Dinu Alexandru
342C4

Signal Processing Project

Steganography

For this project, two main methods of concealing a secret in a carrier data were implemented: image-in-audio and text-in-image. Image-in-audio is implemented using a basic LSB encoding and it is used to demonstrate the ease of embedding a grayscale image into an audio file, without any perceptible differences when replaying the altered audio track. Text-in-image is implemented using a slightly modified algorithm presented in a paper [1]. This method uses the 2D Discrete Wavelet Transform (DWT) on the Cb channel of a YCbCr image to delimit regions of detail and then embeds the text into those regions that are imperceptible to the human eye.

Image-in-audio

The current implementation of this embedding technique hides a grayscale PNG image into a WAV audio file using LSB encoding. The PNG image consists of height x width pixels, each of uint8 size. The WAV file data is read as uint16 bytes.

The embedding scheme encodes each byte B_i of the image into 8 bytes $B_a(1:8)$ of the audio. Each bit b_j of B_i is stored as the LSB of each corresponding byte of the audio, that is $b_j \rightarrow LSB(B_a(j))$. The size of the image is also encoded, as the first 4 bytes (2 x 2 bytes for height and width) of the WAV data.

This scheme requires that the length of the WAV data to be large enough so that each byte of the image gets encoded. This means that for each image byte, we need 8 audio bytes, so the WAV data must be 8 times larger than the image data. In the example code for this method, a 256x256 grayscale image was embedded into an 8 seconds stereo drum loop.

Considering a generic height x width grayscale image, this means that we have to encode height x width bytes into the WAV file.

Also, if the sound is stereo, $wav\_data\_length = 2 \cdot num\_samples$.
So, $wav\_data\_length = num\_channels \cdot num\_samples$.

We also know that, for a WAV file, the following is true:

$$\frac{num\_samples}{sampling\_frequency} = \text{length } (seconds),$$

and for the embedding algorithm to succeed,

$$\frac{wav\_data\_length}{8} = \frac{num\_channels \cdot num\_samples}{8} \geq height \cdot width,$$

which finally leads to:

$$\frac{num\_channels \cdot length \cdot sampling\_frequency}{8} \geq \text{height} \cdot \text{width}$$

Therefore, if one needs to embed a Full HD (1920x1080) grayscale image, one needs a stereo WAV carrier of 188.1 seconds (~ 3.14 minutes), sampled at 44100Hz.

The extraction process for this embedding scheme is straightforward: read the first 4 bytes of the WAV data (to get the height and width of the image) and then just look at the LSB of each byte of the audio, sequentially building each byte of the image.

When playing back the altered sound file, there is virtually no audible difference. The waveforms can confirm:

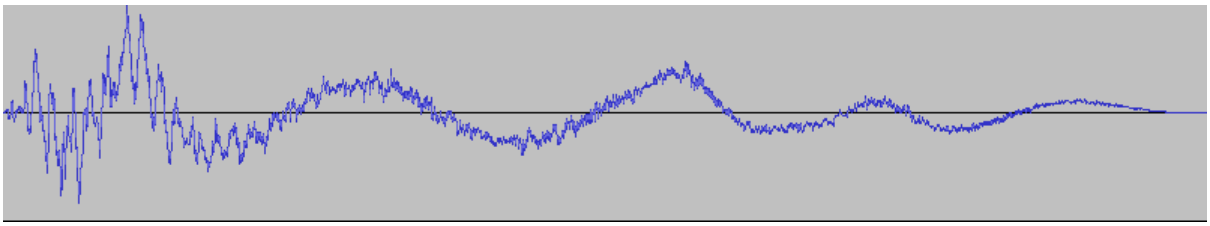

Fig1. Kick sound in the original drum loop

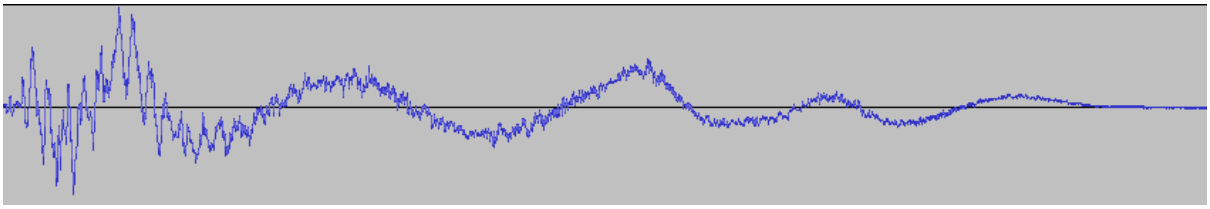

Fig2. Kick sound in the altered drum loop

However, there is a slight visible difference when inspecting the waveforms in the regions where there's no instrument playing (silence). Because of its very low amplitude, this difference is not audible, rather, it just shows some patterns of the embedded data.

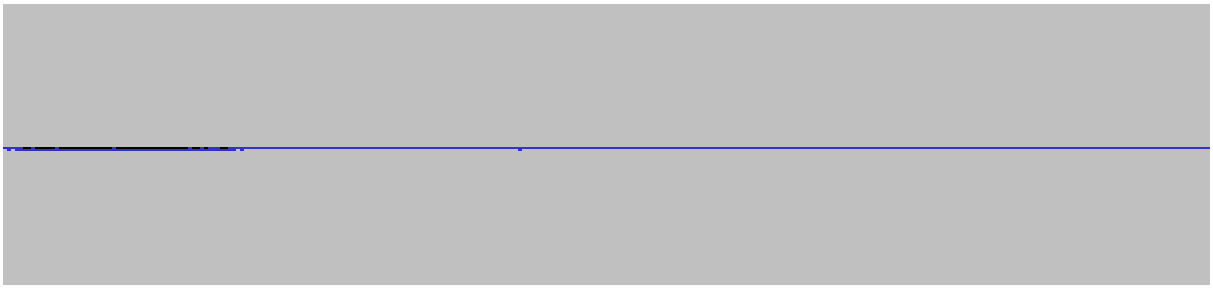

Fig3. Silence in the original audio track

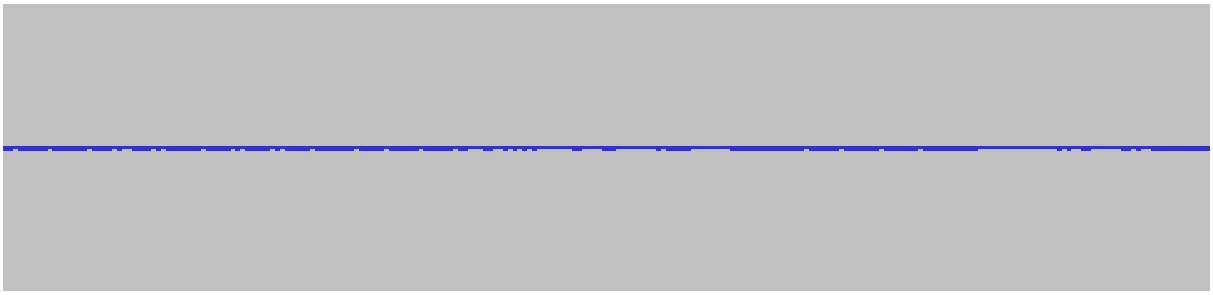

Fig4. Silent region from fig3. in the altered audio track

The Mean Squared Error (MSE) values between the original and the altered audio track are 1.0247e-05 and 1.0348e-05, and the Signal-to-Noise Ratio (SNR) values are 26.7870 and 25.3815 (for each stereo track, respectively). SNR formula is (where xi represents the original i-th sample).

$$SNR = 10 \times \log_{10}\left(\frac{\frac{1}{N}\sum_{i=0}^{N} xi^2}{MSE}\right)$$

To conclude this segment, LSB encoding is a fast and easy way to conceal data. I have presented a method to embed images in audio files, but this technique can also be applied to embed any (binary) data into a larger carrier. Depending on the data that is being concealed and on the carrier, custom embedding schemes can be implemented. One idea is to perform the embedding using a key that must be known by the communicating parties. This key might contain information about how the encoding is done.

Text-in-image

For this method, the building block (where the carrier data is an RGB image) is the 2D Discrete Wavelet Transform [4]. My implementation of this technique is based on two papers [1], [2], with the difference that the algorithm is slightly modified, but, nevertheless, the results are quite impressive, maintaining one of the most important aspect of steganography: imperceptibility.

This scheme works as follows:

First, the input text is converted into a binary stream (each char takes uint8 space; one can optimize for space, if only ASCII is used, therefore each char can take 7-bit space (0-127) and more data can be sent).

Next, the image is converted into YCbCr (luminance / chrominance) colour space. The reason for this is to allow the modification of underlying image data without perceptible visual differences. According to human visual system, human eyes are more sensitive to small changes in brightness of the colour, rather than changes in colour itself [1], [3].

Next, the Haar wavelet [5] is constructed and used to perform the Discrete Wavelet Transform on the Cb channel of the image. This results in 4 sets of coefficients: the approximation coefficients matrix CA and detail coefficients matrices CH, CV, and CD. In the algorithm, this coefficient matrices are labelled LL, HL, LH and HH respectively (L/H = low/high). The *approximation* of the image is represented by LL, and the details are represented by HL, LH and HH. Therefore, the altered image must contain the same LL coefficients. Changing HH and HL coefficients will not visually affect the image.

The text bits will be stored as the HH coefficients. If the size of the text exceeds the size of the HH matrix, then HL is also used. Same as in the previous segment, the size of the embedded text is stored as the first 4 bytes of HH.

Finally, the inverse DWT is applied on LL, HL, LH and HH and the result is the new Cb channel of the altered image (Y and Cr channels are kept intact), then, the image is converted back to the RGB color space.

To extract the text, one must convert the altered image to YCbCr colour space, apply Haar DWT and read the contents of HH and HL matrices.

This method is fast, and the results are impressive: no visual difference between the original image and the altered one, containing the text.

The data size constraints are relative to the size of the HH and HL matrices (height / 2 x width / 2). Assuming one needs 4 bytes to store the text length, the rest (sizeof(HH) + sizeof(HL) - 32) can be used to store the text. In general, it's sufficient to represent char_count as a uint32.

For each char, we need exactly 8 bits, so the relation is:

$$(8 * char\_count + 32) \leq \big(sizeof(HH) + sizeof(HL)\big)$$

$$(8 * char\_count + 32) \leq \frac{1}{2} \cdot \mathrm{height} \cdot \mathrm{width}$$

where the size of HH and HL depend of the wavelet transformation of the original image, and each bit is stored individually (because HH and HL contain relatively small numbers – for example, the mean of HH for a Full HD picture is -3.4144e-04, and for a 512x512 picture is -0.0089).

In the example for this technique, a 512x512 RGB PNG image was used as the carrier (sizeof(HH) = sizeof(HL) = 256x256). This means that the maximum size of the text must be 16380 chars. The entire process of encoding + decoding is very fast (encoding: 0.746 seconds, decoding: 0.058 seconds).

In conclusion, this method of encoding is extremely fast, but it is susceptible to some errors regarding the exact storage of data in HH and HL matrices. Depending on the image, it is possible that, for example, when applying

$$[\dots, \mathrm{HH}_{\mathrm{extracted}}] = DWT\big(IDWT(HH_{embedded}, \dots)\big),$$

$HH_{extracted}$ and $HH_{embedded}$ might differ.

Conclusions

To conclude this paper, I should say that steganography is a much more complicated field. I have addressed some simplified scenarios, where, for example, the robustness of the embedding technique is not considered. In real life situations, one must make sure that under some data alteration, because of various transmissions problems or lossy compression, one can still extract the concealed data with high integrity.

There are multiple approaches to steganography, where cryptography also plays a major role (if, for example, keys are involved). Also, to achieve the best results, one must take into consideration the heterogeneity of data (both the carrier and the concealed). As we have seen in the last segment, because of the changes in image (contrast, brightness etc), applying the transform on the altered data might not correctly reveal the concealed data, bit-by-bit.

Nevertheless, the methods implemented in this work provide some key insights of steganography and represent some interesting starting points for larger, more robust projects.

References

[1] Shinde, M. A. S., & Patankar, A. B. (2017). Image Steganography : Hiding Audio Signal in Image Using Discrete Wavelet Transform, (March), 331–334.

[2] Hemalatha, S., Acharya, U. D., & Renuka, A. (2014). Wavelet transform based steganography technique to hide audio signals in image. Procedia Computer Science, 47(C), 272–281. https://doi.org/10.1016/j.procs.2015.03.207

[3] Dr. Mahesh Kumar, ‖Image Steganography Using Frequency Domain‖, International Journal Of Scientific & Technology Research Volume 3, Issue 9, September 2014.

[4] https://en.wikipedia.org/wiki/Discrete_wavelet_transform

[5] https://en.wikipedia.org/wiki/Haar_wavelet