CSE 569 Lab 1 - Covert Channel

Jacob Gilhaus and Ben Gilman

February 2022

1 Covert Design Overview - 3 pt

The goal of this covert channel is to exfiltrate messages from a building using visual signals. Our covert channel uses a smartphone Morse code converter application to encode and transmit Morse code messages through the use of the phone's flashlight.

In order to build this channel the transmitter must have a smartphone with the Morse Code Converter App installed, as well as access to a window through which the encoded message can be flashed to a receiver outside. The receiver must have access to a video recording device as well as our decoding script and the necessary dependencies. The method to extract the messages is to split the video into images. The brightness of the images is then used to determine whether the transmitter light is on or off, allowing us to construct the Morse code and decode the message.

2 Implementation and Experiment - 6 pt

Our GitHub repository is linked here

Since we decided to pursue the Morse code cyber-physical channel, there were skeleton functions written for us to develop into a working prototype. Here we can describe each step individually, including the goals of the step and how we achieved those goals.

1. Converting the Video into Frames

   The first step was to read in the video by passing in a path and produce frames that can be analyzed individually. To complete this step, we utilized the cv2 library, which produces a video object from which we could parse frame images. During the writing process we kept a count of the number of frames, which the `video_to_images` function returns.

2. Extracting Physical Signals

   Now that we have a number of frames to analyze, we need to extract the physical signals given in the video. The brightness function performs this task by leveraging the PIL Image and ImageStat libraries. We took the mean RGB value of the whole image using ImageStat.Stat(im).mean. To further distill the brightness, we averaged out the 3 values to produce a single brightness metric for each frame.

3. Convert Physical Signals into Lengths

   Up to this point we have successfully converted our video into frames, for each we calculate a brightness metric. We implemented a `brightness_to_length` function to interpret how long the signals were present. Since Morse code differentiates dots and dashes by how long the flashlight shines, we need to distinguish between these two signals. Our technique was to produce an array that reflected lengths of brightness and darkness as positive and negative values respectively.

   Another important part of determining lengths was to establish a threshold for what constitutes a bright/dark frame. We accomplished this by implementing a helper function `get_brightness_threshold`, which finds the maximum and minimum brightness values of the whole video (filtered through a 10 frame rolling average). This way we were able to more reliably measure what constitutes a bright and dark frame, measuring the threshold as precisely the average.

   Once a threshold is established, we can then create signal lengths by creating a new entry in the list for changes from dark to light or vice versa. If the frame is the same brightness category as the previous, we simply increment/decrement the previous entry to reflect consecutive frames of the same category ($1 \longrightarrow 2$ or $-1 \longrightarrow -2$). This technique allows us to return a list of signal lengths that accurately reflects the length of a light or dark symbol by the number of frames that evaluate to said category. We can also trim edge darkness by removing negative first and last values in the array since the message begins and ends with bright symbols.

4. Classify Signal lengths as Symbols

   Using signal lengths, we can then determine which symbol was intended, thanks to the rules of Morse code using nonconsecutive units (1,3,7). First, we calculated the number of frames in a single unit with another helper function. This is particularly helpful for changes in pacing when transmitting the message via Morse code. To calculate the length of a unit, the helper function `calculate_unit_length` simply takes the absolute value of each signal length and returns the most common occurrence (the mode). We can reliably determine the smallest unit this way since Morse code spaces out each signal in a letter with a single unit of darkness. Dots are a single unit of brightness, and together they constitute the most common symbols.

Once we can deduce the unit length, classifying symbols is simply a matter of establishing ranges. Anything less than 2 units is a 1 unit symbol, darkness being between letters and brightness being a dot. To find 3 unit symbols we used the 2-5 unit category, finding both dashes and spaces between letters. Finally, dark values larger than 5 units were the 7 unit darkness that reflects spaces between words. To reflect each symbol we can simply use the numbers 0,1,2,3,4 as suggested.

0: `space_between_words`

1: `space_between_letters`

2: `space_in_letter`

3: `dot`

4: `dash`

5. Parse Symbols and Convert from Morse Code to Plaintext

   Now that we have all of our Morse symbols, we simply need to collect dots and dashes until there is a space between words (or the message ends), effectively ignoring spaces between parts of the same letter. Once one of these occurs, we can convert the dots and dashes to a letter with the `morse_to_letter` dictionary. Next, we add the letter to the plaintext and reset the dots and dashes of the current letter to empty.

   Running through the list of symbols, we can produce a plaintext message that decodes the Morse code in the original video! We also recognized that any incorrectly determined letters would break the decoder. We introduced some robustness by detecting incorrect calls to the dictionary, indicating an invalid character with '[-]', and setting a flag that would wait until the next space character to continue attempting to decode.

Experiment

In order to continue to test our implementation of the Morse Code Cyber-Physical channel, we introduced some additional videos to challenge our techniques and help distinguish the strengths and weaknesses. Foremost, the original video was an extreme close-up of the flashlight, illuminating the entire frame and making the decoding process easy. In order to stress our implementation we introduced the following variants and their corresponding messages. Since we established a goal of exfiltrating the message from a building, we filmed some videos outside to best simulate this condition. All additional videos were an Android Galaxy S20 filming an iPhone XS Flashlight.

Tests:

- encoded (original): welcome to lab1!
- inside zoomed: J@C0B AND B3N
- inside unzoomed: h4ck3d
- inside unzoomed dark: CSE569!
- outside unzoomed close: s3cr3t
- outside zoomed far: s3cr3t
- outside zoomed close: pr1v4t3
- outside zoomed dark far: I'm in
- outside zoomed dark close: I'm in

Following the original video, we also tested using a video taken from across a room, zoomed in on the flashlight (zoomed). Next, we did the same without using the zoom feature on the smartphone (unzoomed). We also introduced other variations like turning off the lights inside (dark), taking the video from outside (outside), both on the sidewalk (close) and across the street (far).

These strategies help delineate important factors. Notably, our implementation was successful on the following variations:

- encoded (original): welcome to lab1!
- inside zoomed: J@C0B AND B3N
- inside unzoomed dark: CSE569!
- outside zoomed dark far: I'm in
- outside zoomed dark close: I'm in

Inside unzoomed was also close, but not entirely correct.

During the process of testing our implementation, we also had the capability to produce plots of the brightness by frame. These plots help us understand the factors in the environment as well as how successful our techniques were.

From these successes and failures we could understand more clearly the effect of the environment on our cyber-physical channel, mentioned in more detail in the next section. Of the 3 figures shown, our implementation was only successful for **Figure 1**. We were also successful on 4 other videos, but our failures most effectively illustrate our limitations.

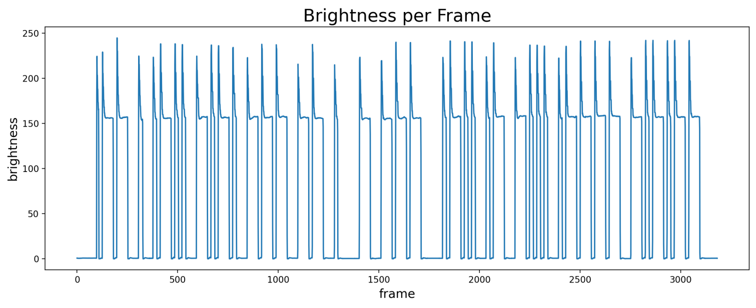

Figure 1: The graph for the original encoded.mov video

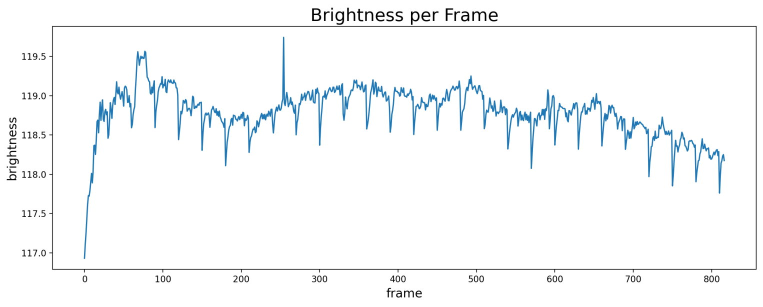

Figure 2: The graph for the outside, unzoomed, close video

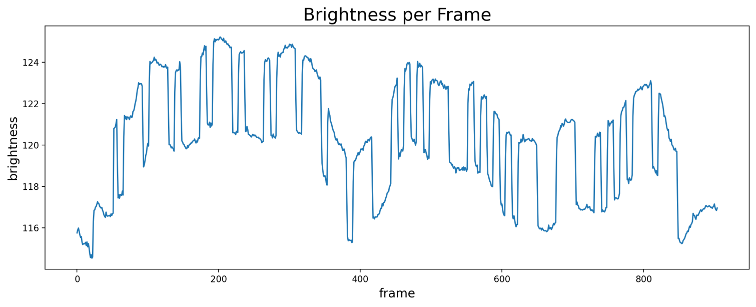

Figure 3: The graph for the outside, zoomed, close video

3 Conclusion and Future Direction - 6 pt

3.1 Limitations

Environmental Factors

The two main physical factors we explored were the physical distance between the light source and the camera and the amount of ambient light. The further the camera was from the light source, the less the effect of the light on average brightness of the frame was. This is due to the smaller percentage of the frame that the light fills at a distance; a higher number of brightened pixels would increase the overall brightness of the frame. This is illustrated by our "inside zoomed" example video decoding correctly while our inside unzoomed video did not.

Another effect of the distance is the increased effect of camera wobble on the result of the decoding. When the camera is zoomed, small changes in the camera angle due to the smartphone being held in a human hand were amplified. These small movements caused what was in the frame to change, for example, showing more or less of a brick wall. The changes in the overall brightness of the frame due to this were more significant than those caused by the light, and so the video was unable to be correctly decoded. This can be seen in the "outside zoomed close" example, shown in **Figure 3**. The peaks and valleys of the graph vary significantly, and these correspond to the wobble in the camera showing more or less of differently colored objects that affected the brightness more than the state of the light.

Lastly, the amount of ambient light is a major environmental factor. This can be seen in **Figure 2**, where the range of brightness caused by the state of the flashlight are small compared to the brightness of the entire frame. In situations where factors such as the distance or camera wobble made a message indecipherable in the light, it was able to be decoded in the dark under the same conditions. This is the case for unzoomed video indoors and all of our outdoor videos. In the dark, a smaller light source has more of an effect on the frame as it appears larger due to lens flare. It also shines on the objects around it, and their brightness is affected more than it would be during the day. Lastly, the adverse effects of camera wobble are diminished as the brightness variation of materials entering and leaving the frame is smaller. For example, bricks and wood appear to be a more similar color in the dark, so more of one or the other in the frame will affect the brightness less.

Equipment and Technology

All videos were recorded through a Samsung Galaxy S20, and all Morse Code messages were flashed by an iPhone XS flashlight. The zoom capabilities of the camera were insufficient at certain distances. A camera with better zoom lens would be able to fill more of the frame with the flashlight from farther away, allowing the covert channel to more easily overcome distance.

The iPhone flashlight is small, and doesn't have much of an effect from far away. A larger light would fill more of the frame and have the same effect as a better camera in combating distance. A brighter light would have the ability to more visibly change the brightness of its surroundings even with a lot of ambient light.

Our videos were recorded by hand and due to this were quite wobbly, affecting the quality of our transmissions. Equipment such as a camera tripod would help to mitigate the adverse effects on the stability of the video.

Our brightness threshold algorithm is quite simple and therefore not very robust. A better Morse Code parser would help in combating the effects of distance, camera wobble and ambient light.

3.2 Proposed Improvements

Our covert channel is limited by the equipment available and certain environmental factors.

The first and easiest improvement would be to take the videos with a higher quality camera with an appropriate zoom lens and tripod. This would mitigate the effects of distance and eliminate wobble. If able to be zoomed in enough, the light could fill the frame and eliminate the effects of ambient light as well.

An improved Morse Code transmitter that is connected to a larger and brighter light would help similarly. The light would be more easily seen from farther away and be more effective even in bright conditions. The difficulty here is to build such a system that can convert text to Morse Code the way our smartphone app is. However, this is not a very large hurdle to overcome.

The most complex improvement would be regarding our decoding software. Currently the brightness threshold is a constant value for the entire video. It is calculated finding the midpoint between the maximum and minimum brightness values. However, outlier peaks or troughs can significantly affect the threshold. The height of the peaks and troughs vary throughout the video due to camera wobble. The distance between the peaks and troughs can also be very small when the light has a small effect on the brightness of the frame, such as when at a distance. An algorithm that can adjust the threshold throughout the sequence of frames would be more robust, either by using a linear or quadratic fit rather than a constant, or by finding the locations of the individual peaks and troughs to determine when the light is on or off.

In conclusion, while our cyber-physical covert channel could have been more robust to limiting factors like camera wobble and ambient light, we were still able to successfully exfiltrate messages from a building using this technique. While there is much room for improvement, we introduced multiple adverse environmental conditions and still decoded 5 out of 8 messages correctly.