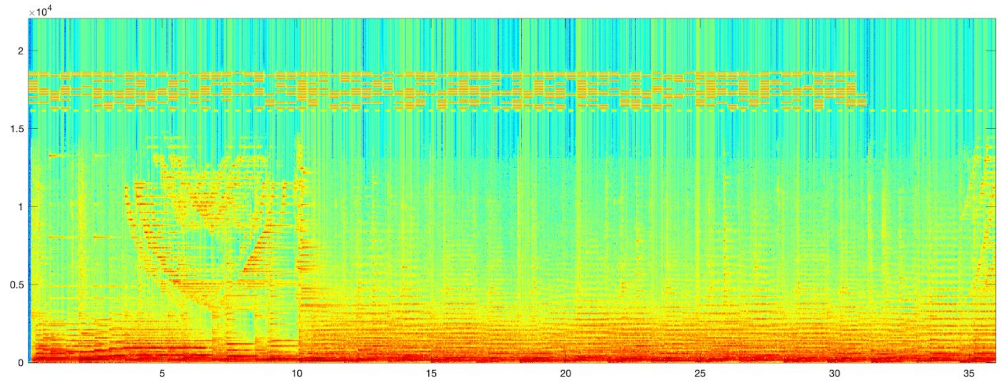
```
>> Encoder
Enter file name: test.txt
Maximum y: 0.70868, Damp factor: 0.0029132
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81
82 83 84 85 86 87 88 89
Encoding complete.
>>
```

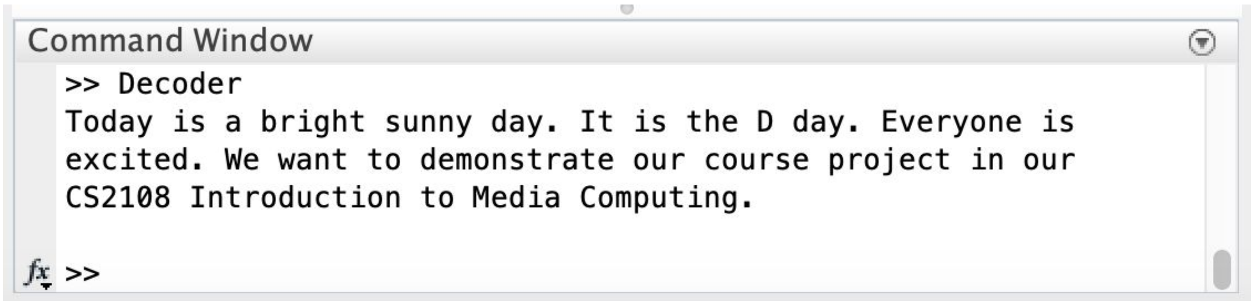
`signalScale` was 0.15, `syncTickScale` was 0.3.

Decoding

Mp4 Decode

The results for reading the mp4 file:



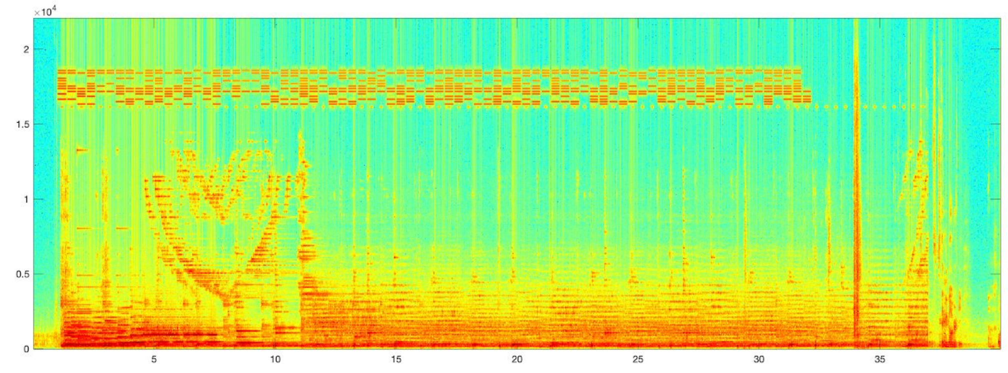


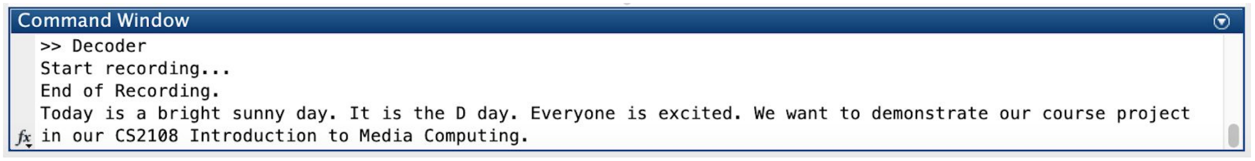
```
>> Decoder
Today is a bright sunny day. It is the D day. Everyone is
excited. We want to demonstrate our course project in our
CS2108 Introduction to Media Computing.

>>
```

Same Laptop Recording Decode

The results for recording audio from the same laptop that plays it:

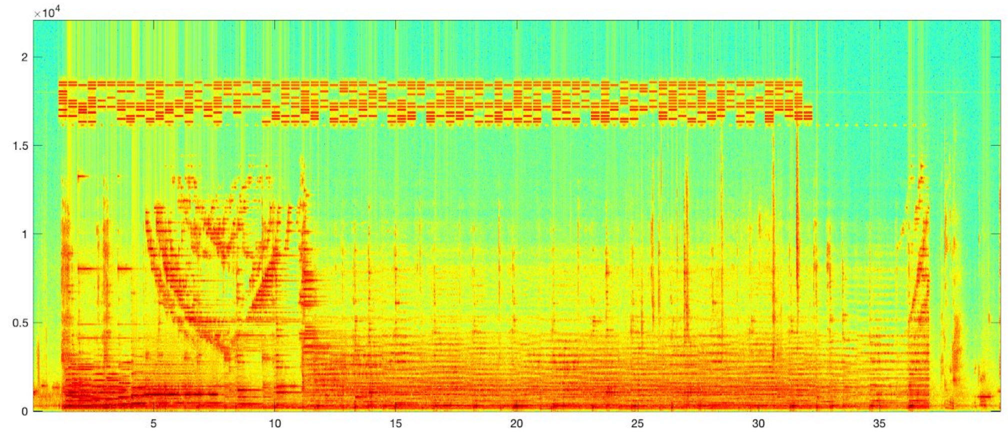


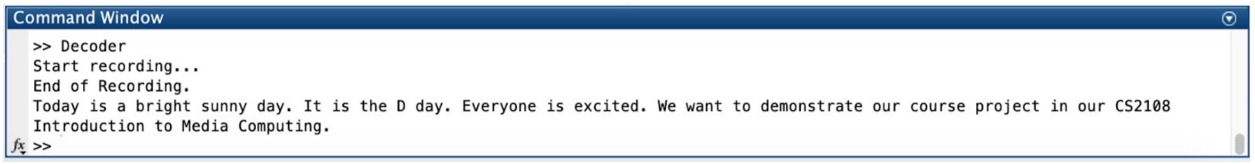

Command Window

```
>> Decoder
Start recording...
End of Recording.
Today is a bright sunny day. It is the D day. Everyone is excited. We want to demonstrate our course project
in our CS2108 Introduction to Media Computing.
```

<u>Different Laptop Recording Decode</u>

The results for recording audio from a different laptop that plays it:




Command Window

```
>> Decoder
Start recording...
End of Recording.
Today is a bright sunny day. It is the D day. Everyone is excited. We want to demonstrate our course project in our CS2108
Introduction to Media Computing.
>>
```

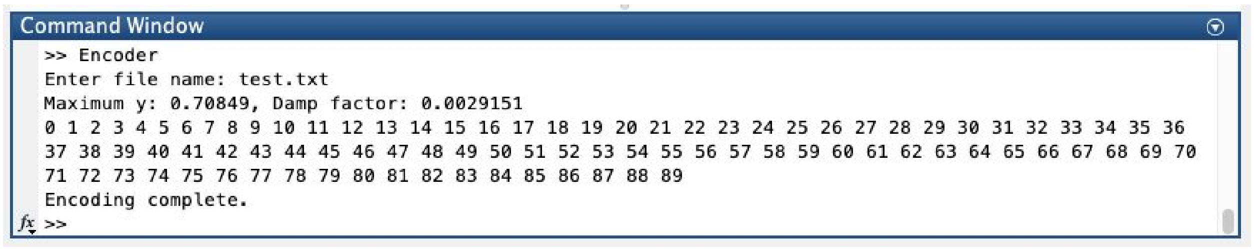
Disclaimer: For air transmission, there is not a 100% success rate, this required more than one attempt. Volume must be adjusted so that no audio clipping occurs while still having a fully detectable signal, power source to the laptop also affects audio quality (consistency across frequency bands).

MP4 File/High Frequency Mode Tests

Encoding

The following message was encoded:

Today is a bright sunny day. It is the D day. Everyone is excited. We want to demonstrate our course project in our CS2108 Introduction to Media Computing.



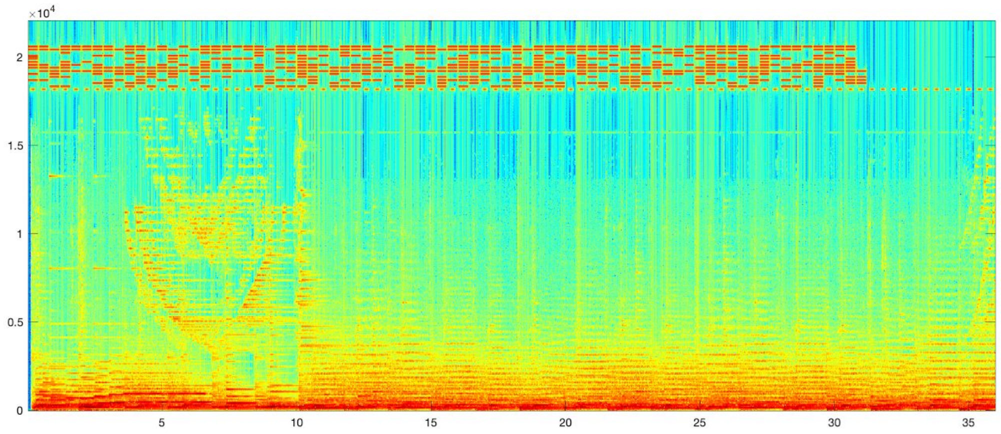
```
>> Encoder
Enter file name: test.txt
Maximum y: 0.70849, Damp factor: 0.0029151
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70
71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89
Encoding complete.
>>
```

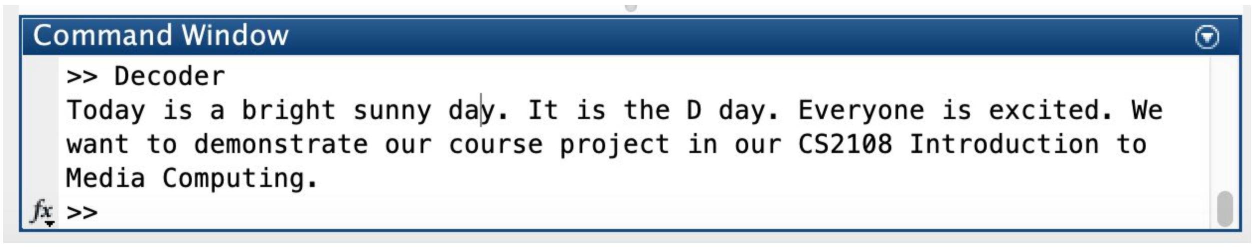
`signalScale` was 1, `syncTickScale` was 0.4.

Decoding

Mp4 Decode

The results for reading the mp4 file:

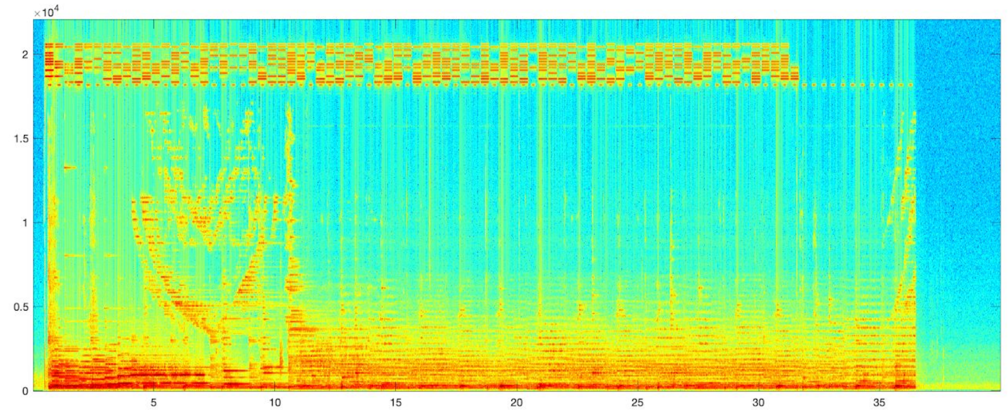


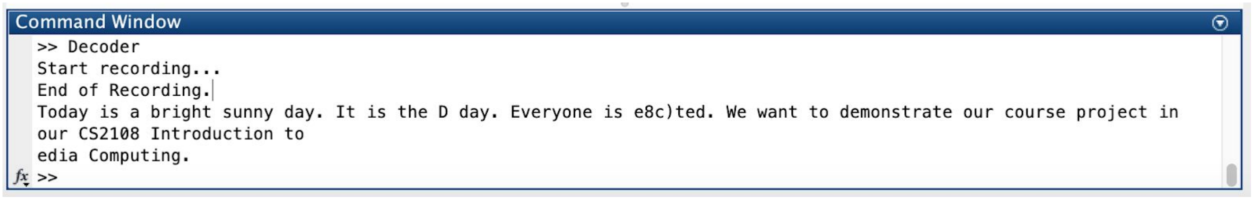


```
>> Decoder
Today is a bright sunny day. It is the D day. Everyone is excited. We
want to demonstrate our course project in our CS2108 Introduction to
Media Computing.
>>
```

<u>Same Laptop Recording Decode</u>

The results for recording audio from the same laptop that plays it:





```
>> Decoder
Start recording...
End of Recording.
Today is a bright sunny day. It is the D day. Everyone is e8c)ted. We want to demonstrate our course project in
our CS2108 Introduction to
edia Computing.
>>
```

3/155 = 1.94% data corruption, which is not perfect, but for the stealth and bandwidth, is fairly reasonable.