CSE 569 Lab 1 - Covert Channel

Erin Miller, Samatha Kodali, Yara Alsiyat, Funda Atik

February 2022

1 Covert Design Overview

The goal of the covert channel was to communicate text through morse-code flashlight videos. This was achieved using a video of a flashlight turning on and off for varying lengths. The first steps of this involved getting the frames of the video into image frames and then getting the brightness of each frame. The brightness of each image was then used to determine the length of consistent signals. These signals were then categorized into their corresponding morse symbols and then mapped to plaintext.

2 Implementation

We used the Morse Code Converter app, and followed the steps described in the assignment to create morse-code flashlight videos in the experiments. Our implementation is provided in detail below:

2.1 Identify Physical Signals

- **Read Video by Frame:** Our goal was to convert our morse-code flashlight video into image sequences. The images had different brightness depending on whether they were lit by the flashlight or not. We implemented the videos_to_images function, which takes an input video and outputs a directory with individual images corresponding to each frame of the video.
- **Extract Physical Signals** Our goal for this step was to determine if a particular image frame was lit by the flashlight or was dark by determining average brightness for each image. This is important for distinguishing which morse symbol is attempting to be communicated. We implemented the brightness method, which first converts all images to grayscale and normalizes their brightness. Then, we return the average brightness for each image. We used the suggested ImageStat library to perform these conversions and measurements.

- **Plot Physical Signals** Once we had the mean brightness for each image, we wanted to plot these brightnesses per frame. To do this we created an array that we filled with the mean brightness for each image and then we called the plot_brightness function we implemented on each of those.

2.2 Convert Physical Signals to Morse Symbols

- **Calculate Lengths of Consistent Signals** In order to start the process of converting the physical signal of the flashlight being lit or dark to morse symbols, we needed to determine how long the flashlight was consistently light or consistently dark. To do this we implemented the brightness_to_lengths function, which calculates the lengths of consistent signals by keeping a counter of how many frames in a row had the same state (either contiguous light images or contiguous dark images). A frame was determined to be light if its brightness was greater than the threshold parameter, and it was dark if its brightness was less than or equal to the threshold parameter. When the brightness transitioned from light to dark or dark to light, the counter was appended to an array called signal_lengths. A counter variable was used to represent the signal lengths, where the counter was incremented for contiguous light frames, and was decremented for contiguous dark frames. Once all the image frames had been cycled through, we returned the array signal_lengths, which contains the length of alternating darknesses/lights recorded.

- **Classify and Match Lengths to Morse Symbols** The next step of our implementation was converting the observed signal lengths into morse code symbols. We implemented the classify_symbols function to accomplish this, leveraging the suggested python wrapper of the ckmeans R package to classify each signal into one of five possible categories.

2.3 Convert Morse Code to Plaintext

The final step of our implementation was to convert the sequence of classified morse signals into plaintext and output a human readable version of the hidden message. We completed the morse_to_plaintext function, which takes this list of morse signals as input and returns the corresponding plaintext. We used the provided morse_to_letter dict to make these conversions. The final output of our program is shown in Figure 1.

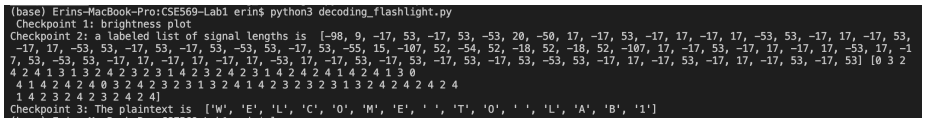


Figure 1: Output of running our completed decoding_flashlight.py script.

3 Experiments and Results

We tested our code with several recordings, including the provided recording and a variety of videos we took at a variety of distances and light conditions in order to get an idea of the limitations and variability of our program. For all cases except the original input video, we used the same encoded text, *"SOS SOS SOS"*. We encoded a relatively long and repetitive text because, for shorter text, our code may not detect all letters at the beginning and the end of the video.

3.1 Testing Given Input Video

We use the given encoded video input to test our code. For this experiment, the threshold is set at *140*. The generated the brightness per frame plot seen in Figure 2. The decoded plain text was *"WELCOME TO LAB1"*.

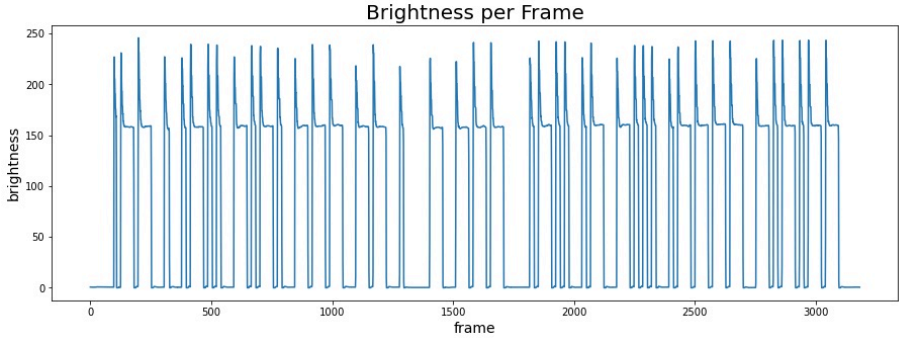


Figure 2: Mean brigthness per image

3.2 Varying Distances and Light Conditions

We tested our code in two indoor lighting conditions. First, we recorded the input video in a dark room, so there was a clear distinction between light and bright frames. Secondly, we recorded the input video in very bright light conditions. As a result, the mean brightness for all dark and light frames was higher than the dark setting. Moreover, we experimented with two distances for each lighting condition, namely far and close distances. For all test cases, the threshold is 140, and the encoded text is "SOS SOS SOS." Figure 3 shows the mean brightness per image for each test case.

Figure 3a was recorded in a dark and far setting. We got a *"Not a real letter"* error. The decrypted plaintext was empty. Figure 3b was recorded in a light and far setting. The decrypted plaintext was *"SAS ST S SR"*. Figure 3c was recorded in a dark and close setting. The decrypted plaintext is *"SWR SWR SW"*. The Figure 3d was recorded in a light and close setting. The decrypted plaintext was *"SAS STS SS"*.

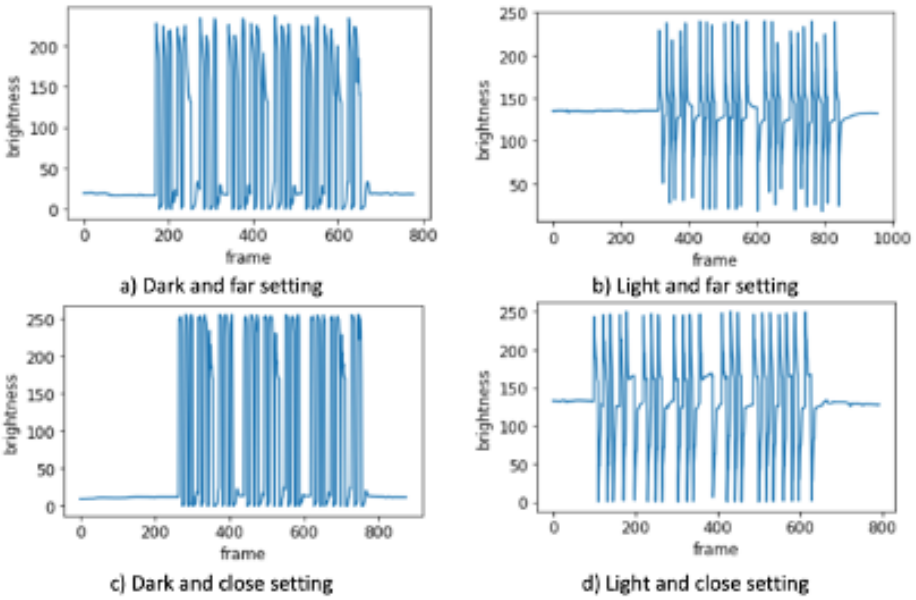


Figure 3: Mean brigthness per image for our test cases

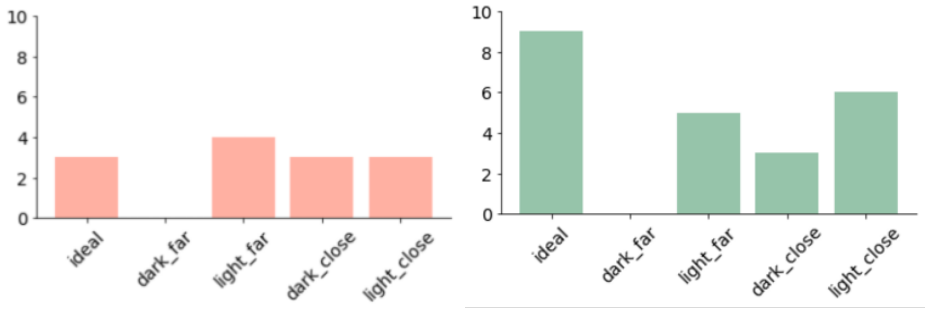
4 Evaluation

We design two evaluation metrics to compare different test cases. The first metric is the number of words guessed. We call it *WORDS_GUESSED*. The second metric is the length of the longest common subsequence after removing space characters between guessed words, namely *LONGEST_SUBSTR_NOSPACE*.

Figure 4 shows the performance of our code under various test cases in terms of the evaluation metric we choose. Surprisingly, for both metrics, our code perform worst in a dark and far setting. When the evaluation metric is chosen as *WORDS_GUESSED*, in a close setting, both dark and light conditions correctly guess the number of words. On the other hand, in terms of *LONGEST_SUBSTR_NOSPACE*, brighter lighting conditions perform better than the dark environment. When the setting is light, near scenes capture higher similarities regarding matching subsequent letters. All tests can be ran in the notebook.

5 Conclusion and Future Direction

In general, communication over covert channels is very difficult in practice, and can often be detected by monitoring system performance and covert channel analysis strategies. In addition, they suffer from a low signal-to-noise ratio and low data rates (typically, on the order of a few bits per second). In our implementation, we observed a wide variation in the performance of our implementation



(a) WORDS_GUESSED (b) LONGEST_SUBSTR_NOSPACE

Figure 4: Performance of test cases under two evaluation metrics

depending on the conditions of the video we filmed. Unless we recorded under very specific and ideal conditions, the output was mostly garbled and the meaning of the message was lost. We think this flaw could be patched over somewhat by further image processing before the morse pattern is detected, but we're not sure exactly what image processing this task would require.